Henning Femmer

Requirements Engineering Artifact Quality: Definition and Control

Technische
Universität
München

Institut für Informatik
der Technischen Universität München

Requirements Engineering Artifact Quality: Definition and Control

Henning Femmer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende:	Prof. Dr. Susanne Albers
Prüfer der Dissertation:	
1.	Prof. Dr. Dr. h.c. Manfred Broy
2.	Prof. Dr. Martin Glinz, Universität Zürich

Die Dissertation wurde am 22.02.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.06.2017 angenommen.

Abstract

Requirements Engineering (RE) artifacts are central entities in the software engineering process. Based on these artifacts, project managers estimate effort, designers create architectures, developers build the system, and test managers set up a test-strategy. Consequently, quality defects in RE artifacts can cause expensive consequences in subsequent software development activities. Therefore, quality control of RE artifacts is key for successful software development projects.

However, quality control of RE artifacts faces two problems: First, the definition of RE artifact quality often remains incomplete, inadequate and imprecise. This problem concerns both the definition of valid quality factors as well as the claimed impacts of these quality factors onto projects. Second, in addition to the lack of a precise quality definition, we also struggle to assess any definition of quality in practice, since finding defects is labor-intensive and error-prone.

In summary, we have a limited understanding of what *high quality* RE artifacts are and need *more efficient methods* to control RE artifact quality in practice. This thesis makes two contribution regarding these problems: (1) The concept of activity-based quality models for RE artifacts and (2) a method to detect quality factors automatically.

(1) Addressing the former part of the problem, the first contribution of this thesis is a novel concept of artifact quality models for RE artifacts: *Activity-based Requirements Engineering Quality Models (ABRE-QMs)*. These models are based on the assumption that RE artifacts are rarely the ultimate goal of a software project, but a means for producing software effectively and efficiently. Therefore, ABRE-QMs define quality factors as properties of RE artifacts with explicit impacts on software development activities, which enables to define quality in a precise manner. We show how to create valid ABRE-QMs in multiple studies: Through interviews, we create an ABRE-QM based on a company's requirements guidelines; through a case study, we elicit various quality factors for requirements maintenance; and finally, through an experiment, we analyze one quality factor in depth at the example of the quality factor passive voice. In our experiment, we show the negative impact of this quality factor onto the activity of understanding requirements. During these studies, practitioners report increased validity and completeness of the quality definitions through ABRE-QM. Whereas the first study analyzes the applicability of ABRE-QMs, the latter two studies show how ABRE-QMs support precise reasoning about quality factors for RE artifacts.

(2) Addressing the latter part of the problem, the second contribution of this thesis is an efficient method to detect violations of an ABRE-QM in a given project: *Requirements smells* are those quality factors that may have negative impacts on activities, that have a concrete indication, and that have a lightweight (e.g. automatic) detection mechanism. We provide an analysis to which extent quality factors can be operationalized as requirements smells and develop an automatic requirements smell detection in a technical validation. In four case studies, we evaluate requirements smells and their detection, showing its potential to detect quality defects and raise the awareness for defects in practice.

This thesis provides academics and practitioners with a more precise understanding of RE artifact quality. In addition, this thesis indicates that certain violations of quality factors of RE artifacts can be detected more efficiently through automatic methods. These contributions enable practitioners to conduct quality control of RE artifacts more efficiently, and based on a well founded quality definition.

Acknowledgements

I would like to thank everyone who helped me on this journey. First, Prof. Dr. Dr. h.c. Manfred Broy, who created this special biotope at the chair for Systems & Software Engineering, where, from the very beginning, I could follow my own path and drive the topic how I considered sensible. This environment initiated my interests in research. I would also like to thank Prof. Dr. Martin Glinz for co-supervising this thesis and especially for helping me to keep up the quality on the last meters.

Furthermore, I want to thank the various companies who shaped my views and tools through advice, time and data: The partners from, among others, BMW, CQSE, Daimler, MAN Trucks and Busses, Munich Re, Siemens, Wacker Chemie, pliXos and Qualicen made me understand and focus on applicable research.

Many colleagues and students walked with me, partly by supporting me in case studies, partly by providing ideas, suggestions and, where necessary, headwind. I also want to thank my friends, who were supporting, even when I could not find the time. They were there when I needed them. You are awesome.

It is difficult to express how thankful I am for my family. Everything I am, stems from you.

Lastly, Kristina, you are the best companion I can imagine.

Publication Preface

The contribution of this thesis is based on the following seven papers:

A [FMM15]: Henning Femmer, Jakob Mund, and Daniel Méndez Fernández. It's the activities, stupid! A new perspective on RE quality. In *International Workshop on Requirements Engineering and Testing*, RET, pages 13–19. IEEE, 2015

D [FKV14]: Henning Femmer, Jan Kučera, and Antonio Vetrò. On the impact of passive voice requirements on domain modelling. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 21:1–21:4. ACM, 2014

E [FKSJ14]: Henning Femmer, Marco Kuhrmann, Joerg Stimmer, and Joerg Junge. Experiences from the design of an artifact model for distributed agile project management. In *International Conference on Global Software Engineering*, ICGSE, pages 1–5. IEEE, 2014

F [FMJ+14]: Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer. Rapid requirements checks with requirements smells: Two case studies. In *International Workshop on Rapid Continuous Software Engineering*, RCoSE, pages 10–19. ACM, 2014

G [FMWE17]: Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017

H [FHEM16]: Henning Femmer, Benedikt Hauptmann, Sebastian Eder, and Dagmar Moser. Quality assurance of requirements artifacts in practice: A case study and a process proposal. In *International Conference on Product-Focused Software Process Improvement*, PROFES, pages 506–516. Springer, 2016

I [FUG17]: Henning Femmer, Michael Unterkalmsteiner, and Tony Gorschek. Which requirements artifact quality defects are automatically detectable? A case study. In *Fourth International Workshop on Artificial Intelligence for Requirements Engineering*, AIRE, pages 1–7. IEEE, 2017

Furthermore this publication with major contributions as second author is also included.

C [BFE+15]: Mohammad R. Basirati, Henning Femmer, Sebastian Eder, Martin Fritzsche, and Alexander Widera. Understanding changes in use cases: A case study. In *International Requirements Engineering Conference*, RE, pages 352–361. IEEE, 2015

Under Review

In addition, these works with contributions are still under review. The author's drafts are included.

B [FV17]: Henning Femmer and Andreas Vogelsang. Requirements quality is quality in use – a novel viewpoint –. *Submitted to IEEE Software*, 2017

Further related work co-contributed by the author of this thesis

[MMFV14]: Daniel Méndez Fernández, Jakob Mund, Henning Femmer, and Antonio Vetrò. In quest for requirements engineering oracles: Dependent variables and measurements for (good) RE. In *International Conference on Evaluation and Assessment in Software Engineering*, EASE, pages 3:1–3:10. ACM, 2014

[MFME15]: Jakob Mund, Henning Femmer, Daniel Méndez Fernández, and Jonas Eckhardt. Does quality of requirements specifications matter? combined results of two empirical studies. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 144–153. ACM, 2015

[EVF16]: Jonas Eckhardt, Andreas Vogelsang, and Henning Femmer. An approach for creating sentence patterns for quality requirements. In *International Workshop on Requirements Patterns*, RePa, pages 1–8. IEEE, 2016

[EVFM16]: Jonas Eckhardt, Andreas Vogelsang, Henning Femmer, and Philipp Mager. Challenging incompleteness of performance requirements by sentence patterns. In *International Requirements Engineering Conference*, RE, pages 1–10. IEEE, 2016

[VFW16]: Andreas Vogelsang, Henning Femmer, and Christian Winkler. Take care of your modes! an investigation of defects in automotive requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ, pages 161–167. Springer, 2016

[ABBF17]: Luca Allodi, Sebastian Banescu, Kristian Beckers, and Henning Femmer. Identifying relevant information cues for vulnerability assessment using CVSS. In *Submitted to the International Symposium on Empirical Software Engineering and Measurement*, ESEM. ACM, 2017

Contents

CHAPTER 1

Introduction

Creating high quality software systems is a difficult and complex endeavour. Within this endeavour, the first step of a software engineering process is *Requirements Engineering* (RE). RE can be seen as a means to understand and define the needs of all relevant stakeholders. Due to its premier role at the beginning of the process, RE is unequivocally understood to be of high relevance on the quality of the resulting system and the process reaching it. Central to RE is the concept of a *requirement* as a capability or property of a system needed by a stakeholder [Gli14].

RE is eliciting, analyzing, documenting, validating and managing requirements. RE activities are commonly separated into five core activities with different goals for each activity [Poh10, pp.48-52]: Elicitation activities search for stakeholders, their goals and requirements. Analysis activities consolidate these goals and requirements into one system to be built (this includes negotiation activities). Documentation activities persist the results of this analysis. Validation activities control the quality of requirements artifacts and activities [Poh10, p.512]. Finally, management activities are concerned with planning and controlling requirements artifacts and activities through the system lifecycle.

Requirements are usually documented. Most projects document their requirements in the form of *RE artifacts*, such as free-text, use cases [JBR99] or user stories [Coh04]. For example, we reported on a case study [MFME15], in which all but one project self-reported that they document requirements (see Fig. 1.1). Other studies, such as by Mendez and Wagner [MW15] and also Mich et al. [MFNI04] further support this observation. However, the form, amount and tools used in documentation varies between rather traditional and so-called agile approaches. Nevertheless, also in agile approaches, such as Scrum [Sch04], the user story is considered the second-most important artifact next to source code.

RE artifacts are documented in natural language. Since RE artifacts usually serve as a medium for communication between stakeholders, RE artifacts must be represented in a form that all stakeholders understand. Therefore, RE artifacts are predominantly written in *natural language (NL)*: Based on a globally distributed

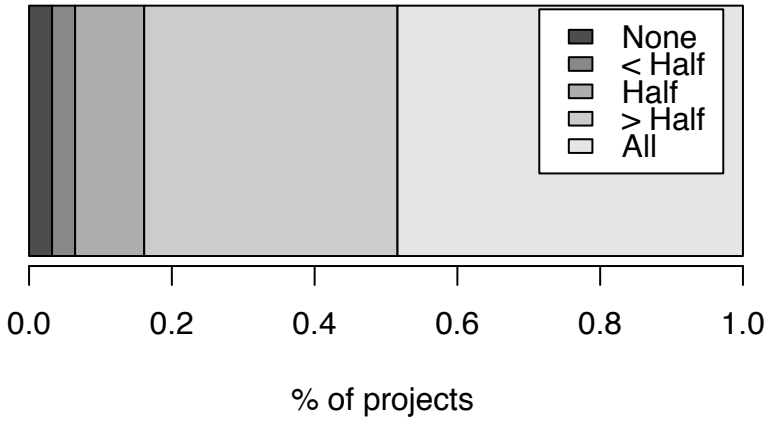


Figure 1.1.: The figure shows the self-reported extent of requirements documentation in practice, as reported in [MFME15]. The x-axis shows the share of projects that self-report to document *none, less than half, about half, more than half,* or *all* of their elicited requirements. The figure shows that less than 20% of the projects reported to document only half of their elicited requirements or less, and that more than 80% of projects document more than half, close to 50% document all of their elicited requirements. Data is taken from [MFME15].

survey with 151 industry participants, Mich et al. [MFNI04] report that 95% of the participants represent their requirements using natural language. Of these, 16% use structured natural language, e.g. through templates and forms and 79% apply "common natural language"[MFNI04].

RE artifacts are an important factor for project success. RE artifacts are central entities in the software engineering process. Based on these artifacts, designers create an architecture, developers build the system, test managers set up a test-strategy, etc. Consequently, the effect of a quality defect in an RE artifact multiplies across all activities that make use of this artifact. This importance is also commonly estimated to manifest itself in the costs, especially if defects are found late: In an early work, Boehm [BP88] claims a cost explosion of factor 50 to 200 on fixing defects during the RE phase against fixing them in the implementation phase (and often it is not even possible to fix them late, according to Lawrence [LWE01]).

To understand this impact in more depth, we conducted an experiment that indicated that defects in RE artifacts propagate to defects in tests. In the experiment, we could confirm the impact of previously injected requirements defects onto creating test cases [MFME15]. In our results, roughly every second tester propagated an injected, obvious RE defect into system tests. Even more drastically, when we introduced a defect that was more subtle, the defect was propagated by every tester in the experiment.

RE artifacts need quality control. Hence, projects employ *quality control* (QC) to prevent, detect or mitigate defects. However, as Mendez and Wagner [MW15] found in a large-scale survey, even though requirements engineers consider QC very beneficial, they find it very challenging at the same time. Furthermore, despite the importance of RE artifact quality, the status quo of RE artifact quality in practice is devastating: Practitioners report that many projects suffer from imprecise, inconsistent, or incomplete artifacts. According to the survey, these defects are major reasons for project failure, e.g. in terms of a software not fulfilling stakeholder needs

or projects being out of time or budget, as reported by Mendez and Wagner [MW15] and also analyzed in more depth in [MMFV14].

1.1. The Problems of RE Artifact Quality Control

Mendez and Wagner [MW15] furthermore report that despite the acknowledged relevance (i.e. that low-quality requirements are expensive and dangerous for project success), less than a third of requirements engineers perform dedicated QC of RE artifacts. We argue that this discrepancy is due to two major problems.

Problem Statement We have a limited understanding of what *high quality* RE artifacts are in a specific context, and need *more efficient methods* to control RE artifact quality in practice.

In the following, we explain these two problems in detail.

Problem 1: Definitions of RE quality and its characteristics are incomplete, inadequate and imprecise.

In RE, the definition of high-quality or good RE artifacts is often provided through normative references, such as quality standards or text books. Taking the currently most relevant RE standard (according to an analysis by Schneider and Berenbach [SB13]), the ISO/IEEE/IEC-29148 [ISO11b] (in the following: ISO-29148), as an example, one can see the various problems of such a normative reference.[1]

ISO-29148 characteristics are incomplete. ISO-29148 describes quality through a set of abstract characteristics (see Ch. 2.4.3.1). When analyzing the characteristics in detail, we see that there are two different types of characteristics: Some characteristics, such as ambiguity, consistency, completeness and singularity are factors that describe properties of the RE artifact itself. In the following, we call these factors *artifact-based properties*. In contrast, feasibility, traceability and verifiability state that activities can be performed with the artifact (we call these *activity-based properties*, see Fig. 1.2). This is a small, yet important difference: While the former can be assessed by analyzing just the artifact by itself, the latter describe a relationship of the artifact in the context of its usage. Yet this usage context is incompletely represented in the quality model: For example, why is it important that requirements can be implemented (*feasible* in the terminology of ISO-29148) and verified, but other activities, such as maintenance, are not part of the quality model? Therefore we argue that normative standards do not take all activities into account systematically, and thus, are missing relevant quality factors.

ISO-29148 characteristics are not context-dependent and thus inadequate. One could go even further and ask about the value of some artifact-based properties such as *singularity*. What is the purpose and reason behind such a property? In this case, usually, singularity is considered important especially in regulated environments, e.g. the medical sector, where regulators put the obligation on software producers to verify each requirement of the system. Identifiable, singular requirements ease such a process. But what if the requirements are not used as, e.g. a legal document, and not tested sentence by sentence? Does the property still benefit to the developers?

1 Please note that while we show the arguments along the ISO-29148 in this section, the same arguments can be made for other definitions of quality.

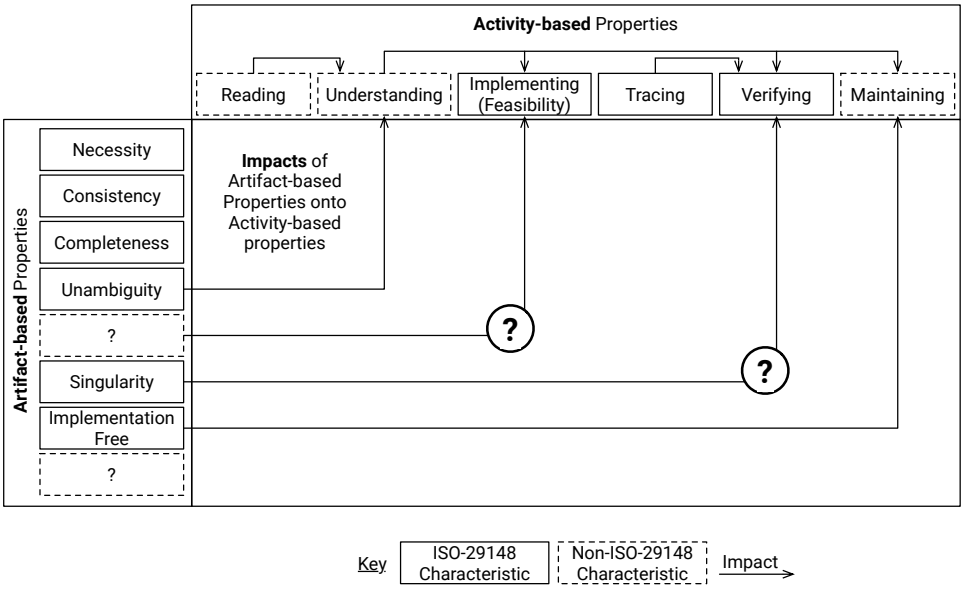


Figure 1.2.: This figure depicts the schematic relation between artifact-based and activity-based characteristics in ISO-29148. We argue that ISO-29148 both lacks characteristics, and also imprecisely defines why, or in which context, certain characteristics are important in RE artifacts.

A normative approach does not provide such rationales. This is different for activity-based properties, such as verifiability, since these properties are defined through their usage: If we need to verify the requirements, properties of the artifact that increase verifiability are important. If we do not need to verify this requirement, e.g. because we use the artifacts only for task management in an agile process, these properties might not necessarily be relevant.

This example illustrates that the apodictic and normative definition of quality is inadequate to the context-dependent nature of requirements quality.

ISO-29148 characteristics lack precise reasoning. For defining most of the aforementioned characteristics, the standard remains abstract and vague. For some characteristics, such as ambiguity, the standard provides a detailed lists of factors to avoid, the *requirement language criteria*. However, these criteria have an imprecise relation to the abstract characteristics mentioned above, and, consequently, the harm that they might potentially cause remains unclear. The list is preceded by the following statement.

> "Vague and general terms shall be avoided. They result in requirements that are often difficult or even impossible to verify or may allow for multiple interpretations. The following are types of unbounded or ambiguous terms:" [ISO11b, p.21]

The list contains nine concrete criteria (that we discuss and implement in Ch. 4.3), but the context and their reasoning remains vague: One example are

> "Negative statements (such as statements of system capability not to be provided) [ISO11b, p.21]

But when we confronted practitioners with this criterion and examples in their requirements, they rejected this criterion. To them, negative statements are very often not ambiguous, nor hard to understand. Especially system constraints have, by definition, a negative nature and might thus be even easier to define in a negative manner (see Publication F). For example, some companies define blacklists for certain libraries (e.g. a product cannot depend on libraries with known vulnerabilities) or languages (e.g. a product cannot use Adobe Flash). In case such a blacklist is given by the company, in order to avoid the negative statement, one would need to transform the blacklist into a whitelist. While this transformation is possible in some situations (e.g. defining which libraries or languages can be used), it is not practically maintainable in situation where the list is large or constantly changing, such as the aforementioned libraries with vulnerabilities. However, there are situations where a positive formulation is easier to understand than e.g. a double-negative. But was this the intended quality factor by the authors of the standard?

The given example indicates that discussing quality in this apodictic way disables both scientists and practitioners to analyze the validity of such a quality factor.

Summary For these reasons, i.e. the question of completeness, the implicit context-dependency, and the lack of validity, we argue that we need a more precise definition of RE quality. The quality model must be built upon quality factors and a profound understanding of their impact onto activities that are conducted with the artifacts. This way, researchers can falsify underlying assumptions and effects, and practitioners can objectively analyze whether or not (or in which context) they should enforce or neglect a certain quality factor.

Problem 2: Manual reviews for RE quality are expensive, slow and inconsistent.

To minimize the risk of quality defects, projects apply manual *quality assurance* (QA) techniques for RE artifacts, e.g. in the form of reviews [Sal13]. In fact, implementing any kind of reviews for RE artifacts is considered one of the key project success factors according to Hoffman and Lehner [HL01]. In most cases, this QA is performed in a manual fashion. However, this manual approach towards QA comes with three main challenges.

1. Manual reviews for RE quality are expensive. Manual reviews of RE artifacts require enormous amounts of manual labor. First and foremost, usually a large set of reviewers is required, since all stakeholders should be involved. To give an indication, in an unpublished study we analyzed a set of 74 review protocols showing that, on average, there were 8 reviewers involved. This problem was also reported as the core problem by the requirements engineers in interviews that we conducted in Publication H.

2. Manual reviews for RE quality create long feedback loops. When a requirements engineer hands in an RE artifact for manual review, this involves multiple people who must be coordinated and synchronized, including organization of meetings, the synchronization of results, the discussion of results, and, finally, the check whether defects were corrected.

In Publication H, we report that for a review it usually takes multiple weeks until the original author receives feedback. To illustrate just one consequence of these slow feedback cycles, the practitioners report that when they receive feedback, also the original authors need time to understand the content of the review again. We

argue that this is inefficient and frustrating for the requirements engineers, and might also lead to misunderstandings.

3. Manual reviews for RE quality lead to inconsistent results. In the current state of practice, quality definitions are often incorporated into guidelines. However, the produced RE artifacts very often violate the guidelines in various ways. For example, many RE artifacts contain passive voice requirements, even though best practice in literature, standards and also most guidelines discourage this (e.g. [Kof07, Lam09, ISO11b, Ter13]). We argue that this gap is due to shortcomings in QC: Reviewers do not constantly have the company guidelines and best practices in mind, but just review in an ad-hoc manner (see e.g. Katasonov and Sakinnen for a detailed comparison of reading techniques [KS05]). In order to make QC for a context-specific quality definition applicable in practice, we need a method that is able to check an RE artifact against a quality definition consistently.

Summary Manual approaches towards RE artifact QA are expensive, slow and inconsistent. Therefore, we must find more efficient methods to consistently control RE artifact quality.

1.2. Scope

The results of this thesis are set in a specific scope. The following assumptions enable to understand the results in its context.

RE quality: We focus on quality in the domain of RE. Approaches or processes that are not based on explicit RE phases, such as some agile SE approaches, are therefore out of scope of this thesis.

RE artifact quality: We focus on the quality of RE artifacts, which implies that the approach might not detect issues in the RE process. This decision is based on the assumption that RE quality manifests itself in artifacts, an assumption that we do not investigate in this thesis. In particular, the relation between process quality and artifact quality is still not understood in research [Men15].

Furthermore, all results presented here have the more impact on the project, the more it relies on RE artifacts. Yet, the purpose of RE, as explained in Fig. 2.2, is to understand the needs and constraints of the customer, to create agreement between stakeholders, to communicate the derived requirements, and to structure downstream engineering activities. These goals could be achieved without RE artifacts. However, RE artifacts support achieving these goals of RE, especially if a project is difficult to overview, for example because of its is large or complex in product structure (see e.g. [Kal13] for an analysis on such project characteristics). In these cases, projects can benefit from relying on RE artifacts to effectively and efficiently fulfil the purpose of RE. However, as discussed before, we found that most projects use RE artifacts for documentation (see Fig. 1.1).

This thesis focusses on projects where these or similar project factors are present and RE artifacts are considered a useful tool in the project.

RE artifact quality from an activity-based quality viewpoint: We focus on RE artifact quality from an activity-based quality viewpoint. In consequence, the focus lies on defects that are visible when using the artifacts. Other defects, e.g. if a RE artifact does not reflect the stakeholders' needs adequately, are not in the focus of this thesis.

Natural Language (NL) representations: According to Mich et al. [MFNI04], 95% of the participants of a global industry survey represent their requirements using natural language. Therefore, we focus on natural language and structured natural language. Specifically, we analyze natural language requirements captured in free text sentences, as well as use-cases [JBR99] and user stories [Coh04]. In terms of AMDiRE [MP14], we analyze the usage model elements and quality requirements.

However, we would argue that the concept of requirements smells can easily be transferred to non-NL and non-RE artifacts. We will discuss these in the outlook.

Automatic requirements smell detection: For the requirements smells, we focus on automatically detectable smells as we see the highest benefit-cost ratio here. This implies that some types of defects (e.g. completeness defects requiring deep insights into the domain or a strong understanding of the requirements semantics) are not subject of this thesis. However, we discuss which defects can and which defects cannot be detected by requirements smells in Chapters 4.4 and 5.

1.3. Contents and Relation to Previous Publications

This work is structured as follows: Chapter 2 summarizes the various notions of quality in software and requirements engineering, and describes fundamentals and related work. We conclude this chapter by identifying the existing research gap.

In Chapter 3, we define a thesis statement based on this research gap and derive research challenges, which we break down into research questions.

In Chapter 4, we summarize the results of this thesis, structured by the research questions. In Chapter 4.1, we propose Activity-based Requirements Engineering Artifact Quality Models (ABRE-QMs). In this chapter, we summarize the results of Publications A and B. Afterwards, in Chapter 4.2, we provide a methodical overview of different approaches for creating valid ABRE-QMs. In this chapter, we summarize Publications C, D, and E, as well as the empirical study of Publication A. In Chapter 4.3, we propose the definition of requirements smells. In Chapter 4.4, we analyze advantages and limitations of requirements smells by reporting on the results of our published studies. The results of both of these chapters are based on the Publications F, G, H, and I.

Chapter 5 summarizes and discusses strengths and limitations of the proposed methods in a larger context, before Chapter 6 summarizes this thesis and provides an outlook into future work.

CHAPTER 2

Fundamentals and Related Work

This chapter discusses the fundamental terms and concepts that we use throughout this thesis. This chapter furthermore provides an overview over the related work to identify existing research gaps.

Therefore, this chapter first explains key terms in requirements engineering. Second, we discuss the concept of quality, quality defects, and quality assurance. Third, we look into the role of quality defects in software engineering projects. Fourth, we summarize the current state in defining RE artifact quality. Lastly, based on this overview and also the overview provided in Publication G, we summarize the state of the art in QC for RE artifacts and analyze the existing research gaps.

2.1. Key Terms in Requirements Engineering

In the following, we define key requirements engineering terms and concepts and explain how we use these terms throughout this thesis. As the IREB combines various state-of-the-art views and is widely applied in industry, we base our definitions on the IREB Glossary [Gli14]. We extend the IREB view through the views of Pohl [Poh10] or explicitly adapt them to our own view, where we consider it required for this work.

Stakeholder. The central term in requirements engineering is the stakeholder. Extending the definition by Glinz [Gli14, p.20], we understand a stakeholder as a role, a person or an organization with a (direct or indirect) influence on a system. This explicitly includes persons or organizations who are impacted by the system (such as members of a company's works council) [GW07].

Requirement and Requirements Engineering Artifact. Using this definition of a stakeholder, Glinz [Gli14, p.17] defines a requirement as one of the following concepts:

1. A need perceived by a stakeholder.
2. A capability or property that a system shall have.
3. A documented representation of a need, capability or property.

In order to differentiate between the need or capability itself and the documented representation thereof, we refer with **requirement** to only the former two concepts and with **requirements engineering artifact** or **RE artifact** to the last item in the previous list. This includes commonly used terms such as requirements document or requirements specification. Examples of RE artifacts are use cases [JBR99] or user stories [Coh04].

System stakeholder and RE artifact stakeholder. In RE, when discussing the stakeholder, we usually refer to the stakeholder of the system (as defined above). However, when we discuss RE quality, we must differentiate between stakeholders of the system and stakeholders of the RE artifact. Whereas the former usually refers to roles impacted by the system, the latter refers to roles (or users) that influence the RE artifact or are impacted by the RE artifact. The latter includes not only the system stakeholders (who often have to inspect, discuss and accept the RE artifact), but in addition also testers, requirements engineers, legal departments or any other person, who, due to the software engineering process applied, has to (potentially) interact with the RE artifact.

Requirements Engineering (RE). RE can be understood as an approach towards the creation of RE artifacts. Yet, depending on the viewpoint, RE can be defined with a focus on process, stakeholders, or risks:

> A systematic and disciplined approach to the specification and management of requirements with the following goals:
>
> 1. Knowing the relevant requirements, achieving a consensus among the stakeholders about these requirements, documenting them according to given standards, and managing them systematically,
> 2. Understanding and documenting the stakeholders' desires and needs,
> 3. Specifying and managing requirements to minimize the risk of delivering a system that does not meet the stakeholders' desires and needs.
>
> [Gli14, p.18]

Please note that these goals are not mutually exclusive.

Requirements Engineer. The requirements engineer is the role which organizes and executes requirements engineering activities (see next paragraph).

Requirements Engineering Activities. In order to create RE artifacts and to achieve the goals of RE, as defined before, we usually define a set of required core activities. The previous definition for RE already hints at these key activities of RE, as visualized in Fig. 2.1. We extend these with other established categorizations [DAEE08, Poh10].

Elicitation: "The process of seeking, capturing and consolidating requirements from available requirements sources." ([Gli14, p.17]) Sources in this context include not only querying or observing stakeholders (in particular users), but also analyzing existing documents, systems or prototypes, or deriving requirements from higher level goals. The elicitation activities results in a set of, potentially contradicting, not necessarily documented, requirements.

Analysis: (including **negotiation**): The activity of understanding requirements and their interconnection in order to create an agreement between all stakeholders one set of non-contradicting requirements for documentation. These activities

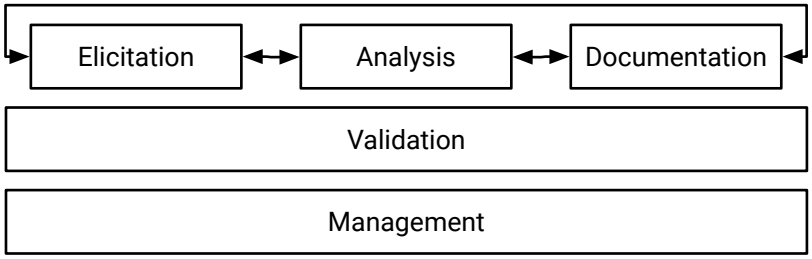


Figure 2.1.: The figure shows the core activities of RE, adapted from Glinz [Gli14], Pohl [Poh10] and Doerr et al. [DAEE08].

result in a consolidated, consistent (i.e. not contradicting) set of requirements for the given system.

Documentation: The documentation activities take this consistent set of requirements and represent them in RE artifacts. For these activities, the requirements engineer chooses a representation (such as goal models [Lam09], use cases [JBR99] or user stories [Coh04]). The documentation phase results in an RE artifact.

In addition, there are further cross-cutting activities that interplay with the previous activities at various levels:

Validation: Glinz defines validation as "The process of checking whether documented requirements match the stakeholders' needs." ([Gli14, p.23]) In our understanding and also in the understanding of Pohl [Poh10], validation is checking whether the content of the RE artifact matches the system stakeholders' needs, as well as whether the content and representation of the RE artifact matches the RE artifact stakeholders' needs. As such, validation also contains quality control activities.

Management: Requirements management, according to Pohl [Poh10], refers to three areas: First, maintaining the RE artifacts across the life cycle (including filing RE artifacts, prioritizing requirements, change management, and other activities). Second, organizing and controlling RE activities. And third, observing the system context, in order to detect changes that require further RE activities.

Please note that, although these activities are consecutive in their nature, they are usually executed iteratively in order to incrementally produce RE artifacts.

This thesis focusses on RE artifact quality, and therefore on the output of the documentation phase. Since we focus on the improvement of the RE artifact quality, we consider the activities, techniques and tools that we propose as part of the validation activities.

2.2. Quality and Requirements Quality Fundamentals

In the following, we dig deeper into notions of quality and specifically, define quality for RE artifacts. We base these definitions on the works by Deissenboeck [Dei09], Wagner [Wag13], and Juran [JB98] .

Quality. Originally, the term *qualitas* refers back to Aristotle [AriBC] and his work on categories. Aristotle defines qualities as properties or attributes of entities. This viewpoint as quality being characteristics free of judgement or evaluation has in

between changed in both common language and engineering into the quality of an object being a judgement of goodness.

Quality is a complex and multi-faceted concept [Gar84]. To understand the different facets of this goodness, Garvin [Gar84] identified five different viewpoints on product quality. They are translated to software engineering by Kitchenham and Pfleeger who summarize the views as following [KP96]:

- The *transcendental view* sees quality as something that can be recognized but not defined.
- The *user view* sees quality as fitness for purpose.
- The *manufacturing view* sees quality as conformance to specification.
- The *product view* sees quality as tied to inherent characteristics of the product.
- The *value-based view* sees quality as dependent on the amount a customer is willing to pay for it.

Depending on the situation, one or multiple of these viewpoints come into play. All in all, these viewpoints serve as generic concepts of quality, which guide the understanding and definition of a concrete instance of quality.

A common definition of quality stems from the ISO 9000 [ISO05]. Here quality is defined as "the degree to which a set of inherent characteristics fulfills requirements" ([ISO05, p.7]). In the remainder, the ISO 9000 family has a process-oriented focus.

In a simpler definition, Juran [JB98, pp.27-28] understands quality as having mainly two meanings: First, he understands quality as features of products which provide customer satisfaction. Second, he understands quality as *freedom-from-deficiencies* in terms of field errors, customer dissatisfaction, etc. In this sense, high-quality products lead to less costs than low-quality products. In the following, we will built upon this understanding of quality, which will lead us to a definition of RE artifact quality defects.

RE Quality. Following the definitions of the goals of RE as understood by Glinz [Gli14, p.18], we understand quality in RE as the degree to which the following goals are sufficiently fulfilled for system stakeholders as well as the project team (see Fig. 2.2):

① Understand stakeholders' needs: In our understanding, high quality in RE is the degree of correct and complete understanding of the goals, expectations and constraints of the system stakeholders.

② Achieve agreement: In addition, high quality in RE is the degree of agreement on a system that manifests the consensus of all system stakeholders. To this end, high quality in RE correctly prioritizes requirements, and ensures that we derive a best-possible solution for the system stakeholders' needs (iteration between problem and solution space, see twin-peaks model [Nus01]).

③ Create the same mental model between all system stakeholders: Furthermore, high quality in RE is the degree to which these system stakeholders' needs and the consensus is correctly and completely communicated between involved system stakeholders in the project.

④ Structure & manage requirements-based activities: Lastly, many project activities are structured along the system stakeholders' needs, e.g. in the form of requirements. Some exemplary activities are estimating costs and schedule of the system, developing the system or testing the system. Consequently, high quality in RE is the degree to which engineers working with the requirements (i.e. the information) can efficiently and effectively use the requirements to execute their requirements-based activities. This can include being able to handle changing requirements over time, if necessary in the project.

Combining this view with the understanding of Juran [JB98], we understand *high quality RE artifacts* as RE artifacts that are free of RE artifact quality defects, which impair the aforementioned goals. Before providing a more precise definition of RE artifact defects, we first need a notion of activity-based quality and quality models.

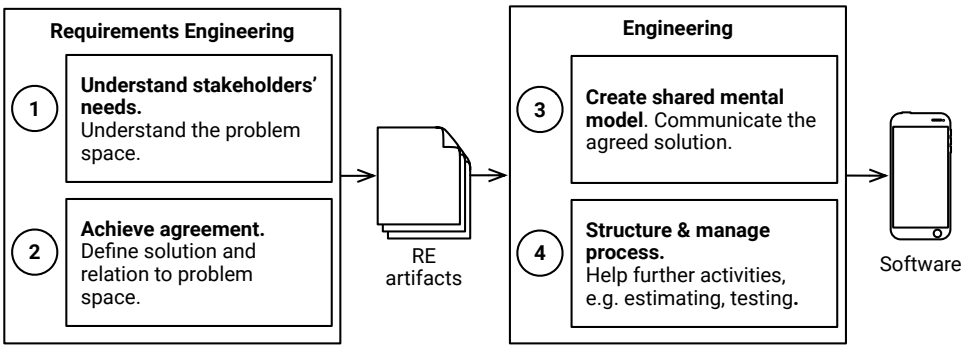


Figure 2.2.: The figure explains quality through the goals of RE in the RE phase and usages of RE artifacts in the engineering phase of a project.

Quality (Definition) Model. We furthermore need a definition of quality. Usually, this complex concept is approached through so-called quality models: A quality model is "a model with the objective to describe, assess and/or predict quality." ([Wag13, p.22])

Deissenboeck et al. [DJLW09] differentiate quality models to the DAP classification, according to the three purposes *definition, assessment*, and *prediction*:

- Definition models aim at decomposing quality into characteristics, and specifying these characteristics, usually in some sort of taxonomy. Examples for such a quality model for software is the ISO/IEC 25010 (SQuaRE) [ISO11a], or the aforementioned ISO 29148 [ISO11b] for RE.
- Assessment models, in contrast to definition models, not only require a definition of quality, but aim at measuring this quality in order to provide a metric for the quality of the object under analysis. An approach towards such quality models with the focus on source code is the EMISQ approach [PGH+08]. In RE, the approach by Davis et al. [DOJ+93] or the model by Lucassen et al. [LDBvdW15] represent assessment models.
- Prediction models, extending assessment models, additionally aim at predicting future characteristics, such as defect proneness etc. of the system. In software engineering, reliability growth models [Lyu96] or the quality assurance cost optimization models by Wagner [Wag07] represent such models. However, their prediction abilities are still limited [FN99, CD09, MK09, Wag13]. To all our knowledge, there are no prediction models for RE or RE artifact quality.

In addition, Deissenboeck and Wagner [DJLW09, Wag13] define multi-purpose models, which serve all three purposes.

Activity-based Quality As we analyze for RE quality in the ISO 29148 in Chapter 1.1, existing quality models often do not differentiate between properties of the artifact (or *product factors*) and the activities that are conducted with the artifact. As Deissenboeck et al. [DWP+07, Dei09] point out, this leads to inadequate quality models, since the former are objective and the latter depend on the specific context.

As a consequence, Deissenboeck et al. [DWP+07] explicitly associate system properties with the activities carried out during maintenance. This laid the foundation for the concept of activity-based quality. In this activity-based understanding of quality, quality is defined as properties of the system, which have a positive or negative impact onto the activities, e.g. maintenance activities, that are conducted with the system.

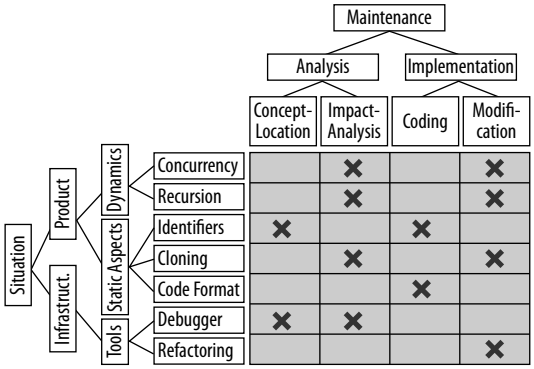


Figure 2.3.: This figure shows the maintainability matrix defined by Deissenboeck et al., taken from their work [DWP+07].

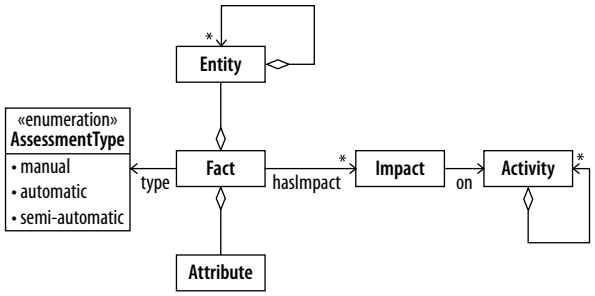


Figure 2.4.: This figure shows the meta model as defined by Deissenboeck et al., taken from their work [DWP+07].

Activity-based Quality Model (AB-QM). Deissenboeck proposed the concept of model-based quality control [Dei09]. In model-based quality control, a central stakeholder (the *quality engineer*) designs a quality model, based on which source code is created and evaluated.

Deissenboeck et al. [DWP+07] then suggest to combine model-based quality control with the concept of activity-based costing [Dru92, Jon96] in order to define maintainability. Activity-based costing is a strategy that enables to define the costs of a product through the various activities to be performed. As a result, activity-based costing can be used to optimize these costs. Through this combination of model-based quality control and activity-based costing, Deissenboeck provides a notion of the (maintainability) quality through its costs, by use of the impact of various product factors onto the maintenance activities (see Fig. 2.3). They manifest these core ideas of AB-QMs in a meta model (see Fig. 2.4). In this meta model, a fact has a set of impacts onto a hierarchy of activities. Furthermore, the fact is related to a set of entities ("objects we observe in the real world" ([KP96])) and attributes

("the properties that an entity possesses" ([KP96]). Lastly, the facts can be assed manually, automatically, or semiautomatically.

This idea was generalized by Wagner et al. [WDW08]. They proposed to integrate activity-based quality models into RE. The core idea is to be able to more precisely specify quality requirements through an activity based quality model. This can then be used to verify that the source code fulfills these requirements. Furthermore, Lochmann [Loc13] instantiated and extended this idea and provided an activity-based model for assessing the quality requirements in source code. Please note that these works define an AB-QM to verifying quality requirements in source code, whereas we focus on defining the quality of the RE artifact itself.

Wagner et al. summarized these works within the Quamoco project [WLH+12]. Based on an explicit meta model and the ISO 25010 [ISO11a], the Quamoco model unites over 200 product factors and 600 measures for Java and C# systems. Quamoco furthermore provides tool support for evaluating code against the Quamoco model. In an empirical analysis [WLH+12], they compare the tool's assessment of five open source systems against expert opinions. Their study shows that, at least regarding maintainability, the tool assessment matches expert opinion.

RE Artifact Quality Defect. This understanding of activity-based quality allows us to more precisely define quality defects. An RE artifact quality defect (in the following short: **defect**) is an instance of a violation of what is defined as high quality by a quality model. In our understanding, a defect is an instance of a quality factor that negatively affects one or more activities (see Fig. 4.7 in Chapter 4.3.2.1 for details).

This enables to refine our previous definition of high-quality: A high-quality RE artifact is free of RE artifact quality defects (similar to Juran [JB98]). RE quality defects are instances of factors of a concrete system, which negatively affect activities to be conducted with the artifact. Therefore, a high quality RE artifact is efficient and effective to use.

Process and Artifact Quality. In engineering, we usually aim at improving the product or *artifact quality* (such as the quality of the source code or the RE artifact). However, there are some approaches that focus on improving the quality of the process which leads to these artifacts, i.e. improving the *process quality*. The basic assumption behind taking this path is that the product quality is determined by the process quality. Following this line of thought, high quality processes would then lead to high quality products [Wag13]. However, as Sommerville [Som11, p.707] points out, these views stem out of the domain of manufacturing, where the product follows a straight engineering process. In contrast, software engineering, is much more of a creative architecture and design process, where other factors, such as the development technology, people, and project variables, such as cost, time and schedule, also significantly influence the resulting product quality (see Fig. 2.5). Jones [Jon00] analyzed the number of defects delivered by companies with various CMM levels, which is the predecessor of the now widely-distributed process quality assessment CMMI [CMM06]. He found that, while the average number of defects decrease with higher CMM level (i.e. higher process quality), the best of class CMM 1 (worst process quality level), still outperform the worst of CMM class 5 (best process quality level) [Jon00, Wag13]. This means that even companies with the worst process quality can outperform companies with highest process quality.

All in all, the relationship between product and process quality is still unclear. Therefore, in the following, we focus on product quality itself. However, in the

outlook, we discuss extensions of this work beyond product (or artifact) quality (see Chapter 6.2).

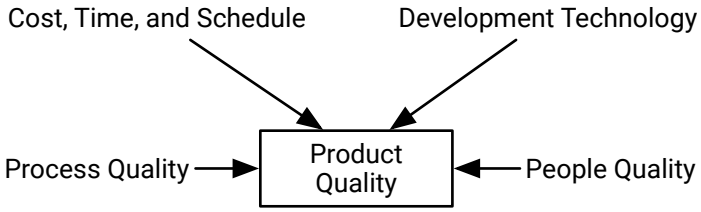


Figure 2.5.: This figure shows process quality as one factor deciding product quality. Taken from Sommerville [Som11, p.707].

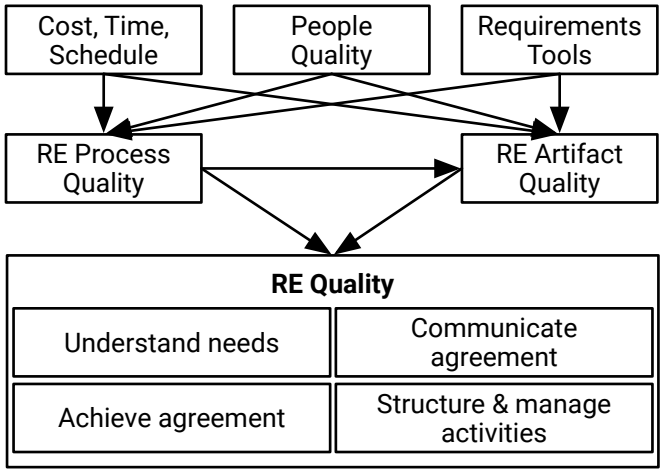


Figure 2.6.: This figure shows the relation between various context factors, RE quality, RE process quality and RE artifact quality.

RE Quality, RE Process Quality, and RE Artifact Quality. Consequently, we assume that RE process quality influences the quality of RE artifacts, just as do other context factors, such as cost, time, and schedule, people or tools.

Yet, the goals of RE, as explained in Fig. 2.2 can be achieved without RE artifacts: We could discuss and create a shared understanding without creating documents[1]. After all, some teams successfully develop software with none or only few RE artifacts.

However, RE artifacts have a supporting role reaching the goals of RE and during the following engineering activities in a project. RE artifacts force the writer to make a thought explicit, it enables to persist thoughts and agreements, and it furthermore enables communication that is dispersed in time and location.

Consequently, various project factors increase the impact and relevance of RE artifacts. These factors include the size of the project in terms of number of people, longevity of the created solution, local dispersion of the team, temporal dispersion of the team, legislative constraints, etc.

1 For a detailed analysis of this, see the analysis by Glinz and Fricker [GF15].

RE Artifact Quality Assurance. In ISO 24765 defines quality assurance (QA) i.a. as "a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements" ([ISO10]). More specifically, Wagner notes that QA "includes all technique we use to build products in a way to increase our confidence as well as techniques to analyse products" ([Wag13, p.20]). In consequence, QA can be either *analytical* ("analysing the state of the quality of a product" ([Wag13, p.20])) or *constructive* ("constructing a product in a way so that it meets its quality requirements" ([Wag13, p.20])).

In ISO 9000, quality assurance ("providing confidence that quality requirements will be fulfilled" ([ISO05, p.9])) is part of quality management, which is understood as "coordinated activities to direct and control an organization with regard to quality" ([ISO05, p.9]).

RE Artifact Quality Control. The terms quality assurance and quality control are difficult to distinct [Wag13, p.19]. Based on the definition by Deissenboeck [Dei09, p.64] for software products, we understand quality control (QC) for RE artifacts as three-faceted: RE artifact quality control is a process to

- analyze the quality of an RE artifact,
- compare it to the quality model, and
- take the necessary actions to correct the difference.

In this regard, both QA and QC discuss analytical assessment of artifacts. Therefore, some authors, such as Deissenboeck [Dei09] consider QC to be one part of QA.

In the following, we will use the term quality control (QC) when we discuss our RE artifact quality improvement process or when focussing on the processes of analytical QA. We will use the terms automatic, manual, analytical or constructive QA, when we refer to concrete techniques, such as reviews or automatic analyses.

2.3. RE Artifact Quality Defects in the Software Engineering Process

The sources and consequences of RE quality defects strongly depend on the process in context. In the following, we want to briefly discuss sources and consequences of RE artifact quality defects.

2.3.1. RE Artifact Quality Defect Sources

In order to illustrate potential sources for RE artifact quality defects, consider the idealized and generic SE process depicted in Fig. 2.7. In this process, the requirements engineer elicits requirements from a previously elicited set of system stakeholders. The requirements engineer then analyzes these requirements until a coherent view on the system emerges. The requirements engineer documents the requirements in the form of an RE artifact. This artifact is used first by the project team, which creates and ships a product, based on this information. This product is used by various users. In addition, the product owner[2] compares the shipped product against the RE artifact in order to determine whether the shipped product is indeed what he ordered, i.e. whether the system fulfills the requirements defined in the RE artifact. The product owner can be the same person as the customer (i.e. the role that pays the bill at the end), but can also be someone who is put in charge

2 Please note that the product owner role is not the same role as the user role. Whereas the latter interacts with the system, the former defines its goals and makes scoping decisions.

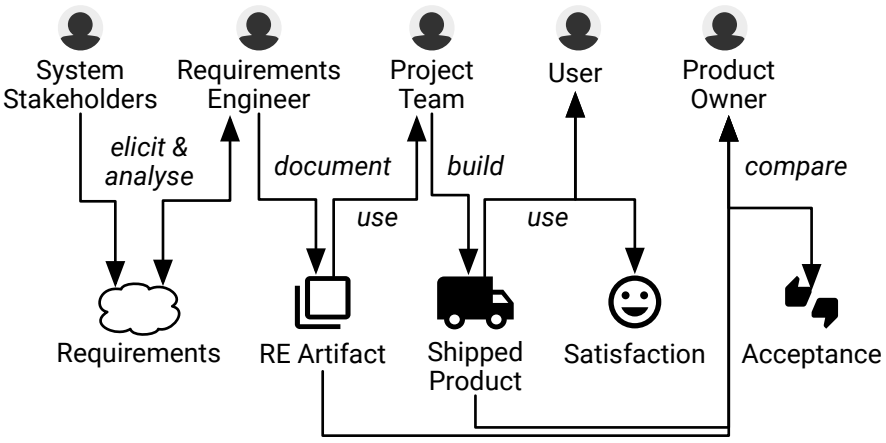


Figure 2.7.: This figure depicts an idealized model of RE artifacts in the software engineering process.

by the customer. The idealized process illustrates that the RE artifact can contain defects from one of the following two sources:

Defects introduced in elicitation & analysis: Either the defect existed independent of the documentation and the requirements engineer merely documented the (already existing) defect. Examples of these defects are incorrect or infeasible requirements or also requirements where functionality, quality requirements or system constraints are missing (i.e. *complete set of requirements* in the terminology of standards, see also Section 2.4.3). These defects root from issues in the RE elicitation, analysis or, even before, incorrect assumptions of the system stakeholders.

Defects introduced during documentation: If the defect is not related to the requirement itself, it must be related to the RE artifact (its documentation). In this case, since the only purpose of the RE artifact is usage, we refer to an unusable requirements artifact. Unusable requirements artifacts can violate a broad set of activities. In general, some of the common issues are unreadable, ununderstandable, unverifiable, or unmaintainable requirements artifacts. There can be many reasons why these artifacts are unusable. Common reasons include incompleteness of individual requirements (i.e. the information necessary to execute the activity are not present, see e.g. [EVF16, EVFM16] for details), ambiguous or inconsistent requirements documentation. We discuss common forms of usage, and the consequences during usage in more depth in the next section.

In this work, we mostly focus on the latter type of issues. However, when we face an issue in an RE artifact, these two categories blur. For example, an incomplete RE artifact can stem from both the documentation and also elicitation defects.

2.3.2. Consequences of Low RE Artifact Quality

Since RE is inherently a social discipline and involves complex psychologic processes, the consequences of RE quality defects are also complex and difficult to predict. In addition, since RE is only a means to an end, we have to understand not the direct, but the indirect impacts of RE quality, such as project success. Accordingly, some research [MW15, MMFV14] exists that analyzes and systematizes the complex cause and effect chains of RE. However, we are still far away from a thorough

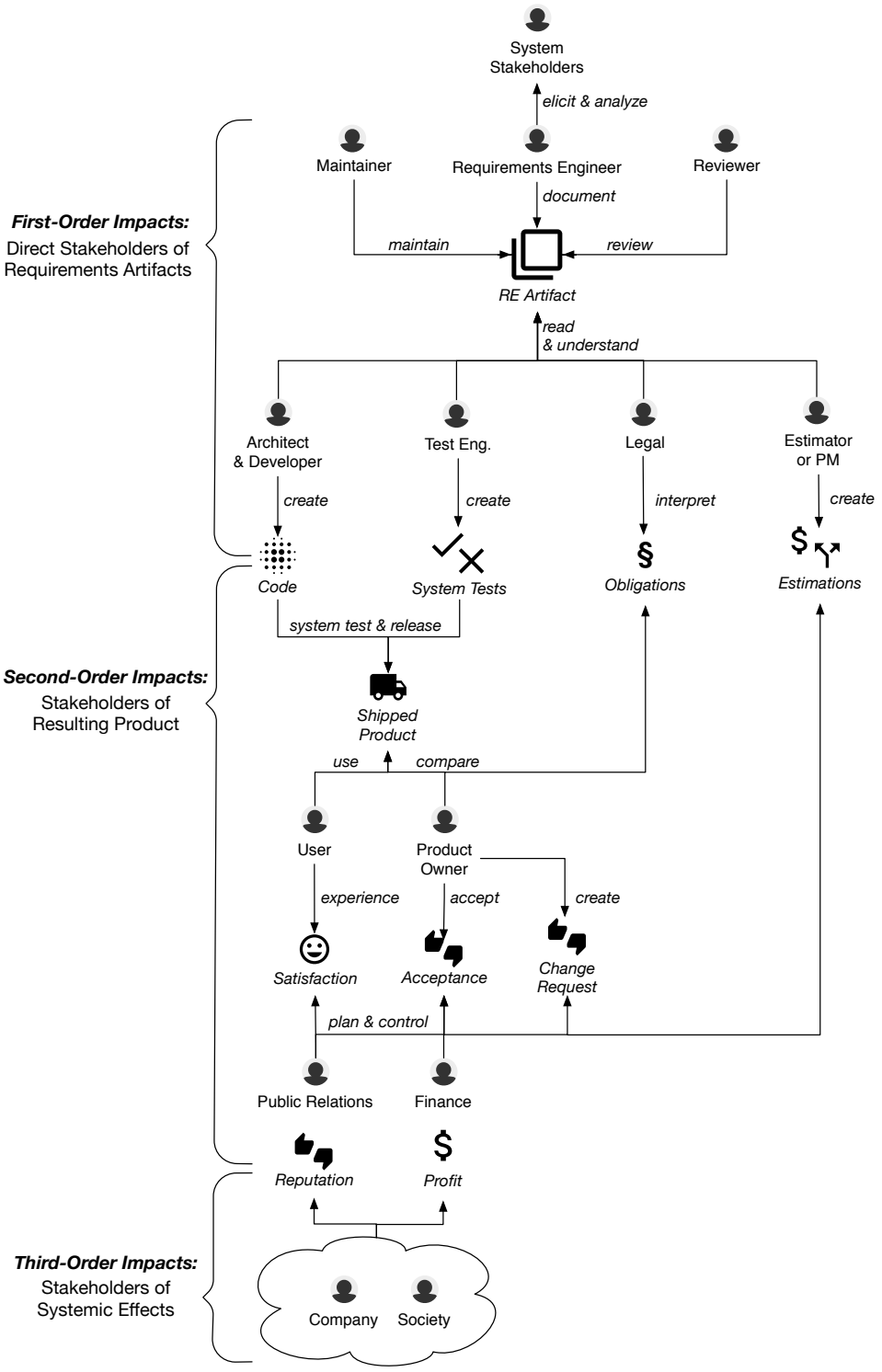


Figure 2.8.: This figure depicts potential consequences of defects in an RE artifact, starting from direct users (first-order impacts), going over indirect users (second-order impacts), to systemic or third-order impacts.

understanding thereof. In the following, we illustrate the potentially devastating consequences of bad RE artifact quality and also the complexity of the matter, through an exemplary chain of events. For this, we extend the discussion initiated by Gorschek and Davis [GD08], and also extended by ourselves [MMFV14].

In Fig. 2.8, we detail the activity of building a shipped product from the previous process in Fig. 2.7 and extended the process beyond the user and product owner. We can categorize the impact of RE artifact quality into three levels: Based on the definition by Hilty [HSW05], we call these impacts first, second- or third-order impacts, depending on the distance between RE defect and the impact. In addition, some impacts lead to problems during follow-up activities (impacts on *effectiveness*) while other lead to additional efforts (impacts on *efficiency*).

In the following, we assume that the requirements engineer unknowingly used ambiguous phrasing during documentation. Thus, in the terminology described above, this is a defect that hinders using the RE artifact.

First-Order Impacts: The direct impacts of ambiguous phrases in RE artifacts in an idealized process are `Maintainers`, `Reviewers`, and the various users. All of these are `Readers` of the artifact. Within this role, the defect impacts the efficiency of reading the artifact. Regarding effectiveness, `Maintainers` (usually requirements engineers themselves) might change the requirements inconsistently due to the ambiguous phrase. `Reviewers` of the requirements could overview an inconsistency or another serious issue because of the defect. `Architects, Developers` and `Test Engineers` could require additional effort, because they build the wrong system or test the system against the wrong requirements. This can also lead to and undiscovered invalid tests or an invalid product. `Legal` representatives could discover unintended legal bindings (e.g. no legal binding of requirements) and derive invalid obligations from the requirements. Lastly, if `Estimators` (such as the aforementioned roles or project management) misunderstand the requirement, the estimations resulting can be flawed (too high or too low).

Second-Order Impacts: The stakeholders of the resulting product are the `User` and the `Product owner`. If tests and requirements mismatch in their misunderstanding (and thus discover their mismatch in interpretation), this might lead to product delays. On the other hand, if tests and requirements match in their misunderstanding, the shipped product is invalid, i.e. does not reflect the stakeholders need. Both options lead to unsatisfied users and an unsatisfied product owner. If the incorrect product matches the legal obligations, the product owner still has to accept the product and create (and pay for) change requests for the mismatches. If the incorrect product mismatches the legal obligations, the development side of the project has to fix the system (and pay for fixes and consequences). In both of these cases, one side of the partnership will be unsatisfied with the result. These consequences might then lead to bad reputation of either side and mismatch between planned and actual delivery date and costs might lead to financial risks. All in all, these effects can lead to a lost customer.

Third-Order Impacts: Lastly, there can be systemic effects from these first and second-order effects. These third-order effects describe impacts onto the company or the society, e.g. through the bad reputation or the lost trust. This includes also impacts on persons that do not directly interact with the system. However, these effects are very speculative, not only in RE, but in systems theory in general. Therefore, we will not go into details here.

This discussion of the causes and effects of defects in RE artifact quality aims at illustrating the following points. First, as we discuss in Section 2.3.1, we can differentiate between defects originated before and within the documentation. Second, as

we illustrated in this section, small issues in RE artifact quality can have tremendous impacts onto projects. Third, these impacts come from the complex interplay of artifacts and stakeholders in the process. Lastly, the further away the defect source is from its impact, the more vague and imprecise is the cause-and-effect relationship between source and impact. Therefore, in this thesis, we focus on impacts where source and consequence are close in order to avoid confounding factors blurring the analysis.

2.4. Related Work on Quality Models in Requirements Engineering

Various approaches towards RE quality models exist. In the following, we first show, from an academic viewpoint, how various authors build models for defining RE artifact quality. Here, we describe generic quality models and briefly explain more specific adaptions. We then explain, from a rather industry-focused viewpoint, how various standards summarize best practices.

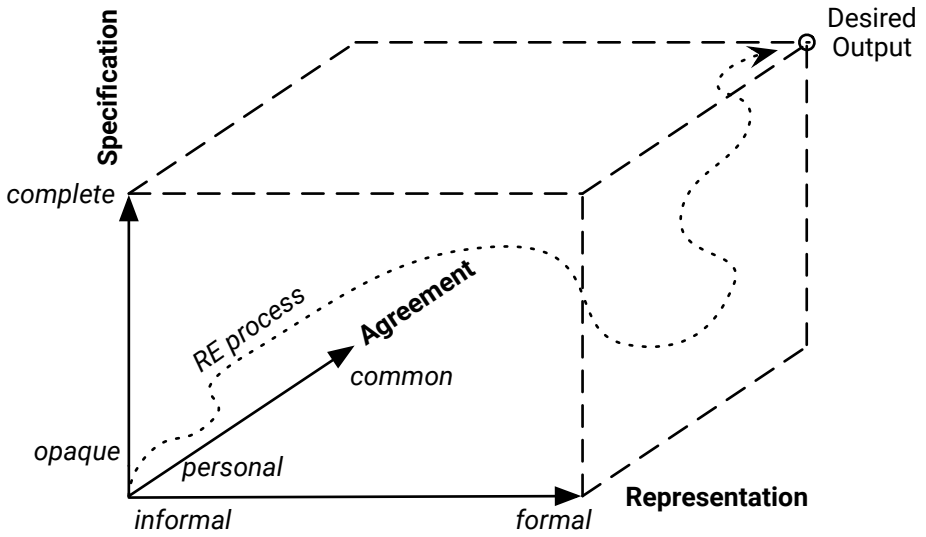


Figure 2.9.: This figure shows the three dimensional quality model as defined by Pohl [Poh93]. Figure taken from [KLS95].

2.4.1. Generic RE Quality Models

Over the years, various researchers have approached the concept of RE quality. Along the most prominent line of research, Pohl [Poh93] models RE quality along three fundamental dimensions (see Fig. 2.9). These dimensions are *specification* (degree of completeness), *representation* (degree of formalization), and *agreement* (degree to which a common view was obtained). In the understanding of Pohl, the RE process is then a path within this model, towards a complete, formal and commonly agreed specification.

Around the same time, Lindland, Sindre and Sølvberg [LSS94] were also discussing the shortcomings of existing quality definition[3]. In particular, in RE at the time, more and more researchers tried to apply various forms of modelling. To deal with this understanding of RE as conceptual models, and for a more systematic approach towards quality than a list of characteristics, Lindland et al. developed a novel framework for the quality of conceptual models. The framework is based on the following concepts:

Model are all statements made by the conceptual model.

Language are all statements possible with the vocabulary and grammar provided.

Domain are all statements that are "correct and relevant about the problem at hand." ([KLS95])

Audience interpretation are all statements which the audience perceives to be contained in the model.

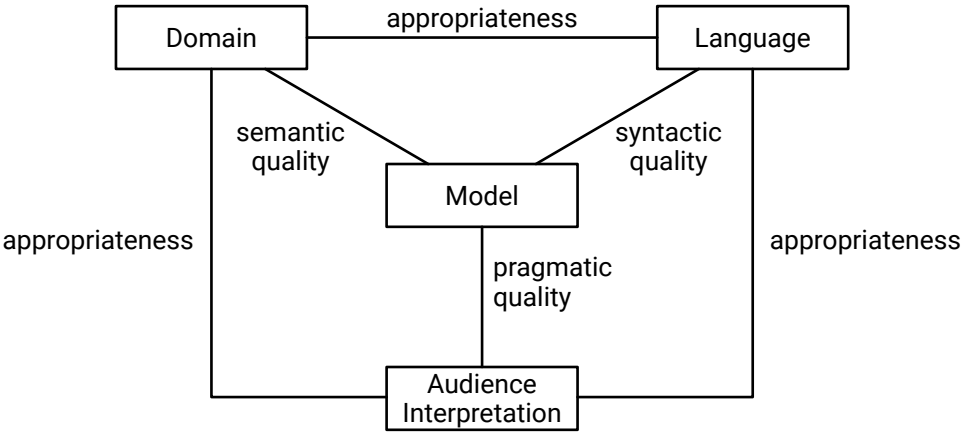


Figure 2.10.: This figure shows the terminology of the semiotic quality model by Lindland et al. as defined in [LSS94]. Figure taken from [KLS95].

Based on these basic concepts, the framework defines appropriateness, as well as three types of quality (see Fig. 2.10):

Appropriateness is degree of fitness between domain, language and audience.

Syntactic quality is the extent to which the model is correct with respect to the applied language.

Semantic quality is the extent to which the model is correct with respect to the domain.

Pragmatic quality is the extent to which the model is understood by the audience.

Afterwards, the framework classifies existing quality characteristics into these categories.

In comparison, despite different terminology, both models address similar issues. As Krogstie, Lindland, and Sindre [KLS95] analyze, Pohl's model aims for complete formalization, which might not necessarily be "always desirable" ([KLS95]). However, Lindland's model misses the characteristic *agreement*. Therefore, Krogstie et al. [KLS95] include the missing concepts proposed by Pohl into the Lindland model through the following concepts:

3 The existing quality definitions at that time were lists of characteristics, very similar to what is still considered best practice in standards.

Participant Knowledge is understood as the union of the knowledge of the problem of all actors in the audience (i.e. "all possible statements that would be correct and relevant for addressing the problem at hand according to the knowledge of the actor" ([KLS95]).

Perceived semantic quality is analogue to semantic quality: Whereas semantic quality is the correctness of the model with respect to the domain, the perceived semantic quality, is the correctness of the interpretation of the model with respect to the understanding of the domain.

Social quality is defined as the agreement between social actors.

Finally, Krogstie [Kro98] furthermore extends the framework by categorizing the proposed quality characteristics of Davis et al. [DOJ+93] into the framework and adding a last quality facet:

Physical quality consists of externalization and internalization. Externalization means that the knowledge of all actors can be contained in the model. Internalization means that the model is persisted and available to the actors.

Besides this, Krogstie added various characteristics of appropriateness, which we leave out for the sake of clarity.

2.4.2. Specific RE Quality Models

In addition to these generic models, various authors have introduced specific quality models, depending on the representation used. For example, Berry et al. [BBG+06] define a quality model for natural-language specifications combining their previous works [FFGL00, BKK03, BK04] in this area. For this, they define a set of quality characteristics, similar to the ones discussed in requirements standards, as well as a set of manifestations of problems, i.e. lexical, syntactic, structural and semantic. Afterwards they classify various defects in natural language according to the model. Similar approaches exist for use case quality [AS02, AS02, FGLM02] as well as user story quality [Coh04, LDBvdW15, LDvdWB16]. All these models have in common to focus on intrinsic properties of artifacts rather than on its usage for the engineering endeavor.

2.4.3. RE Quality Models in Standards

To explain the status quo of RE quality in standards, in the following we refer to two of the current standards. First, according to a recent systematic analysis by Schneider and Berenbach [SB13]), the ISO-29148 [ISO11b] "is actually the standard that every requirements engineer should be familiar with" ([SB13]). Second, the International Requirements Engineering Board (IREB) recently received more and more attention through their certification. Therefore, in the following, we discuss the perspectives on quality of these two institutions.

2.4.3.1. ISO/IEEE/IEC-29148

The ISO-29148 [ISO11b] is the most current standard applicable to RE. As such, it replaced the now superseded IEEE Recommended Practice for Software Requirements Specifications (IEEE-830) [IEE98]. According to Schneider and Berenbach, it can be considered "the mother of all requirements standards as it gives a rather extensive description of the domain of requirements engineering" ([SB13]). It is tailored particularly to systems engineering, but can be applied to various types of software-intensive systems in various domains.

The standard specifies:

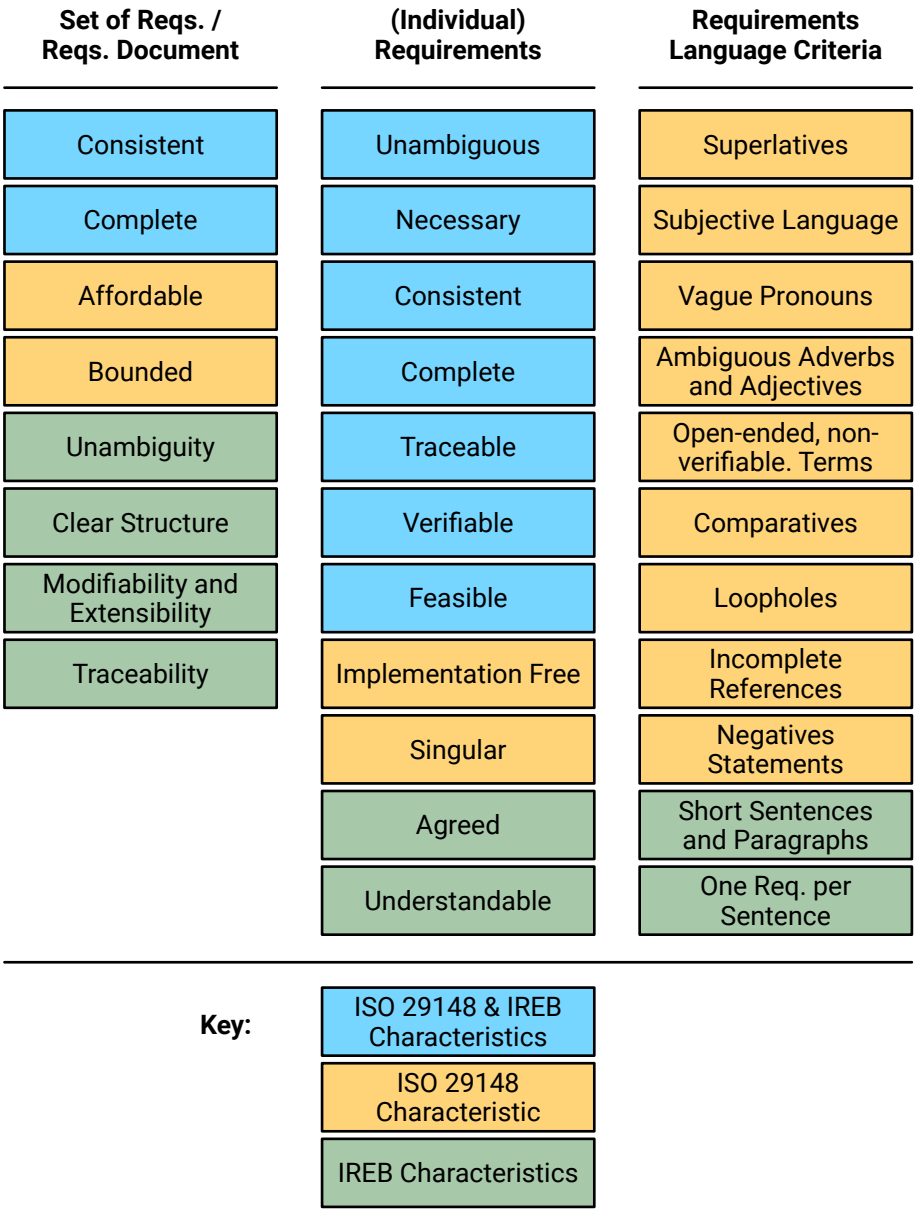


Figure 2.11.: This figure depicts the quality characteristics of ISO 29148 and the IREB syllabus in comparison. Blue characteristics are shared characteristics, orange and green characteristics appear only in one of the standards. Please note that, as we discuss in the text, some characteristics are shared between the standards by their name, but vary in the precise meaning of the characteristics.

- Definitions for the most important concepts in RE
- Processes for software and systems requirements
- Required information items to be produced during these processes
- Content of these information items
- Guidelines for information items through templates and rules for natural language
- Relations to other software lifecycle standards

For this thesis, we are especially interested in the understanding of quality, according to the standard.

RE artifact quality according to ISO-29148. According to the ISO-29148 standard, a good RE artifact is one that follows its templates and formats. Regarding the content, the standard defines a set of quality characteristics, which we depicted in blue and orange in Fig. 2.11. The ISO 29148 defines the characteristics on three levels: In the level of a set of requirements, individual requirements, and requirements language.

At the highest level, according to ISO 29148, a set of requirements should be [ISO11b, pp.11-12]:

- **Consistent:** The ISO 29148 subsumes three aspects under this category. No individual requirements is contradictory, no individual requirements is duplicated, and the same term is used for the same concept throughout the whole set of requirements.
- **Complete:** The ISO 29148 understand as complete that "the set of requirements needs no further amplification because it contains everything pertinent to the definition of the system or system element being specified." ([ISO11b, p.11]) In addition, the set does not contain any TBx's, such as *to be defined (TBD).*
- **Affordable:** The ISO 29148 requires that the "set of requirements can be satisfied by a solution that is obtainable/feasible within life cycle constraints" ([ISO11b, p.12]).
- **Bounded:** Lastly, ISO 29148 requires that the specified requirements stay within the identified scope.

At the second level, ISO 29148 defines nine characteristics for each individual requirement [ISO11b, p.11]:

- **Unambiguous:** This combines two aspects: First, requirements can be only interpreted in one way, and, second, requirements are easy to understand.
- **Necessary:** This combines the following aspects: First, a removal of the requirement leads to a deficiency. Second, the requirement is not obsolete and, last, planned expiration is clearly identified.
- **Consistent:** An individual requirement is consistent if it is free of conflicts.
- **Complete:** An individual requirement is complete if it needs no further amplification. In particular, "it is measurable, and sufficiently describes the capability and characteristics to meet the stakeholder's need." ([ISO11b, p.11])
- **Traceable:** An individual requirement is traceable if it is both upward-traceable to its source, and downward-traceable to derived requirements or implementation artifacts.
- **Verifiable:**An individual requirements is verifiable if, based on the requirement, it is possible to prove that the system satisfies the requirement. The standard furthermore states that "Evidence may be collected that proves that the system can satisfy the specified requirement." ([ISO11b, p.11])

- **Feasible:** An individual requirement is feasible if it is technically achievable without major technology advances and within system constraints and acceptable risks.
- **Implementation Free:** An individual requirement is implementation free if it does not add unnecessary constraints to the design.
- **Singular:** An requirements statement[4] is singular if it only contains one requirement. In particular, it is not singular if it contains conjunctions.

Lastly, at the lowest level, the standard defines a set of *requirements language criteria*[5]. This set defines various grammatical and lexical critera which should be avoided. In particular, the list contains *comparatives, superlatives* and *vague pronouns* as grammatical features to avoid, and *subjective language, ambiguous adverbs and adjectives, loopholes*, and *negative statements* as lexical phrases to avoid. Lastly, it mentions incomplete references.

Issues in ISO-29148. In addition to the general critique motivated in Chapter 1.1, the details of the standard reveal a broader list of issues in the details of the ISO 29148's quality definition. In the following, we briefly list a subset of the problems:

- The characteristic *traceable* is defined in a recursive manner.
- The characteristic *verifiability* requires that a requirement is provable. Assuming a proof to be a formal verification, we argue that for most requirements this definition is hardly achievable in practice. Hence, the definition adds that "evidence may be collected that proves that the system can satisfy the (...) requirement." ([ISO11b, p.11]) However, this latter definition is a significant weakening of the former. The ISO 29148 leaves it to users of the standard to interpret which definition to use and how to interpret it.
- However, the list requirements language criteria is only part of the recommendations for natural language. Throughout the document, the standard provides further implicit or explicit suggestions for higher quality, such as the usage of active voice [ISO11b, p.10].
- The characteristic *consistency* is not defined consistently. While for an individual requirement, it just refers to freedom of conflicts, for a set of requirements, it also refers to duplication and consistent usage of terminology.
- One could discuss to which extent *necessary* and *implementation free* refer to the same criterion. Every violation of implementation free, is also a violation of necessity. The same holds for unambiguity and verifiability. Every ambiguous requirement is by definition not verifiable.

2.4.3.2. IREB

The IREB[6] is a nonprofit organization, which was founded in 2006, and which is the provider of the CPRE (Certified Professional for Requirements Engineering) certification scheme. As such, IREB is not a standardization authority such as ISO, IEEE or others. Instead, it provides certifications for requirements engineers in practice. All in all, more and more companies make use of this certification authority. IREB announced that, over the last years, over 27,000 practitioners in 66 countries have passed their examination[7].

4 Please note that in contrast to the remaining characteristics, ISO 29148 defines singularity based on a requirement statement, instead of a requirement.

5 Please note that, in the following, we will use the term *characteristics* on the abstract quality definition level, and *criteria* for concrete (e.g. language) guidelines.

6 `http://www.ireb.org`

7 `https://www.ireb.org/service/statistics/`

Therefore, IREB does not standardize RE or RE artifacts. However, due to their widespread acceptance, the course material of IREB serves as a standard commonly accepted in industry. The following discussion of IREB's understanding of quality builds upon two sources: The definitions, provided by Glinz [Gli14], and the current syllabus for the certification [Int15]. In the following, we assume that the official IREB syllabus is based on the official IREB glossary provided by Glinz[8].

RE artifact quality according to IREB. In the IREB Glossary, quality is described in an abstract manner, instead of a concrete quality model: "Quality: The degree to which a set of inherent characteristics of an entity fulfills requirements." ([Gli14, p.16]) Therefore, in order to understand RE artifact quality, IREB asks us to define the requirements to an RE artifact.

In the syllabus [Int15], this is broken down into concrete characteristics[9]. Again the characteristics are differentiated through three levels: the requirements document level, the requirements level, and some rules for natural language [Int15, p.16]. In the following, we describe these characteristics, in relation to the ISO 29148 (see Fig. 2.11).

For the requirements document level, IREB describes 6 characteristics, 2 of which are shared with the ISO 29148 (i.e. *consistency and completeness*). IREB adds *unambiguity, clear structure, modifiability and extensibility*, and *traceability*.

- **Consistency:** Just as the 29148, IREB defines consistency as being free of contradicting statements [Gli14, p.11]. However, this does not include redundancy or specific references to terminology.
- **Completeness:** Completeness is defined as the degree to which all information is present that is required for developing the correct system [Gli14, p.10]. As such, the definition of completeness in IREB is very similar to the definition in 29148.
- **Unambiguity:** Glinz defines unambiguity as the degree to which a requirement cannot be understood in multiple ways. It is not explicated in either document, but we assume this is the same definition applied also for a requirements document.
- **Clear Structure:** This term remains undefined in [Gli14] and [Int15].
- **Modifiability and Extensibility:** These terms remain undefined in [Gli14] and [Int15]. However, *changeability* is defined as the degree to which an artifact can be modified [Gli14, p.10].
- **Traceability:** Glinz only defines traceability at the level of individual requirements. It remains undefined for a requirements document.

For each requirement, IREB defines the following 9 characteristics. It contains 7 characteristics shared with 29148 (namely *unambiguous, necessary, consistent, complete, traceable, verifiable, feasible*, and adds two (*agreed* and *understandable*; see Fig. 2.11).

- **Unambiguous:** (see above).
- **Necessary:** Remains undefined.

8 Please note that as required by ISO/IEC 17024:2012 [ISO12], IREB is not involved in the certification process. Hence, IREB provides neither the training, nor the certification themselves. As such, in contrast to the syllabus, the training material can vary between training facilities. Therefore, we refrained from using additional resources such as mandatory training material, since these are not obligatory for practitioners taking the training.

9 IREB refers to these characteristics as *quality criteria*. To maintain consistency, we will use the term *characteristics*.

- **Consistent:** Consistency is only defined at the level of a set of requirements, see above.
- **Complete:** Glinz defines completeness for a single requirement as the degree to which the requirement contains all necessary information [Gli14, p.10].
- **Traceable:** According to Glinz, traceability relates requirements with its origins, with its implementation, and with requirements it depends on [Gli14, p.22].
- **Verifiable:** Glinz defines verifiability as "the degree to which the fullfillment of a requirement (...) can be checked" ([Gli14, p.23]).
- **Feasible:** Remains undefined.
- **Agreed:** Remains undefined.
- **Understandable:** Remains undefined.

For natural language, the IREB syllabus provides few guidance. At the respective location, it advices only short sentences and singularity. One could argue that the transformational processes of language effects [Int15, p.18] could further be interpreted as such language guidance. For the rest, the syllabus promotes sentence patterns.

All in all, the IREB's understanding of quality remains unsatisfying. As mentioned above, the syllabus just names the characteristics and does not provide definitions. While the terms that are defined in the glossary provide rather clear and sensible definitions (except maybe for the recursive definition of traceability), many terms are not defined at all. Therefore, we argue that there is a need for more consistency between glossary and syllabus.

2.4.3.3. Discussion: Comparison between the ISO and IREB Standard

The IREB and ISO 29148 views on quality agree on some parts, but differ all in all. We first discuss the agreements, before the differences. Afterwards, we describe our conclusions from the discussion.

The standards both define a quality model through a simple list of characteristics. According to the standards, good requirements documents are those in which the characteristics are fulfilled. The standards share nine of the characteristics (see Fig. 2.11), mostly those characteristics that were defined in earlier literature and standards, such as the IEEE 830 [IEE98].

However, the standards disagree on more characteristics than they agree on. In particular, the standards completely disagree when it comes to concrete language criteria. And even when the standards agree on the characteristics, as soon as they define the characteristics, their interpretations differs significantly. Take, for example, consistency. While the IREB takes only disagreeing requirements into account, the 29148 also understands issues with duplication and terminology under this definition. In addition, the definitions of Glinz take the characteristics as continuous variables, wheres the definitions of the standard are binary.

In summary, while the two standards take the same approach towards quality, as soon as they get more concrete, they differ tremendously. This is especially true for the concrete, assessable language criteria. We argue that these differences indicate two problems. First, the missing agreement at the level of concrete language criteria indicates that we do not yet know what is good or bad quality, and that we have little to no established understanding of the impacts of concrete language criteria. Second and even more problematic, the missing agreement at the level of abstract quality characteristics indicates there is no established understanding and approach towards quality for RE artifacts as a whole. Although Glinz' [Gli14] proposed

definition of quality could be such an approach, the concrete definitions in the syllabus unfortunately does not follow this definition.

2.5. Research Gap

In the following, we summarize the research gaps. We look at quality definitions first, before discussing QA afterwards.

2.5.1. Research Gap: Quality Definitions for RE artifacts

To summarize this section, there are various viewpoints onto quality, namely:

Generic theories: Various authors have worked on approaches to define RE quality in a systematic way. While these approaches are helpful to define quality factors, their generic approach does not take into account that requirements engineering is just a means and not an end. For one example, Pohl [Poh93] assumes that formalization is a goal of the RE process. We disagree with this view, since a formal model that is never used nor understood has no purpose in a software engineering project.

Standards and generic quality characteristics: Unfortunately, the current standards for RE quality are rather a list of characteristics than a structured approach towards quality. Although IREB contains some hints towards thinking about RE artifact usage, this is not yet reflected in the quality model. In particular, the strong differences between the currently most relevant standards show how arbitrary the existing characteristics are, and that we do not have a systematic approach for deciding which characteristics are relevant, when and why. In addition, the meaning of various characteristics is too abstract and unclear.

Specific quality factors: The same holds for approaches for specific, more concrete quality factors, such as *negative statements*. Existing approaches are defining lists of characteristics without an adequate concept of quality as a foundation. In consequence, the reasoning behind quality factors remains unclear, which makes the question whether or not to make use of the quality factors dubious in practice.

All in all, we have generic theories, sets of characteristics, and concrete quality factors. While some approaches provide a holistic view on quality, none of these takes the usage of the RE artifacts and the context into account. Therefore, we still have a limited understanding of what high quality RE artifacts are in its usage context.

2.5.2. Research Gap: QA for RE Artifacts

In order to assure that RE artifacts are of high quality in practice, various QA techniques can be applied. A detailed analysis of existing methods for QA of RE artifacts can be found in Publication G. As explained, we can differentiate into constructive and analytical, and automatic and manual approaches. Of these, automatic analytical methods have the potential to address the problem described in Section 1.1. In the following, we summarize the state of the art in application of automatic methods for analytical quality assurance and describe open research gaps. For this, we summarize related work from an evaluation, a quality definition, and a technical perspective[10].

10 Please refer to Publication G for the detailed analysis of existing methods for QA of RE artifacts that lead to this summary.

First, one gap in existing automatic QA approaches is the lack of empirical evidence, especially under realistic conditions. Only few of the introduced contributions were evaluated using industrial requirements artifacts. Those who do apply their approach on such artifacts, focus on quantitative summaries explaining which finding was detected and how often it was detected. Some authors also give examples of findings, but only few works analyze the accuracy of their automatic approaches in depth, especially in the vague domain of ambiguity. When looking at the characteristics that are described in ISO 29148, we have not seen a quantitative analysis of precision and recall. Furthermore, reported evidence does not include qualitative feedback from engineers who are supposed to use the approach, which could reveal many insights that cannot be captured by numbers alone. However, we postulate that the accuracy of quality violations very much depends on the respective context. This is especially true for the vague domain of natural language where it is important to understand the (context-specific) impact of a finding to rate its detection for appropriateness and eventually justify resolving the issue.

Second, the existing approaches are based on proprietary definitions of quality, based on experience or, sometimes, simply on what can be directly measured. The ARM tool [WRH97] is loosely based on the IEEE 830 [IEE98] standard. However, as the recent literature survey by Schneider and Berenbach [SB13] states: *"the ISO/IEC/IEEE 29148:2011 is actually the standard that every requirements engineer should be familiar with"*. We are not aware of an approach that evaluates the current ISO 29148 standard [ISO11b] in this respect. As the analysis of existing works in Publication G shows, for most language quality defects of ISO 29148, there has not yet been a tool to detect these quality defects. To all our knowledge, for neither of these factors, there is an differentiated empirical analysis of precision and recall. Yet, many other quality models (most notably from the ambiguity handbook by Berry et al. [BKK03]) and quality violations could lead to Requirements Smells, as far as they comply with the definition given in the next section.

Finally, taking a more technical perspective, our Requirements Smell detection approach does not fundamentally differ from existing approaches. Similar to previous works, we apply existing NLP techniques, such as lemmatization and POS tagging, as well as dictionaries. For the rules of the ISO 29148 standard, no parsing or ontologies (as used in other approaches) were required. However, to detect superlatives and comparatives in German, we added a morphological analysis, which have not yet seen in related work.

In summary, we need more evidence on automatic QA for requirements artifacts via systematic studies in terms of distribution, precision, recall, and relevance, as well as by means of a systematic evaluation with practitioners under realistic conditions.

CHAPTER 3

Research Design

This chapter outlines the design of this thesis. In particular, based on the problems described in Chapters 1 and 2, we derive a thesis statement and research goals. Afterwards, we detail these goals into research questions. For each research question, we design applicable methods and summarize the contributions. Lastly, we explain how these contributions relate to other works, to which the author also co-contributed.

3.1. Problem and Thesis Statement

In the introduction, we described the problems of incompleteness, inadequacy and imprecision of the current state of the practice in RE artifact quality. In the previous chapter, we described that the state of the art in RE artifact quality research is not able to address this problem. We summarized this in the following problem statement:

Problem Statement: We have a limited understanding of what *high quality* RE artifacts are and need *more efficient methods* to control RE artifact quality in practice.

Thesis Statement: In our work, we address these problems through an activity-based understanding of quality and automatic detection of quality factors. In summary, we argue that an activity-based quality model enables to more precisely and completely model RE artifact quality for a given context. Moreover, to apply these quality models and therefore to improve RE artifact quality in practice, a combination of automatic and manual methods can help to increase efficiency, speed and consistency of RE artifact quality control.

3.2. Research Challenges and Research Questions

The problem and thesis statements raise two challenges, which we refine into two research questions each (see also Fig. 3.1). The first part sets the theoretical basis, whereas the second part targets more efficient solutions for quality control.

Challenge 1: We need a precise and valid understanding of what high quality RE artifacts are in a specific context.

RQ 1: How can we precisely define quality for RE artifacts? To precisely discuss what high quality requirements are, and to have a basis for quality assurance, we need a model to systematically reason about RE artifact quality. In other words, we need a model that explains which factors of an RE artifact define it as a high-quality artifact. This model must be adaptable to various contexts, and must allow to accept or refute a factor, based on an systematic argumentation.

RQ 2: How can we create valid quality models? The language to define such a quality model is not sufficient. We furthermore need applicable methods to verify that the models are valid. We need approaches that enable to either build valid models from scratch or differentiate the correct from the wrong factors.

However, as explained in Chapter 1.1, to just understand and define quality in a precise manner is not sufficient since manual QA lacks efficiency in practice.

Challenge 2: We need more efficient methods to control RE artifact quality in practice.

RQ 3: How can we efficiently ensure quality factors? In practice, even if there is an established quality model, projects struggle to ensure that their RE artifacts adhere to this quality model. Therefore, we furthermore need an efficient method to support requirements engineers keeping the desired goal of artifact quality. To answer this, we propose an approach called automatic requirements smell detection.

RQ 4: What are the benefits and limitations of requirements smell detection? To validate that the requirements smells approach in fact achieves the stated goals, we validate the advantages and limitations of such an approach. In particular, we are interested to understand in which cases automatic smell detection cannot support requirements engineers.

3.3. Methods and Contributions

In Fig. 3.1 we provide an overview of the approaches and contributions of this thesis. The contributions are structured along the two aforementioned problems. Each problem is addressed through first, an analysis and design phase leading to a constructive approach (RQ 1 and 3), and, subsequently, the approach's evaluation phase (RQ 2 and 4). Each research question is answered by one or multiple contributions, which we explain in depth in the following. Furthermore, Fig. 3.1 shows how the contributions relate to the Publications.

RQ 1: How can we precisely define quality for RE artifacts?

Method: We suggest *activity-based RE artifact quality models (ABRE-QMs)* to precisely define quality in a given context. Based on a quality-in-use viewpoint, an ABRE-QM defines quality as a set of quality factors of entities that have an impact on activities in the software development process. We furthermore provide an approach to define ABRE-QMs and discuss a research roadmap.

Contribution 1. The notion of Activity-based RE artifact Quality: RE artifact QA requires a precise understanding of quality. As explained in Chapter 1.1, to this end, quality models are often incomplete, inadequate and imprecise in their

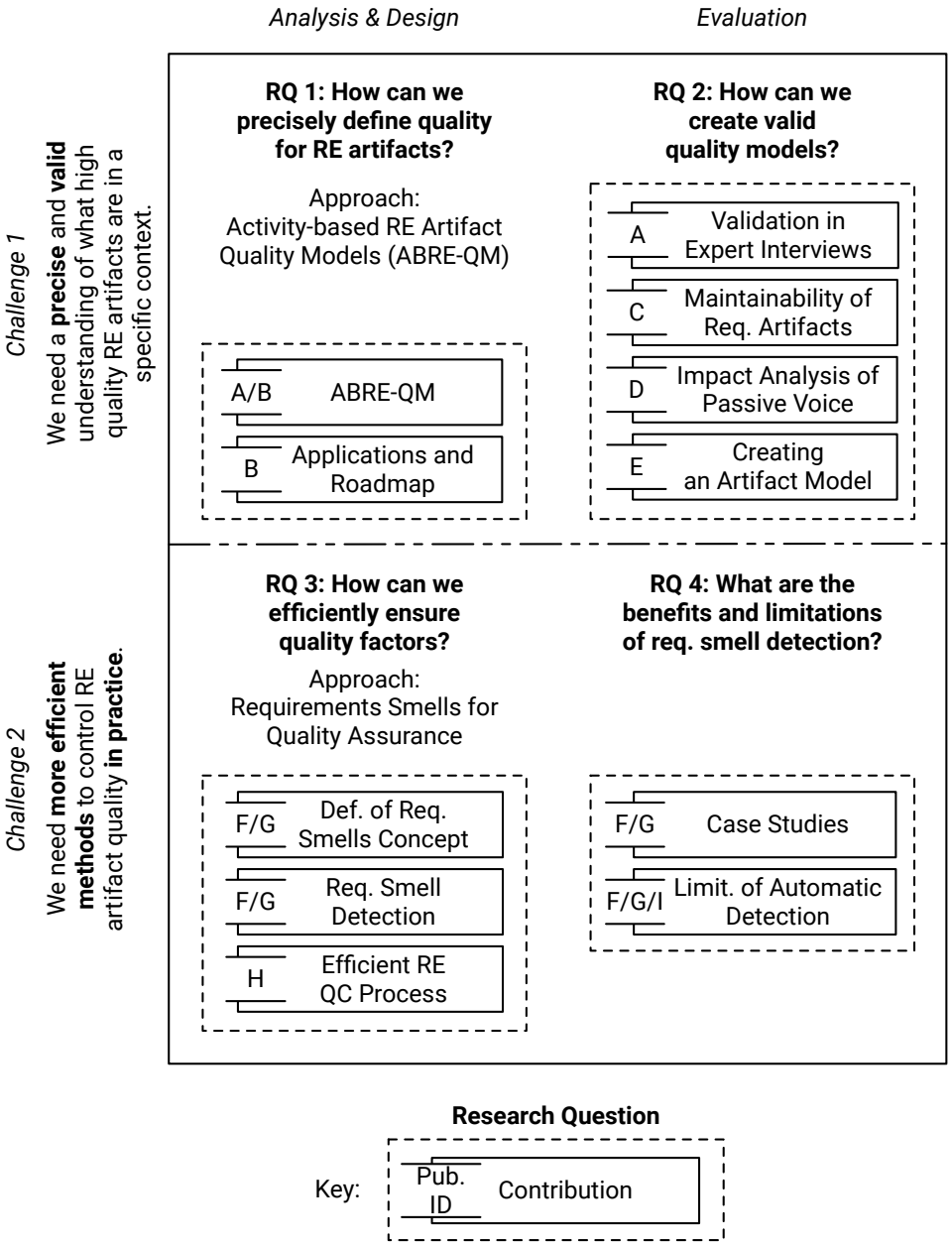


Figure 3.1.: This figure provides an overview of the contributions of this thesis. The figure relates the research challenges (on the left), research questions (RQ 1-4) and contributions (in boxes), together with the related publications (A-I)

reasoning. We present an approach that enables to define quality of RE artifacts in a specific context by applying activity-based quality models. Activity-based RE artifact quality models (ABRE-QMs) define quality through properties of artifacts (i.e. quality factors) and the impact of these properties onto the activities that are conducted in a specific software development context. Thereby, ABRE-QMs can define requirements quality precisely and thus enable to adjust the quality definition specifically to the corresponding context. This contribution furthermore enables to discuss and define quality in a common language.
Results: We define a meta-model for ABRE-QMs, and show that ABRE-QMs enable to precisely model quality characteristics in practice. Practitioners indicate the potential of an ABRE-QM to check the definition of quality (e.g. in the form of guidelines) in terms of completeness and correctness.

Contribution 2. A research roadmap for RE artifact quality: The aforementioned definition of RE artifact quality has various implications onto RE artifact quality, both in terms of applications and research to be conducted.
Results: We contribute an analysis of existing applications of RE artifact quality models. Furthermore, we outline a research roadmap that defines research objectives along the content of this model.

RQ 2: How can we create valid quality models?

Method: To create valid quality models, we need to validate the defined quality factors. This thesis discusses interviews, case studies, and experiments as three different approaches for empirical validation of the impact of quality factors. Based on the design and execution of two industrial studies and one experiment, the approaches are compared and evaluated, leading to three different applications. Regarding requirements quality, the studies revealed various quality factors for requirements maintenance and the risks of passive voice for understanding of requirements. In addition, we contribute an approach to refine existing artifact models, which is a basic necessity for ABRE-QMs.

Contribution 3. Expert interviews for model validation: In an interview study, we validated an ABRE-QM as a whole by translating a company's guidelines into a quality model and discussing the resulting model as well as the benefits of such a model in contrast to the existing guidelines.
Results: The experts reported that ABRE-QMs can increase validity and completeness of the company guidelines. Regarding the validation method, expert interviews show advantages in terms of validation speed, since it is possible to quickly discuss even a large number of impacts. However, this method comes with the risks of various types of personal bias of the interviewees. We therefore suggest to choose interviews to validate a quality model as a whole and, whenever impacts result in discussion, formulate a set of hypotheses that can afterwards be analyzed in depth with case study research or experiments.

Contribution 4. Case study research on maintaining RE artifacts In a case study, we analyzed quality factors for maintenance activities. By classifying 14 months of changes in a project's RE artifacts, we aimed at understanding which parts of RE artifacts are most affected by maintenance activities and how.
Results: The study revealed that use cases evolved mostly in alternative flows. The study also indicated that changes that must be conducted in multiple locations are among the most difficult and error prone changes and that terminology and descriptions of user interfaces are among the most often changed content. Regarding the validation method, case studies are less prone to personal bias, however, the selection of cases can heavily influence the

results. We therefore suggest to choose case studies for activities and quality factors that are still unknown ground.

Contribution 5. An experiment on the impact of passive voice: In an experiment, we inspected one quality factor in depth, namely passive voice, and its impact on understanding requirements. For this, we provided subjects with real-world passive voice requirements sentences and measured the number of errors they produced when modelling their understanding. We compared the results to a control group, which we provided with the same requirements written in active voice.

Results: The subjects that we provided with passive voice requirements showed significant problems in understanding relations between the described domain objects. Regarding the validation method, experiments seems to reveal the most substantiated impacts. However, we must carefully evaluate external validity, since experiments usually inspect variables in a very isolated setting. In addition, experiments require high effort in both experiment setup and experiment execution. We therefore suggest to choose experiments for very specific, but unclear quality factors.

Contribution 6. Creating an agile artifact model: In order to create activity-based quality models, we need artifact models to define quality factors on. Not always is such a model present in practice. In this contribution, we needed an artifact model for distributed agile project management, largely focussing on requirements. To derive such an artifact model, we refined an existing artifact model in a cooperation with plixos GmbH and analyzed the results.

Results: We contribute an artifact model to support the construction of tools for managing distributed projects. For this, we use a previously defined reference artifact model for agile methods and enhance it for the use as a real-world data exchange model. The study indicates customization that is required in specific cases and shows an approach for defining customized artifact models.

RQ 3: How can we efficiently ensure quality factors?

Method: To bring such ABRE-QMs into practice, we need a more efficient method for RE artifact QA. Therefore, we propose to detect violations of specific quality factors of such an ABRE-QM automatically. For this, we transfer the concept of code smells to RE as *requirements smells*. Based on ABRE-QM, we define requirements smells as well as requirements smell detection approaches. We validate the approach technically in a prototype and discuss various usage scenarios. In addition, we conduct interviews to understand the shortcomings of existing manual QA and create a QA process that combines manual and automatic QA for a more efficient approach towards RE artifact QA.

Contribution 7. Requirements Smells: We define quality assurance with requirements smells, a method for automatically checking a natural language RE artifact against certain quality factors. A Requirements Smell is an indicator of a quality violation (based on an ABRE-QM), which may lead to a defect, with a concrete indication and a concrete detection mechanism. We furthermore provide a taxonomy for requirements smells.

Results: We define requirements smells for automatically checking an RE artifact against certain quality factors.

Contribution 6. Requirements Smell Detection and Tool Support: We provide a technical validation for requirements smell detection through a prototype that detects requirements smells in various types of RE artifacts.

Results: The approach shows that various requirements smells can be automatically detected through approaches in natural language processing.

Contribution 8. A Process for Efficient QA: We analyze the problems of QA in a case study at a company in industry. In this case study, we conduct interviews with practitioners at Munich Re on challenges of existing QA processes. Based on these interviews, we propose a combined approach of manual and automatic QA.
Results: The data analysis of the interviews resulted in 8 main problems. We define a set of principles and proposals to address these problems, which result in a flexible review process that includes both manual and automatic QA. The results extend the existing framework of Katasonov and Sakkinen [KS05], among others, by automatic requirements smell detection and the concept of activity-based requirements engineering quality.

RQ 4: What are the benefits and limitations of requirements smell detection?

Method: We apply requirements smell detection in a series of cases provided by three industrial and one university context. Based on qualitative and quantitative data, we show the potential of requirements smells to detect quality defects, but also the varying precision of such an approach. In addition, we provide an analysis that explains which types of quality defects can be assured with requirements smells and which cannot.

Contribution 9. Practical Evaluation of Requirements Smells: We evaluate the requirements smells in terms of

1. precision and recall of requirements smell detection,
2. practical relevance of requirements smell findings,
3. awareness of found defects to practitioners, and
4. applicability of the process for practitioners.

The evaluation is performed in multiple case studies in three industrial contexts with Daimler AG, Wacker Chemie AG, and TechDivision GmbH and an academic context at the University of Stuttgart. The cases are distributed across various software development processes and various forms of requirements representation.
Results: Requirements smells are present over all methods and domains. We can detect requirements smells with a reasonable precision and produce findings that are relevant to practitioners. Practitioners from different domains and contexts state that the approach can be successfully integrated into the QA process.

Contribution 10. Limitations of Automatic Detection: In the previous contributions, we focused on automatic requirements smell detection. In this contribution, we analyze to what extent RE quality defects can be automated. Based on results of RE artifact reviews, we analyze the scope of what is possible in automatic smell detection.
Results: The studies show five main reasons for undetectable quality criteria: Stakeholder or domain knowledge, requirement of deep natural language understanding, knowledge of system scope or goal, knowledge of process information, and vaguely or subjectively defined criteria. In an analysis of a large, industrial RE artifact guideline, we estimate that 52% of the criteria can be checked either perfectly or with a good heuristic. For detection of violations, most criteria require just simple heuristics. The main reason why criteria cannot be automatically detected are imprecise or unclear definitions.

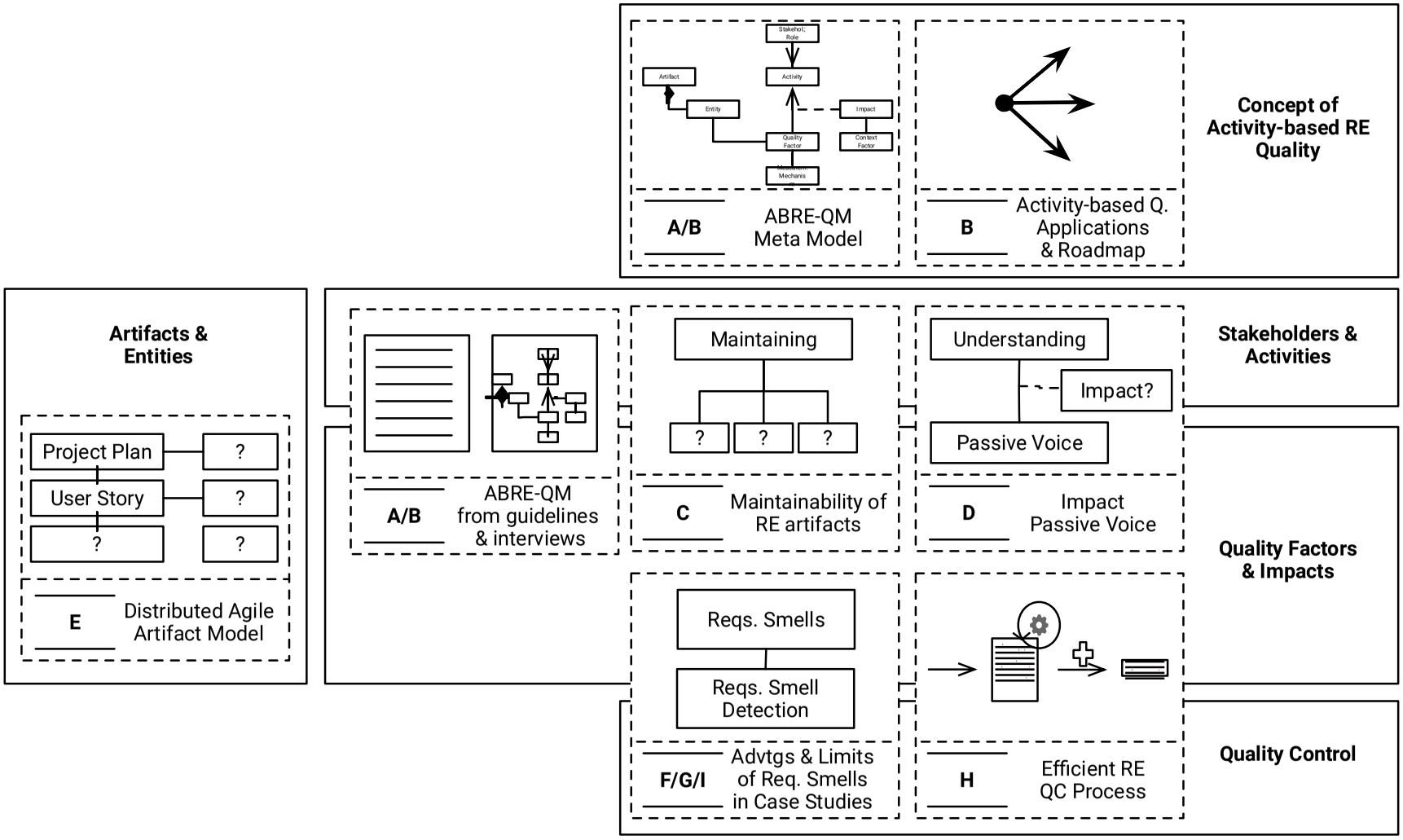


Figure 3.2.: This figure provides an overview of the results of this thesis. The figure relates the results with the related publications (A - I) into the concepts of activity-based RE quality.

3.4. Results Overview

Fig. 3.2 shows the results of the aforementioned contributions within the concepts of an ABRE-QM. Publications A and B answer RQ 1 by defining the basic concepts of this thesis, namely ABRE-QM. Different instances of ABRE-QMs are analyzed throughout answering RQ 2: Publication E explains how to define an artifact and entity model. Publications A, C, and D show different impacts and their validation: In Publication A, we create a complete ABRE-QM from guidelines and interviews, in Publication C we execute case study research to detect quality factors for maintainability, and in Publication D we study the impact of passive voice onto understanding activities in experiments. Lastly, in RQ 3 and RQ 4, we focus on more efficient methods for quality assurance, through the development and evaluation of requirements smells in Publications F, G and I, and the development of a more efficient RE quality control process in Publication H.

3.5. Further Related Works Co-Contributed by the Author

In addition to the included publications, we contributed to various further publications that are related to these aforementioned contents, but not included in this thesis (see Preface). In the following, we briefly go over the content of each publication and explain how it is related to this work.

In [MFME15], we first analyze in a given context whether or not RE artifacts are created and used. The study answers this in the affirmative. Second, we design an experiment, based on an ABRE-QM, and similar to our Publication D, yet for other quality factors and for the activity *creating test cases*. The results showed that incorrect information in RE artifacts leads to worse test cases. It could not show a significant effect of negative statements (a quality factor from the ISO-29148 standard [ISO11b], see also Chapter 2.4.3.1) on the created test cases. This provides evidence for a similar hypothesis stated by domain experts in our study in Publications F and G.
Relation to this work: These studies have two implications for this thesis. First, the results serve as motivation and indicate that RE artifacts are created and used in practice. Second, the experiment provides further evidence that we can use ABRE-QMs to increase our understanding of RE artifact quality through experiments.

In [VFW16], we analyze reports of manual reviews in the automotive domain. Inspecting a sample of these reports indicates, among other aspects, that completeness and ambiguity are the main defects in this context, as reported by practitioners.
Relation to this work: The study provides further motivation for improving quality control for RE artifacts. In particular, an activity-based quality viewpoint enables a more precise definition of defects, and our smells particularly address ambiguity and completeness issues.

In [MMFV14], based on survey research, we define variables for RE artifacts and model the relations between these variables both inside and outside RE. The study indicates the complexity of this endeavour and shows that these variables can only partly be observed and measured.
Relation to this work: The approach gives a first glance on how to understand second and third-order impacts in RE (see also the discussion and future work section of this thesis).

In [EVF16], and [EVFM16], we make use of the activity-based paradigm to define completeness for RE artifacts. In addition, the works provide constructive support in the form of sentence patterns to improve RE artifacts.
Relation to this work: The contributions of these works are twofold: First, they enable to precisely define the quality factor completeness, based on RE artifact use. Second, these works show how ABRE-QMs can also help in constructive QA, in contrast to the rather analytical QA approaches presented in this thesis.

In [ABBF17], we transfer the concept of requirements smell detection to vulnerability descriptions. For this, we created a tool that enables the author to identify missing content in vulnerability descriptions through detection of keywords. The study indicates that the approach can easily be transferred to quality factors of other artifacts.
Relation to this work: The study strengthens our confidence into the broad applicability of requirements smells and our requirements smell detection.

CHAPTER 4

Summary of Results

In this chapter, we summarize the results of the appended publications and describe their interrelation. Following the methodology explained in Chapter 3.3, we go through the results of each research question. We start from the question of how to precisely model RE artifact quality in RQ 1 and analyze the question of how to ensure validity of these models in RQ 2. In RQ 3, we then propose a method to efficiently ensure certain parts of the quality model, before we empirically evaluate strengths and limitations of such a method in RQ 4.

4.1. RQ 1: How Can We Precisely Define Quality for RE Artifacts?

Even though it is widely acknowledged that the quality of RE artifacts is an important factor for project success or failure (see e.g. [Bro06]), there is not yet a common agreement on what the term RE artifact quality really means. Requirements standards, such as the IEEE-830 Recommended Practices [IEE98] or its successor the ISO/IEEE/IEC-29148 [ISO11b], give normative guidelines. However, as we describe in Chapter 1.1, we argue that these standards provide only incomplete, inadequate and imprecise definitions of RE artifact quality.

Therefore, we contribute a novel view on requirements engineering artifact quality, which defines RE artifact quality from a quality-in-use perspective. The proposed model defines quality based on the explicit impact of a quality factor on certain activities and therefore enables to express RE artifact quality in a more precise, adequate and potentially also complete manner. This section summarizes and partly extends the results of Publication A and B

4.1.1. Summary of Approach: ABRE-QMs

To precisely define RE artifact quality, we designed activity-based RE artifact quality models (ABRE-QMs). In the following, we describe the concepts behind ABRE-QMs, before reporting our experiences on validation of ABRE-QMs in the next research question (Section 4.2).

4.1.1.1. ABRE-QM Meta Model

ABRE-QMs are based on a quality-in-use paradigm in RE: We postulate that creating an RE artifact is rarely an end in itself, but just a means to understand and reach the project's goals. Following this line of thought, the purpose of the requirements artifact is to support the stakeholders in the activities they are performing in the project[1].

To describe the structure of ABRE-QMs, we provide an ABRE-QM meta model that introduces the concepts needed to describe an ABRE-QM. The ABRE-QM meta model adapts and extends the QUAMOCO meta model [DJLW09, Dei09, WLH+12]. The QUAMOCO meta model is used to explicitly define quality-in-use characteristics of source code, such as maintainability [DWP+07, Loc13]. We simplify, but also extend the meta model to adapt it to RE artifact quality.

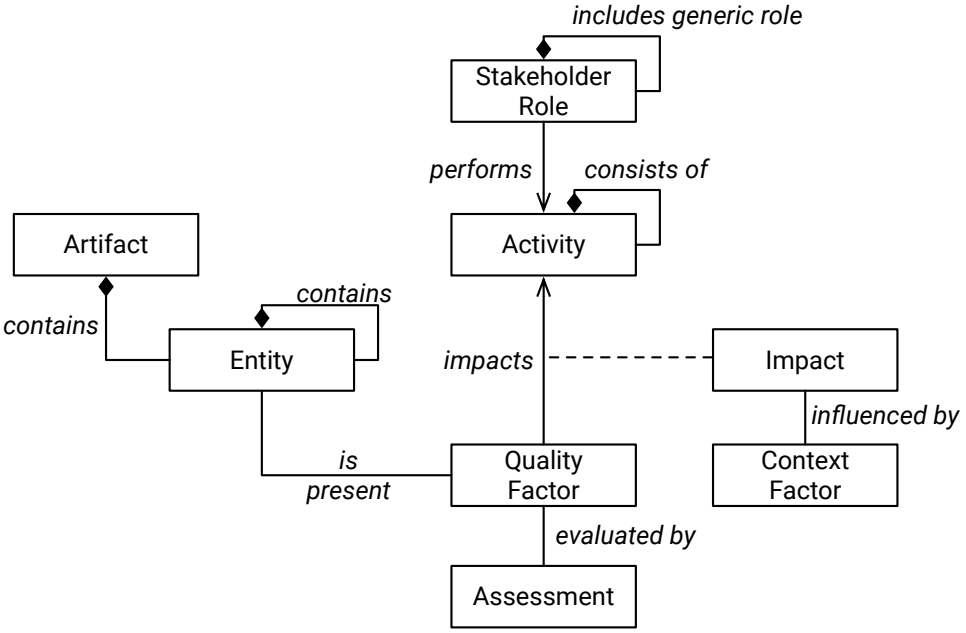


Figure 4.1.: This figure shows the ABRE-QM meta model. The model consists of artifacts and their decomposition into entities, quality factors and their impact onto activities, which are performed by certain stakeholder roles. Impacts are influenced by context factors. Lastly, quality factors are evaluated by assessments.

ABRE-QMs define quality as an instance of the following elements (see Figure 4.1):

An artifact is a documented collection of requirements entities, which is produced during an RE process. An example for an artifact is a `use case document`.

An entity is a coherent documented information. An entity can be a content item [MPKB10], but can also be further decomposed, e.g. into the linguistic components of such a content item (see Section 4.3.2.2 for an instance of such a decomposition). Examples for entities are a `use case`, an `alternative flow` or a *step* within the flow.

A stakeholder role is the role of someone with an interest in the RE artifact [Poh10], such as the tester. Each role can include more generic roles. For example, both

1 Obviously, some projects exist, in which only a certain RE artifact is created and therefore the requirements artifact itself is the project goal. However, even in these cases there is a larger goal for which the RE artifact is just a means. The project is just scoped in a manner that this higher goal is outside of the project scope. In the following, we assume that this is not the case.

`test engineers` and `developers` are also `readers` of the requirements artifact. Therefore, quality factors that affect the activity `read` oder `understand`, affect all `readers` of the artifact, including `test engineers` and `developers` through their included generic role `reader`. This allows combining shared activities that multiple stakeholders must execute.

An activity is an invested effort, which involves one or more of the aforementioned artifacts, such as creating test cases, and one or more of the aforementioned stakeholder roles, such as the `test engineer`. An activity can be broken down into subactivities. For example, the testing activity is decomposed into `creating, running,` and `maintaining` test cases.

A quality factor is a property that is or is not present in an entity. This property must be objectively assessable through a measure to be used for quality control[2]. Depending on the abstraction level of the referred entity, we can differentiate five types of quality factors (extending the definition of Berry et al. [BBG+06], see also Section 4.3.2.2 for more details):

- Domain-semantic are quality factors that define properties of the system with reference to the domain (outside the specification). Examples for this quality factors are in the direction of *validity* or *feasibility*, such as the quality factor whether `use cases` have `business-value` or also whether an `RE artifact` contains *contradictions* between the described system and the domain.
- Language-semantic are quality factors that define properties of the system as it is described. One example is a quality factor whether an `RE artifact` contains contradictions within the text.
- Structural-syntactic are quality factors that describe properties of syntactic relations within the artifact and content model. Examples here are fields to be filled out (under certain conditions) or required and forbidden references between parts of the content model.
- Grammatical-syntactic are quality factors that describe properties of the grammar of a description. A well-known example for this quality factor is `passive voice`.
- Lexical-syntactic are quality factors that describe properties of one or more single elements of the language, such as words or phrases not to be used in the RE artifact.

However, these boundaries are often fuzzy as quality factors can refer to multiple levels. An example for this is a grammatical ambiguity that hints at an semantic incompleteness. As such, a passive voice requirement (as a grammatical-syntactic quality factor) might imply that we do not know yet who is performing a certain action in the system (a language or even domain semantic quality factor).

An impact is an explicit relation between a quality factor and an activity. The impact influences either *effectiveness* or *efficiency* of that activity. This impact is explicitly discussed through: First, a reason, i.e. an argumentation why the presence of a specified characteristic (the quality factor) of an artifact impacts the associated activity; second, consequences on costs, schedule or quality of the developed system; and third, a source from which this impact was derived and which can provide further information, i.e. a requirements quality standard or corporate guidelines.

2 Please note, that for this notion, continuous properties must be transformed into binary values. E.g. in order to use continuous properites such as *business value*, we have to transform the factor into a binary factor, such as *business value is higher than expected effort.*

A context factor influences the impact of a quality factor. For example, the problematic impact of a `passive voice` requirement varies, depending on the background of the reader. If the reader has no or few domain knowledge, the passive voice has a stronger impact. In contrast, in cases where the reader is well aware of the domain and the ideas of the system, the impact can be less problematic. Please note that the usage context is already partly represented through stakeholder roles and activities. These context factors can be understood as an extension of this usage context onto individual quality factors. Context factors are of the following types:

- Human and team context factors refer to the knowledge or situation of an individual stakeholder or a team of stakeholders.
- Process context factors refer to the process within which the activities are conducted.
- Tool context factors refer to the tooling through which's means the stakeholder role uses an RE artifact.

An assessment is a description for evaluating an entity against a quality factor. The application of a assessment against an entity (analogue to Deissenboeck [Dei09]) results in a (potentially empty) set of quality defects. Just as Deissenboeck describes, we see three potential categories of assessments: manual, automatic, and semi-automatic assessments.

4.1.1.2. Exemplary Quality Factor: Explicit Steps in Use Cases

To foster understanding, this section provides an exemplary excerpt of an ABRE-QM (see Fig. 4.2). The example shows the definition of one quality factor, which are the presence of explicit steps in a use case flow.

1. Artifacts and entities: A `use case document` (e.g. [Coc98]) is a common artifact for specifying functional requirements to software systems. A use case document contains one or multiple use cases, which usually contain a `basic flow`, which is a sequence of steps that describes how the user interacts with the system.

2. Stakeholder roles: For the sake of simplicity, in this example, we consider only `test engineers`.

3. Activities: When we analyze how a test engineer processes the use case document in a specific project, we find out that among other activities the test engineers goes through the use case steps and `creates test step(s)` for the use case's basic flow.

4. Quality factors: It is considered good practice in use cases to `explicitly separate` each step instead of describing the whole basic flow in one text block. With the aforementioned context and activity in mind, we understand why a use case with this quality factor is considered higher quality: The test engineer can directly translate the use case steps to test steps. Therefore, the test engineer's task of creating a test sequence can be executed more effectively (and maybe also more efficiently) when the factor is present in the use case. Fig. 4.2 explicates this reasoning through a positive ('+') impact in an ABRE-QM. Please note, that for simplicity, we only discuss one of the impacts of this quality factor here.

5. Context factors: One could consider the applied tool to be a context factors. Depending on the concrete tool in use, the translation is more or less efficient.

6. Assessment: One could discuss various types of assessments, depending on the tool used. A easy-to-apply assessment is a manual review, which can spot this quality defect. In addition, for various requirements management tools,

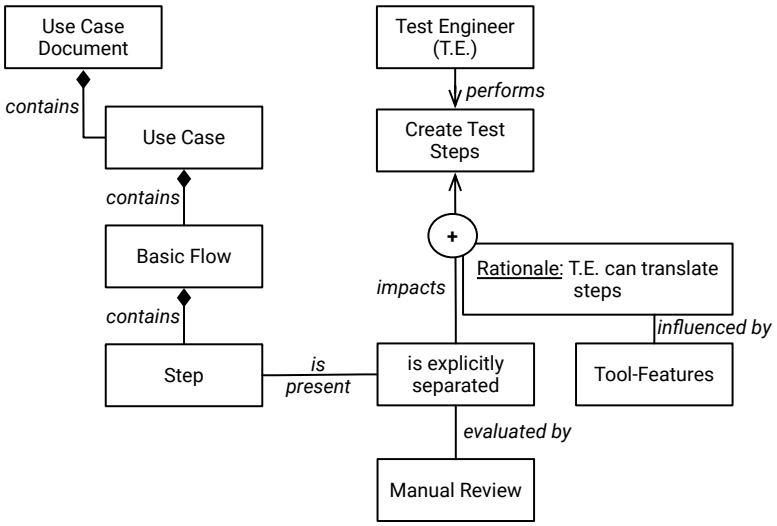


Figure 4.2.: This figure shows a simple example excerpt of a quality factor in an ABRE-QM. The excerpt discusses why explicitly separated steps in basic flows of use cases are considered good quality. In this example, we discuss the impact onto creating test steps, i.e. explicitly separated steps in basic flows allow more efficient and effective creation of test steps through reuse.

one could discuss automatic (or at least semi-automatic) methods through automatic analysis of the use case's structure.

This example shows the definition of one quality factor. An ABRE-QM is a composition of a set of such quality factors with their respective relations. RE artifact quality is thus defined through an ABRE-QM.

4.1.2. Conclusion to RQ 1

So far, we have a limited understanding of what the term high quality RE artifacts means in a specific context. Existing definitions of RE quality define quality in an apodictic manner, but the defined characteristics remain incomplete, inadequate and imprecise. To address this shortcoming, we propose activity-based RE artifact quality models (ABRE-QMs). Through explicit modelling of the impact of quality factors onto activities, the method enables to define RE artifact quality from a quality-in-use viewpoint.

Quality is thus defined as a set of quality factors. Each quality factor explicitly defines its impact onto a stakeholder role's activities in the project. This ensures that each quality factor comes with a precise reasoning of consequences. In addition, since impacts are related to activities, such an understanding of quality takes the context into consideration. Furthermore, we can discuss completeness of such a model by iterating through the stakeholder roles and activities: The quality model is complete if it defines all quality factors required to ensure that all stakeholder roles can execute all their activities efficiently and effectively.

This contribution enables researchers to provide practitioners with a precise definition of what they consider to be low quality, why (i.e. because of which consequences) and in which context (i.e. which activities are performed). Practitioners can then use such a precise quality model and, based on artifacts, activities and impacts, decide which quality characteristics are relevant for their context.

However, defining quality through precise quality factors does not ensure that these quality factors correctly reflect the true impacts in practice. We also have to ensure that we create valid quality models in the sense that the claimed impacts, in fact, hold. In the next section, we report on three studies that we executed to analyze the validity of ABRE-QMs. In addition, also ABRE-QMs can be incomplete through the lack of certain quality factors. We discuss this aspect together with other limitations and extensions in Chapter 5.

4.2. RQ 2: How Can We Create Valid Quality Models?

The contributions for RQ 1 show how to create an ABRE-QM based on explicit quality factors and impacts. The resulting model reflects explicit reasoning behind each aspect of the model. However, reasoning can easily be invalid and impacts can vary by contexts. Therefore, it is essential that impacts are validated.

Currently, since there is no structured and explicit definition of RE artifact quality, there is also no systematic approach to evaluate whether quality characteristics hold or not. In general, there are three approaches to empirically test hypothesis (c.f. Wohlin et al. [WRH+12]): We contribute validation of quality factors based on expert's experience through *interviews*, based on in-depth analysis of individual projects in *case studies*, and, lastly, based on controlled manipulation of variables in artificial settings in *experiments*. In addition, based on three individual studies, we discuss in which situation which approach helps to validate quality factors.

This contribution enables researchers to evaluate whether certain quality factors in fact hold for their activities. However, while we built upon the ABRE-QM meta model, the approaches base on a quality-in-use viewpoint and thus can also be used outside of an ABRE-QM to reason about quality in RE.

This section is based on and summarizes the results of Publications A, D, and C.

4.2.1. Foundation: Defining Artifact Models

The ABRE-QM takes as a prerequisite an artifact model, on which the quality factors are based. However, not always do we have these generic artifact models, as we require in the ABRE-QM (see Section 4.1.1). In case we do not, we need to find ways to create such a model. In the following, we report on the results of such a study. This section summarizes the results of Publication E.

In a cooperation with industry partners, we create such a model in the context of globally distributed teams in agile software projects. The model focusses on agile project management, with a particular focus on agile requirements. In the study, we extending an existing reference models together with our industry partners, and analyzed the differences. Our study shows that large parts of the artifact model must be refined, either by creating new classes or by adding attributes to existing classes. However, the study also shows that even though refinements are necessary, the original model provides enough material to start with. Lastly, our study also raises questions on the dissemination of one specific concept in practice. This could indicate that, for RE artifact quality models, it might not be sufficient to just rely just on reference models.

4.2.2. Summary of the Approaches

To validate the model, we must validate all elements and their relationships. This means that for a specific quality model, we must validate artifacts, entities, stakeholders, activities, quality factors and impacts. Of these, the core of the model are

quality factors and impacts. Therefore, in the following, we largely focus on these two elements during the evaluation[3].

In empirical software engineering, researchers often differentiate three general approaches (see e.g. [WRH+12]): surveys, case studies, and controlled experiments. In the following, we describe each method on its own, providing a description how this technique can be applied to validate a quality factor.

4.2.2.1. Validation of Quality Factors in Expert Interviews

The first approach to validate quality factors are surveys, e.g. in the form of expert interviews[4]. Surveys allow collecting information from people describing their knowledge or behaviour [WRH+12]. Given a certain context, such as a domain, a field, or a specific company, we can employ interviews to validate a large set of quality factors by collecting this information from adequate, knowledgable stakeholders. Our approach consists of four steps:

1. **Select experts:** The first step, subject selection, aims at selecting a representative set of experts for each of the potential stakeholders.
2. **Inspect artifacts & entities:** The second step validates the artifacts and entities, e.g. by iterating through project artifacts. Since in practice this can reveal a plethora of artifacts and entities (see e.g. [MWL+12] for such studies), reduction of artifacts together with local experts can clarify the scope and greatly reduce the required effort.This involves questioning which artifacts exist, which artifacts are in fact used, and how they relate to each other.
3. **Inspect stakeholders & processes:** The third step aims at understanding the role and process model of the project. Therefore, the goal of this step is to develop a complete list of stakeholders and their most important activities.
4. **Validate impacts:** The goal of the fourth step is to validate the quality factors and their impacts. In particular, in this step, we must evaluate the following questions:
   - *Are all relevant quality factors and their impacts present?* If not, we need to extend the quality model and consider also extending the survey.
   - *Are all quality factors and their impacts relevant?* A quality factor and its impacts are *relevant* if it measurably affects the activities of a stakeholder. In case we have irrelevant factors, we must remove them from the model.
   - *Are all quality factors and their impacts accurate?* A quality factor and its impacts are *accurate* if the presence of the quality factor at the entity has the described effect on the activities. When answering this question, we could find counter-examples, in which an impact does not hold. In these cases, we can decide to either remove the quality factor completely, or correct it by splitting up the entities, activities, or even the quality factors. For an example of such an analysis, see the UI design details excerpt in Publication A.

Please note that this process may not necessarily be executed sequentially, but rather iteratively. For example, Step 3 could reveal new stakeholders, which need new expert selection etc.

3 Even though stakeholder analysis itself can be a difficult and sometimes cumbersome activity. For further details, Pacheco and Garcia [PG12] summarize the state of the art in stakeholder identification in their systematic literature review.

4 Surveys can come in the form of interviews or questionnaires [WRH+12]. Even though each of these two methods enables to apply a different set of techniques, the advantages and disadvantages for validating quality factors largely apply for both. In the following, we focus on interviews, since it allows more fine-grained discussion and analysis.

4.2.2.2. Validation of Quality Factors in Case Studies

A case study investigates a certain study object in real life in its original context and in detail, usually applying multiple techniques [WRH+12]. Consequently, one can apply case studies in a constructive manner to understand a quality factor in context or discover new quality factors. Again, the approach follows the elements of the quality model:

1. Select a field and one or multiple cases: As a first step, we need to select the field under analysis and cases, where the data (see Step 4) is available.

2. Inspect artifacts & entities: (See expert interviews).

3. Inspect stakeholders & processes: (See expert interviews).

4. Validate or mine quality factors: In this step, we analyze which problems or defects in the case originated from artifacts and caused negative consequences for corresponding activities. To observe these, one can look through the produced artifacts, e.g.:

- Review protocols that indicate the effort that was invested during QA of the RE artifacts
- Incorrect test cases or incorrect test results that show that the test engineer misunderstood the RE artifacts
- Requirements change requests or defects in bug tracking systems that can be traced back to RE artifacts, to understand defects in the RE artifact that are discovered during development or further activities
- Changes that have been performed on the RE artifacts in order to understand maintenance efforts

Lastly, in line with the suggestions by Wohlin et al. [WRH+12], we suggest to triangulate the results with further sources, such as interviews.

4.2.2.3. Validation of Quality Factors in Controlled Experiments

As a third approach, we can analyze quality factors and their impacts in (controlled) experiments. In experiments, we systematically manipulate individual variables to understand the impact of a certain treatment on further variables [WRH+12]. Consequently, we can apply experiments to analyze the hypotheses we have about a single quality factor and its impacts on activities. Following the terminology of [WRH+12], the experiment design is as follows:

Define treatment: The treatment consists of the different aspects of the quality factor. Therefore, we usually compare specifications where a quality factor is present (*Treatment group*, e.g. passive voice requirements) against those where a factor is not present (*Control group*, e.g. active voice requirements).

Select dependent variables and metrics: The dependent variables are metrics V of the activities (e.g. understanding a specification), according to the impact. This means, that for impacts on effectiveness, we measure the number of errors made, for efficiency we measure the time that is required to execute the activity.

Select subjects: Since the quality model builds upon the quality in use, and the dependent variables are the number of errors or the time required for a certain task, we need human subjects to execute this task. We argue that, as far as possible, the subjects should resemble the real stakeholders of the artifact to prevent confounding factors skewing the resulting data.

Define hypothesis: For the independent variable V and the groups {Treatment, Control}
$H_0 : V(\text{Treatment}) \leq V(\text{Control})$
$H_A : V(\text{Treatment}) > V(\text{Control})$

Select data analysis: Hypothesis testing can then be performed according to standard testing procedures (see e.g. [WRH+12] or [WMMY12]).

All in all, experiments allow analyzing single quality factors in detail.

4.2.3. Summary of Results

We executed each of the aforementioned methods in a different case. In the following, we report on the individual results. We start with a summary of the validation of a quality model in interviews. Afterwards, we briefly explain the results of a case study on quality factors for RE artifact maintenance. Finally, we summarize our experiment on the impact of the quality factor passive voice on understanding RE artifacts.

4.2.3.1. Case 1: Validation through Expert Interviews

This section summarizes the results reported in Publication A.

Goal. The goal of this study, as formally described in Tab. 4.1, was to evaluate whether an ABRE-QM representation of requirements quality definitions helps team members in a software project.

Table 4.1.: Research Goal of Case 1

Evaluate an activity-based quality model **with respect to** the validity of impacts, as well as benefits and drawbacks to validate a company-specific quality definition **from the point of view of** project team members **in the context of** use cases in a industrial business information system development setting.

Study Design. We performed the study at the insurance company Munich Re. As study subjects, we identified a leading requirements engineer and an experienced developer. To create the model, we took the company's 28-pages use case guidelines and translated each of the roughly 50 rules into an ABRE-QM representation. In a first workshop with the experts, we validated artifacts, entities, stakeholders and activities present in their software development environment. In a second workshop, we iterated through each quality factor and impact and discussed the validity of this impact. After the validation, we openly discussed the benefits and drawbacks of such a representation.

Results. For most of the impacts, the translation was straightforward to conduct. However, for a few quality factors, such as *presence of UI details* as well as *the use of subflows*, the explication of arguments revealed tradeoffs for quality factors. For example, the presence of UI details in RE artifacts has various impacts on the further activities. Fig. 4.3 shows a summary of the impacts, both positives impacts on understanding and testing, but also negative impacts on maintenance and use case execution. In Publication A, we discuss this tradeoff in depth. In another example, Munich Re enables the reuse of flows in use cases through *subflows*, a construct that is equivalent to a function call in programming languages. The validation explicated that this concept has advantages during *maintaining* and *reuse*-activities, but disadvantages in *reading* and *understanding*-activities. The

Figure 4.3.: This figure shows an ABRE-QM excerpt for UI Design Details. The excerpt shows how ABRE-QM can be used to systematically analyze quality factors through its impacts.

quality model thus shows this tradeoff and could enable a cost-based evaluation, as is known from activity-based quality models in other disciplines such as source code [DJLW09, Dei09, WLH+12] or system tests [HJE+14, Hau16].

After the workshops, the experts expressed that they see two advantages of such an approach: First, making the impacts explicit and discussing tradeoffs serves as a validation of their own implicit quality understanding, as well as their guidelines. Debating the tradeoffs, in their opinion, could help improving the guidelines. Second, the experts mentioned that the ABRE-QM could help improve the completeness of the guidelines: Since very few of the impacts lead directly to testing, they discussed whether their use cases were optimized for testers or if additional rules are required.

In summary, the results indicate that interviews enable to discuss a larger set of quality factors. For example, in this case, we could discuss a whole quality model in two workshops. This method also enables us to explicate some of the arguments and counter arguments for or against some quality factors. However, when facing unknown or contradicting quality factors, we need more precise and less subjective methods.

4.2.3.2. Case 2: Generating Factors for Maintainability through Change Analysis

The majority of costs in the life cycle of a project do not go into the creation of software, but into its maintenance [Boe81, Gla98]. While quality factors for software maintenance have been studied for source code [DJLW09, Loc13], quality factors for maintenance of RE artifacts is less known. This section summarizes Publication C.

Goal. For this study, we differentiate between semantic changes of RE artifacts, which change the system described, and syntactic changes, which change the description of the system. The goal of the study (as formally described in Tab. 4.2) is to analyze these changes to gain a deeper understanding of maintenance in RE artifacts. This allows us to derive quality factors for unnecessary, cumbersome, and risky activities during RE maintenance.

Table 4.2.: Research Goal of Case 2

Analyze changes in contents of use case artifacts
with respect to frequency, location, change types, and level of risk
from the point of view of requirements engineers in practice
in the context of a large business information system in maintenance.

Study Design. In order to understand how to design RE artifacts for good maintainability, we analyze: How many of the changes in maintenance are semantic changes (i.e. changes of the system described) versus syntactic changes (i.e. changes how the system is described)? In addition, we analyzed, during these changes, what were quality factors of the RE artifact that made changing the artifact (subjectively) difficult?

To answer these questions, we analyzed 15 months of changes in the use cases of an industrial software project in maintenance phase. We developed a typology of changes, based on the differentiation between semantic and syntactic changes and quantified the distribution within this typology. We furthermore identified and quantified related changes. We discussed the results with practitioners and analyzed their perception of difficult changes.

Results. The study shows that roughly every second change in the RE artifact did not change the functionality of the system itself, but only the documentation of the system within the RE artifacts. Since at least the syntactic changes could be avoided if carefully designed upfront, we argue that these efforts show the relevance of maintainability for RE artifacts. Furthermore, certain use cases were especially prone to changes, due to their unfortunate structural organization. We discovered that the maintenance of alternative flows in use cases is of special relevance for maintenance, since these are the parts that frequently change. Lastly, a qualitative and quantitative analysis aiming at a deeper understanding of problematic changes identified the quality factor of a changing terminology as well UI design details causing many changes. Practitioners reported that interconnected use cases negatively influenced maintenance, since keeping consistency in these use cases makes their changes particularly difficult and risky.

All in all, the results indicate the potential of case study research to identify previously unknown quality factors in a relatively unknown field. However, we have to carefully analyze whether the results are specific to this individual project. In addition, in case study research, we cannot control the variables, thus preventing us to draw conclusions about what would have happened if the project had documented their RE artifacts differently. Therefore, this method enables us to explore on unknown territory, but we have to take results with a grain of salt.

4.2.3.3. Case 3: Experimental Validation of a Quality Factor

In a third case, we analyzed a very common quality factor for RE artifacts in depth: passive voice. Passive voice is widely considered a problem in RE [Kof07, ISO11b, Wie05]. Nevertheless, passive voice is widespread in RE artifacts in practice (for examples see [MoD11, Bos07, BHH+03, MCH+12]). Consequently, the question arises: Does passive voice really cause problems in understanding RE artifacts? Therefore, we inspected *passive voice* as one quality factor, in order to understand its impact on *understanding* activities (see Fig. 4.4). This section summarizes the results of Publication D.

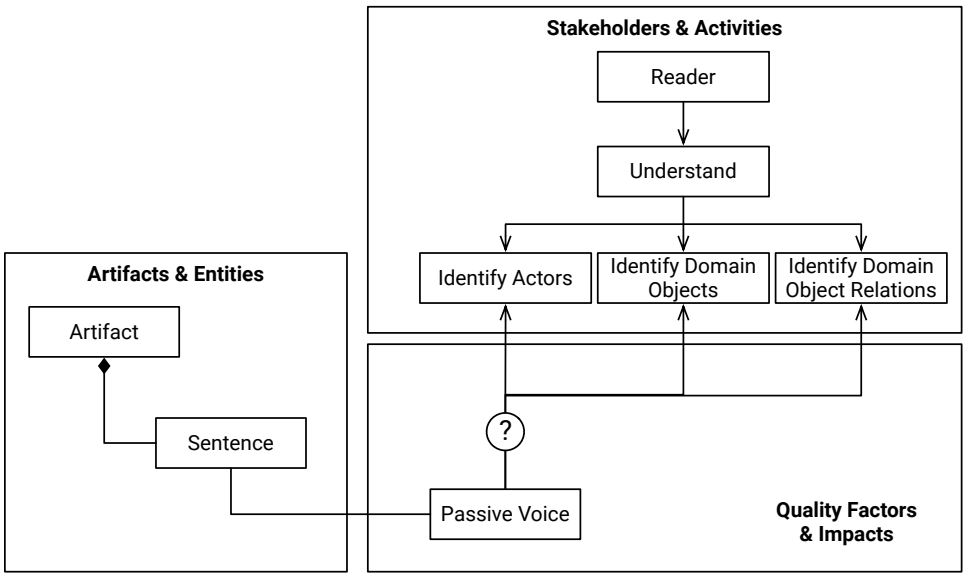


Figure 4.4.: In our experiment we analyzed the impact of passive voice sentences in RE artifacts onto the reader. This figure shows the corresponding ABRE-QM.

Goal. The goal of this study, as formally described in Tab. 4.3, is to understand whether passive voice in RE artifacts causes problems for readers in understanding.

Table 4.3.: Research Goal of Case 3

Characterize the impact of passive voice sentences in RE artifacts on understanding **with respect to** the quality of the artifacts produced from the activities *find actors*, *identify domain objects*, *identify domain object relations* **from the point of view of** the reader of an RE artifact **in the context of** an analysis of requirements from real projects by graduate and undergraduate students in computer science.

Study Design. For this study, we took a set of industrial RE artifacts and extracted requirements that contained sentences in passive voice without agents (*by*-phrase). We then asked our study subjects to provide us with information about the sentence, including the main actor, the main domain objects and their relations (in form of a domain model). We then compared the results of the treatment group with the results of a control group, who received the same requirements translated into active voice.

Results. The results were inconclusive regarding whether or not passive voice hinders identification of actors: Although the active voice group performed better, these results were not statistically significant. The identification of domain objects was equally difficult for both groups. However, the passive voice group performed significantly worse when asked to identify how the domain objects relate. Many participants in the treatment group were even unable to relate objects at all (see Fig. 4.5).

This experiment indicates that taking a quality-in-use perspective allows us to more precisely reason about quality factors. In addition, experiments can be used to inspect effects of certain quality factors in depth. However, experiments are a very costly endeavour, both in terms of time and personell. In addition, it might not be possible to study all effects (e.g. long-term-effects) in such an artificial environment with reasonable time and money.

4.2.4. Conclusion to RQ 2

In this section, we contribute three approaches to analyze and increase the validity of an ABRE-QM.

The three studies provide a picture of the strengths and limitations of the three approaches: Expert interviews enable to validate a large set of well-known quality factors in the domain and experience of the expert. The biggest threat here is that all acquired evidence is based on the (potentially biased) knowledge of experts. However, expert interviews might be difficult in rather unknown fields, such as RE artifact maintenance. In these cases, exploratory studies, such as case studies, provide valuable knowledge. However, they require significantly more effort to conduct, and might also not be generalizable to other cases. Lastly, for specific quality factors, experiments can analyze their impact in artificial contexts, yet only with considerable effort. Although this is the most precise of the methods since it allows controlling of many variables, this comes with the risk that results might not generalize to an industrial setting. In addition, conducting experimental analysis

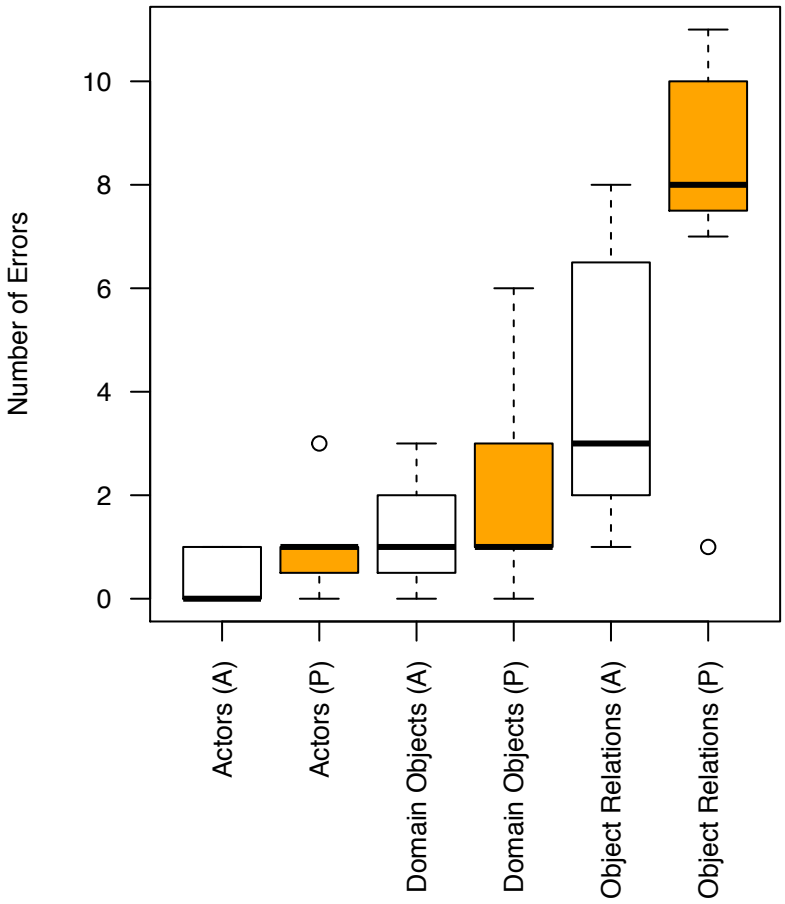


Figure 4.5.: This figure shows the boxplot graph of the results of our experiment. It compares the number of errors of participants with active voice RE artifacts (A, white) with passive voice RE artifacts (P, orange). Our experiment showed a non-significant impact onto identifying actors, no impact onto identifying domain objects and a significant impact onto identifying domain object relations.

might not always be possible for the complex activities that a software engineering project comprises.

In summary, the studies indicate the benefits of ABRE-QM for both industry and academia: For industry, our interviews revealed the potential benefits in terms of validity and completeness of existing quality models. Furthermore the model can support to weigh the different quality factors through quantification of the activities. ABRE-QM shows how to create a quality model that motivates each quality factor in a precise manner, which enables employees to understand and discuss each factor, depending on the project context. For academia, the model can establish a common language to compare and combine studies and, in the long term, built a common theory of RE artifact quality. The falsifiability of quality factors can lead to a more fact-based discussion of quality in RE.

In future, we envision that knowledge in RE artifact quality is encapsulated in a common theory. ABRE-QM provides the language, as well as the theoretical basis for such a theory. We furthermore show three approaches how hypotheses in such a theory can be evaluated. However, other quality factors in future work might require variations of these approaches (e.g. in the direction of global, instead of local surveys [MW15]).

4.3. RQ 3: How Can We Efficiently Ensure Quality Factors?

ABRE-QMs enable to model quality from an activity-based perspective. However, such an understanding of quality forms just one side of the medal. When we analyse widely accepted quality factors in industrial RE artifacts, we see that violations of these quality factors are widespread (to give just one example, see the number of passive voice requirements in [MoD11, Bos07, BHH+03, MCH+12]). To understand this phenomenon, we contribute an analysis of challenges of RE artifact QC at Munich Re in Publication H. One of the results in this study is that pracitioners need faster and more efficient methods to control RE artifact quality in practice. To takle the inefficiencies in RE artifact QC, we contribute an approach for automatic detection of a set of quality factors in natural language RE artifacts. We call this approach *requirements (bad) smells*.

This section provides definitions, relates the approach with ABRE-QM, and gives a taxonomy of requirements smells and requirements smell detection approaches.

4.3.1. Summary of Approach: A More Efficient Process for RE Artifact QC

To understand the challenges of RE artifact QC, we conduct a case study at Munich Re in which we analyze the challenges in practice. Based on this case study, we propose an RE artifact QC process that combines manual with automatic QA. This section summarizes Publication H.

Goal. QC of RE artifacts struggles in practice. Mendez and Wagner [MW15] report that less than a third of requirements engineers perform analytical QA of RE artifacts. However, the concrete challenges of RE artifact QC remains an open question: We lack knowledge on the challenges of real-world requirements QC processes, and efficient processes for RE (artifact) QC.

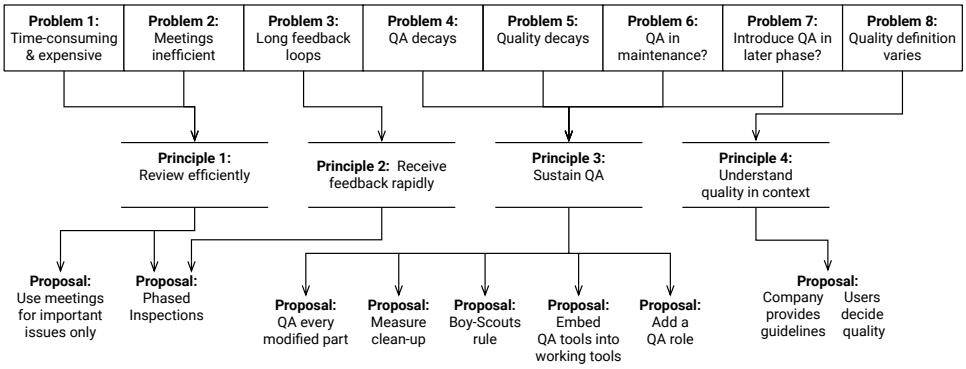


Figure 4.6.: This figure visualizes the resulting challenges and proposals for a more efficient QC process.

Study Design. To understand these problems, we report on an first exploratory case study, in which we analyze the challenges at Munich Re, one of the largest reinsurance companies world-wide, to understand the individual problems of requirements engineers in practice. Based on the outcomes, we develop a process that aims at controlling these challenges to enable a sustaining QC process for RE artifacts.

Results. In our case study, we found that the main problems in QC for RE artifacts in this case were a missing common quality understanding, the low feedback speed, low efficiency in the QC process, and, consequently, the lack of creating a sustaining QC processes. Based on these challenges, we make nine proposals to increase the efficiency of the RE artifact QC process (see Fig. 4.6). To combine the proposals, we contribute a process for requirements artifact QC that is designed to address these problems through various techniques. We discuss feasibility and impact of the process with industry, who acknowledge its potential to increase efficiency and to provide a more sustaining QC process in practice.

4.3.2. Summary of Approach: Requirements Smells

In the previous study, we argue that two of the main challenges of RE artifact QC today is that manual QA is inefficient and slow. We furthermore motivate to combine manual with automatic QA to ensure certain quality factors already during the time of writing, similar to a spell-checker. In the following, we explain the concept of *requirements smells* and give a taxonomy of requirements smells.

This section is based on Publications F and G, and extends them through the relation to ABRE-QM and a taxonomy of requirements smells.

4.3.2.1. Defining Requirements Smells

A requirements smell is an indicator of a quality violation, which may lead to a defect, with a concrete indication and a concrete detection mechanism.

In detail, we consider a requirements smell as having the following characteristics:

1. A requirements smell is an *indicator* for a quality violation of a RE artifact. For this definition, we understand requirements quality in terms of quality-in-use, meaning that bad RE artifact quality is defined by its (potential) negative effects on activities in the software lifecycle that rely on these RE artifacts.

2. A requirements smell does *not necessarily lead to a defect* and, thus, has to be judged by the context (supported e.g. by (counter-/)indications). Whether a requirements smell is or is not a problem in a certain context must be individually decided for that context and is subject to reviews and other follow-up quality assurance activities.
3. A requirements smell has a *concrete indication* (symptoms) in an entity of the RE artifact itself, e.g. a word or a sentence. Requirements smells always provide a pointer to a certain location that QA must inspect. In this regard, it differs from general quality characteristics, e.g. completeness, that only provide abstract criteria.
4. A requirements smell has a *concrete detection mechanism*[5]. Due to its concrete nature, requirements smells offer techniques for detection of the smells. These techniques can, of course, be more or less accurate.

Furthermore, we define a *quality defect* as a concrete instance or manifestation of a quality violation in the artifact, in contrast to a *finding* as instances of a smell. However, like a requirements smell indicates for a quality violation, the finding indicates for a defect. Fig. 4.7 visualizes the relation of these terms as well as the given definition of requirements smells.

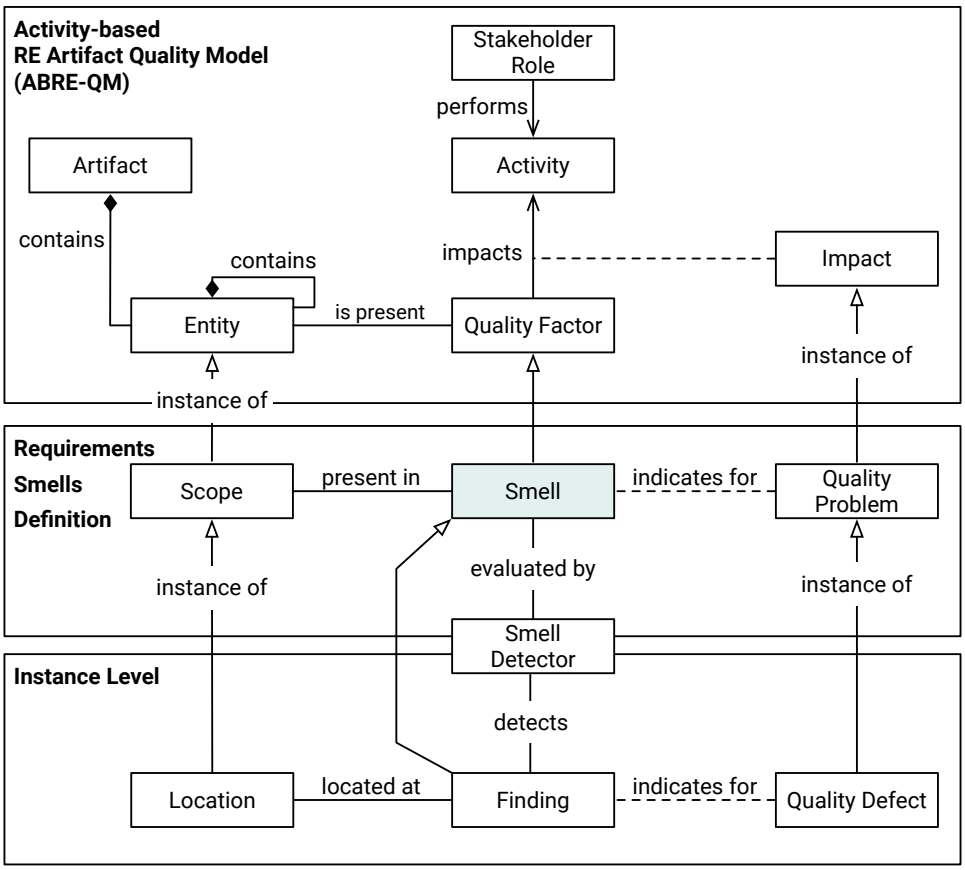


Figure 4.7.: This figure shows terminology of requirements smells. In this terminology, we define requirements smells as specific quality factors of ABRE-QMs.

5 An instance of an *Assessment* in the terms of ABRE-QM in RQ 1.

In the following, we focus on natural language requirements smells, since, as we discussed in the introduction, requirements are mostly written in natural language [MFNI04]. Furthermore, as motivated by our study in Publication H, the best benefits of requirements smell detection come with automation. Therefore, the remainder of the thesis discusses requirements smells where the detection mechanism can be executed automatically (i.e. it requires no manual creation of intermediate or supporting artifacts).

4.3.2.2. Defining a Requirements Smell Taxonomy

Various natural language requirements smells exist (see the appendix to Publication G for an overview). In the following, we present a classification of these smells according to their scope (see Fig. 4.7). This classification allows to understand which types of requirements smells exist.

As defined above, each smell has a detection mechanism, is indicator for a quality problem, and has a scope on which a requirements smell can potentially exist. In each scope, different quality problems exist, and for each scope, different detection mechanisms are required. In the following, we detail these scopes, before explaining our requirements smell taxonomy.

Scopes for Requirements Smells Entities, as defined in the ABRE-QM, can be broken down deeper than just the level of artifact model (e.g. use case documents, use case steps, etc.) but also into what we call the *language model*. The language model reflects the state of the art natural language processing and includes information on sentence boundaries and other grammatical features, such as the parse tree [JM14, SK13] or Part-of-Speech (POS) tags [JM14, GE09, HEZ15]. An excerpt of such a resulting model, including the language model, is depicted in Fig. 4.8. In this figure, we can see the artifact model with entities on top, where each element that is built on natural language text, e.g. *Step* is a *TextItem* containing one or more text chunks. These text chunks are broken down into its grammatical features. The features included in this language model represent the grammatical features of state-of-the-art NLP (based on the popular collection of NLP methods Stanford Core NLP [MSB+14] and DKPro [dCG14]) and might grow with research progressing in NLP.

Requirements Smell Types Based on the scopes provided above and the definition by Berry et al. [BBG+06], we suggest the following classes of requirements smells.

Structural Smells refer to smells within the artifact model. These can be of either of three typical types:

1. {min,max} Number of content items, e.g. a use case must have between 5 and 9 steps. This includes also the presence or absence of content items.
2. Naming conventions: E.g. that all use cases are two-digits, numbered and preceeded by a project abbreviation.
3. Coordination of elements, i.e. each element at a certain location must have an identical element at another location. One example is the factor that each referenced step in a flow must be defined elsewhere.

Grammatical Smells refer to smells that have a grammatical feature as the scope[6]. Grammatical smells are mostly in the form of counting of a grammatical

6 This is strongly related to the class of *syntactic* manifestations of Berry et al. [BBG+06]. Berry refers to the meaning of syntax as defined by Chomsky [Cho57]. However, we consider grammatical smells a more fitting category, due to the different meanings of syntax in the linguistic and computer science world: In computer science, every automatic requirements smell

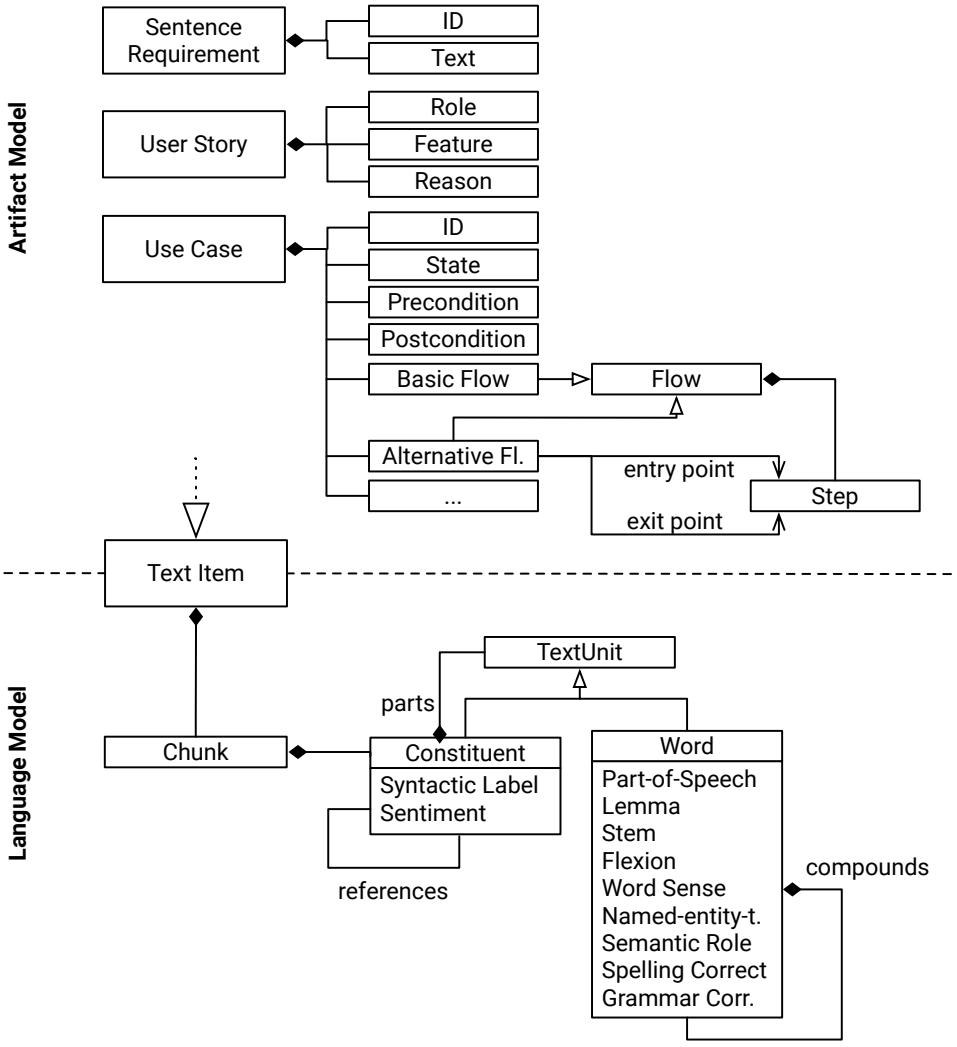


Figure 4.8.: This figure visualizes a detailed view on the entities of an ABRE-QM in the context of requirements smells. It shows the interplay between artifact model on the top and language model on the bottom. For simplicity, we omitted relations between textual entities of the artifact model and their parent class *Text Item*.

feature (e.g. number of subclauses), including the presence of absence of a grammatical feature (such as superlatives or passive voice).

Lexical Smells refer to smells where a word or phrase in itself is problematic in RE artifacts, e.g. due to ambiguity. Lexical smells are mostly in the form of presence or absence of certain combinations of words.

Semantic Smells are those smells that aim at detecting problems in the content that is described. For example, the smell *UI Design Elements in Use Cases* is able (based on word lists) to understand that a use cases probably describes something that is considered a problem in maintenance (see also Section 4.2.3.2 and Publication C for further details on this quality factor). Since the semantics can only be detected automatically if it is reduced to syntactic features, semantic smells must be broken down into syntax (such as the other smells) to automatically detect findings.

Please note, that these are the basic types of requirements smells. The concrete smells often form a mixture of these elements, e.g. the naming convention requires that the use case starts with a verb, and thus needs some grammatical information.

This smell definition allows to classify different types of requirements smells. Furthermore, this taxonomy might allow future work to define requirements smell detection languages, as well as false positive filters for each type of requirements smell.

4.3.3. Conclusion to RQ 3

Even when practitioners widely agree on quality definitions, the quality of RE artifacts in practice often does not live up to these definitions. In this section, we first contribute an analysis of problems in today's RE artifact QC as well as the concept of requirements smells which allow to detect certain quality factors automatically and thus, faster and more efficiently.

The case study shows eight problems in RE artifact QC, which can be summarized in four challenges: RE artifact QC must be more efficient, provide faster feedback, must sustain in projects over time, and must reflect the quality definition in the respective context.

For the latter, RQ 1 and RQ 2 provide ABRE-QM to define quality from a quality-in-use viewpoint. For the former two, we argue that through a combinations of automatic and manual QA techniques, RE artifact QC can provide feedback quicker and more efficiently. Therefore, we contribute a definition and a taxonomy of requirements smells, and explain how requirements smells relate to ABRE-QMs. Furthermore, in Publication G, we contribute a technical validation of requirements smell detection by means of a prototype. However, to understand whether such a combination of techniques leads to a more sustainable QC process is subject to longitudinal studies, which we leave to future work.

In order to understand the benefits of requirements smell detection in practice, we need to empirically analyze to which extent requirements smells are present in RE artifacts, as well as which defects we can find, and which defects we cannot find. We focus on these questions in the next section.

detection must be syntactical, since computers only work on syntactical elements. Therefore, we refer to the smells of this category as grammatical smells.

4.4. RQ 4: What Are the Benefits and Limitations of Requirements Smell Detection?

In the previous section, we described requirements smells, an approach for automatic support in RE artifact QA. Yet, so far it is open how widespread the problem is in practice, whether the approach produces relevant findings for practitioners, how accurate the findings are, and also which issues the approach is able and which issues the approach is not able to find.

To build this understanding of smells in requirements artifacts, we conducted an exploratory multi-case study with both industrial and academic cases.

This section summarizes the empirical studies of Publications F, G, and I.

4.4.1. Summary of Approach

In the following, we briefly summarize the research questions as well as the cases that we analyzed in Publication G.

4.4.1.1. Research Questions

The research objective, to analyze whether automatic detection of requirements smells helps in RE artifact quality assurance, can be broken down into four research questions:

RQ 4.1: How many requirements smell findings are present in RE artifacts? To see if the automatic detection of requirements smells in RE artifacts could help in QA, we first need to verify that requirements smells exist in the real world. The answer to this question fosters the understanding how widespread the requirements smells under analysis are in industrial and academic RE artifacts.

RQ 4.2: How many of these findings are relevant? Not only the number of detected requirements smells is important. If many of the detected requirements smells are false positives and not relevant for the requirements engineers and developers, it would hinder QC more than it would help. As relevancy is a rather broad concept, we break down RQ2 into two sub-questions.

> **RQ 4.2.1: How accurate is the requirements smell detection?** The first sub-question looks at the more technical view on relevance. We want to find false positives and determine the precision of the analysis in terms of correct detection of the defined requirements smell.
>
> **RQ 4.2.2: Which of these findings are practically relevant in which context?** This second sub-question is concerned with practical relevance. We investigate whether practitioners would react and change the requirement when confronted with the findings.

RQ 4.3: Which requirements quality defects can be detected with requirements smells? After we understood how relevant the analyzed requirements smells are, we want to understand their relation to existing quality defects in RE artifacts. Hence, we need to check whether, and if so, which defects in RE artifacts correspond to requirements smells, as we understand requirements smell findings as indicators for defects.

RQ 4.4: How could requirements smells help in the QC process? Finally, we collect general feedback from practitioners whether (and how) requirements smell detection could be a useful addition to QC for RE artifacts and whether as well as how they would integrate the requirements smell detection into their QC process.

4.4.1.2. Case Description

We performed a multi-case study with four different cases. Three of these were industrial cases from 12 projects of three different companies. In addition, one of the cases was academic, where we analyzed RE artifacts of 51 different teams. In total, we analyzed RE artifacts of the size of more than 265,000 words.

The first three cases contain requirements produced in different industrial contexts: embedded systems in the automotive industry (Daimler AG), business information systems for the chemical domain (Wacker Chemie AG) and agile development of web-based systems (TechDivision). While the first two represent rather classical approaches to Requirements Engineering, the third case applies the concept of user stories, as it is popular in agile software development. The fourth case has an academic background (University of Stuttgart) and employs use cases and textual requirements. Regarding subject selection, for each industrial case we selected practitioners involved in the company, domain and specification.

For practical reasons, we could not evaluate each research question in each case: For example, RQ 4.3 depends on the existence of reviews with documented results, which often not exists in practice. Furthermore, depending the answers of RQ 4.4 on the potentially less experienced students from Case D would introduce a threat to the validity of our evaluation.

4.4.2. Summary of Results

We empirically analyzed whether or not the concept of requirements smells also holds in practice. In the following we shortly report on the results.

RQ 4.1: How many requirements smell findings are present in RE artifacts?

The number of findings in RE artifacts strongly correlates with the size of the artifact (see Fig. 4.9, Spearman correlation of 0.9). There are roughly 44 findings per 1,000 words and some contexts show a striking similarity in the number of findings for their artifacts. In our cases, the automotive requirements had a lower number of findings whereas student artifacts contained a higher number of findings relative to the size of the artifacts. The most common findings are for the smells loopholes and vague pronouns.

RQ 4.2: How many of these findings are relevant?

Precision A manual classification of a set of more than 600 findings resulted in an deeper understanding of the precision of requirements smell detection. As shown in Fig. 4.10, the precision is on average around 59% but varies between requirements smells. We consider this reasonable for a task that is usually performed manually. However, this also depends on the relevance of findings to practitioners, which we also analyze in RQ 4.2. The study also reveals improvements for future work through the application of deeper NLP.

Recall When analyzing the accuracy of an automatic detection, we must look not only at precision, but also at recall, i.e. the ratio of all detected findings to all defects of a certain type in an artifact. To this end, we inspected one artifact of each case, in total a set of roughly 16,200 words, and manually identified the findings in each artifact. The manual inspection revealed 200 findings in this artifact sample and an average recall of 0.82. Fig. 4.10 shows the summary of the results: The comparison

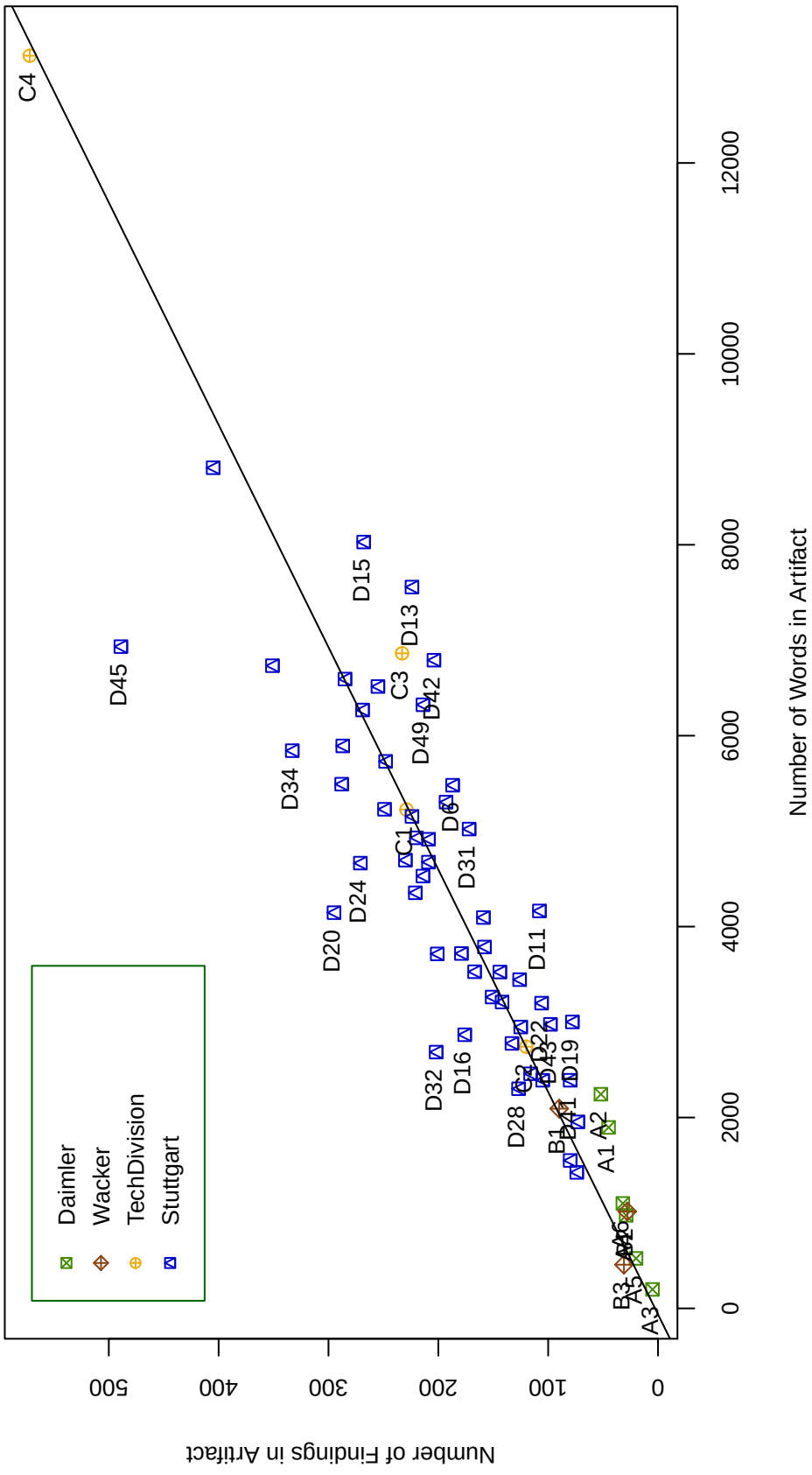


Figure 4.9.: This figure shows how the number of findings strongly correlates with size of artifact. For readability reasons, for the Stuttgart cases (blue) only IDs of less correlating artifacts are displayed.

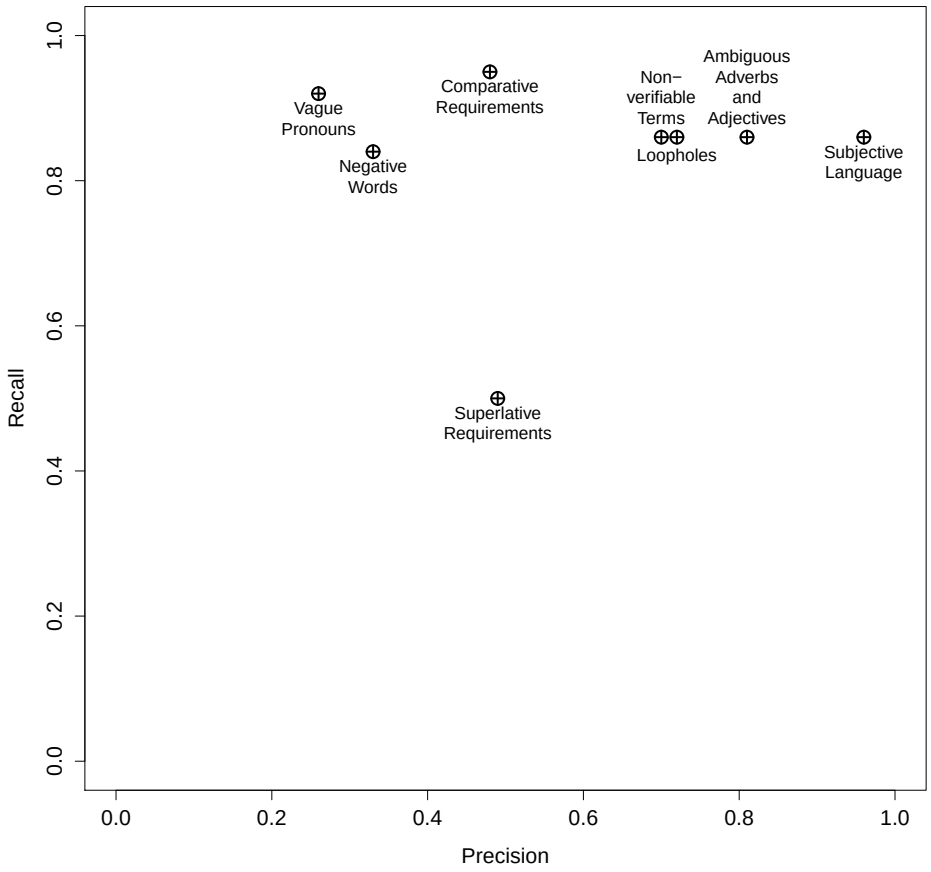


Figure 4.10.: This figure shows the precision and recall of various requirements smells.

shows a recall between 0.84 and 0.95 for four of the five investigated smells. The highest recall was achieved by the comparative smell, with 0.95, which means that the smell detection missed one in 20 findings. The fifth smell, with the lowest recall, is superlative smell with a recall of 0.5. However, this smell is one of the rarest of the smells, as one can also see in the results to RQ 4.1. Therefore our analysis of the recall of this smell is based on few data points. Hence, we suggest to take the recall of this smell with care, and suggest that future studies should investigate this issue in more depth.

Practical Relevance Publication F indicated that requirements smell detection is able to find relevant defects. In Publication G, we analyzed this in more detail in the Case TechDivision. In summary, the practitioners in Case TechDivision expressed that 65% of the discussed findings were relevant, as they lead to lengthy discussions and unnecessary iterations in estimation. They also saw the problem of legal binding, but in contrast to the practitioners of Case Daimler and Wacker, they considered these findings less relevant due to their Agile software development approach. Based on these results, they expressed their strong interest in exploring smell detection for projects; we explain the results of this discussion in RQ 4.4.

RQ 4.3: Which requirements quality defects can be detected with smells?

The comparison of smell detection results and review findings revealed that automatic smell detection can point to issues in both representation and content (see Figure 4.11). The defects, as reported from the reviews, indicate that more defects can be automatically detected. Just as for static code analysis, however, we see that automatic analysis can only indicate some defects and thus must be accompanied by reviews [WJKT05]. In particular, we identified three types of defects that are hardly detectable automatically: First, subjective and fuzzy defects, that even the reviewer cannot clearly specify. Second, defects in which deep semantic understanding of the text is required. Third, defects that require common-sense reasoning, domain knowledge or other additional information that is not documented in the RE artifacts.

In-depth Analysis In Publication I, we deepened our understanding of the relation of defects that can and defects that cannot be detected automatically. Therefore, we analyzed a complete set of 166 rules from the RE artifact guideline used at a large Swedish company. The analysis brought three key insights: First, we estimate that a substantial share of the rules can be automatically checked: Only 25% of the rules cannot be checked automatically, 41% of the rules could be deterministically detected, 12% can be checked with good accuracy heuristics, 11% with medium and low-accuracy heuristics, respectively (see Fig. 4.12). Second, the techniques that are needed for checking these rules are usually simple techniques, such as regular expressions or lemmatization [JM14]. And third, the main reason why a rule could not be automatically checked, was not a need for better NLP, but a need for more precisely defined rules in RE artifact guidelines.

RQ 4.4: How could requirements smells help in the QC process?

The qualitative study with three industrial cases could show a general agreement on potential benefits of using smell detection a quality assurance context. When asked how they would integrate the requirements smell detection, they see possibility for both analytical and constructive QA, provided, however, this integration would not increase the required effort, e.g. by integrating the detection into existing tool chains.

4.4.3. Conclusions to RQ 4

These results indicate that requirements smell detection can detect findings that are relevant to practitioners. It shows that requirements smells are present across various domains and approaches, yet with varying density. The requirements smell detection's precision and recall varies between the approaches, but has a high potential to improve based on the false positives detected in the study. The study furthermore provides a map of which kind of issues can and which cannot be detected. Lastly, a further analysis of guideline rules indicates that a substantial share of quality factors can be checked automatically.

Figure 4.11.: This figure shows findings in requirements reviews, classified by content/representation and estimated detection accuracy.

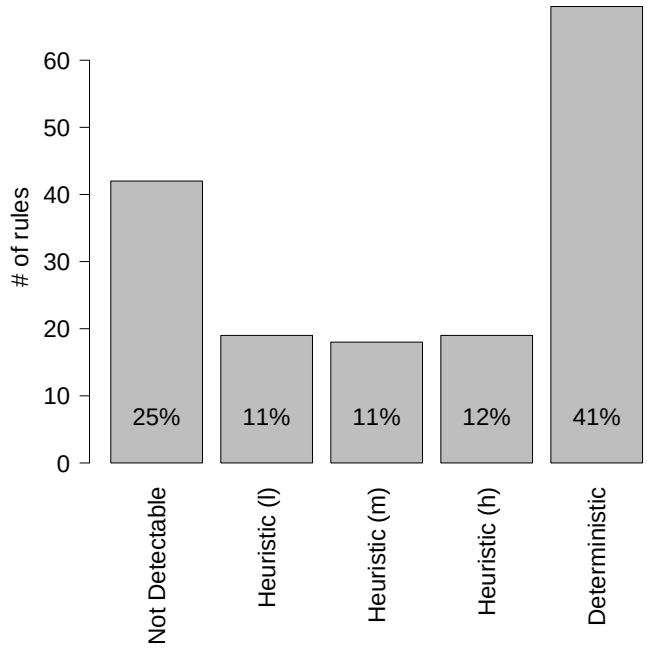


Figure 4.12.: This figure summarizes the estimated detection accuracy of RE artifact guideline rules used in a large company.

CHAPTER 5

Discussion of Results

After proposing and evaluating ABRE-QM and requirements smells, we discuss the strength and limitations of both of these approaches. This section summarizes the experiences that we collected in the studies of Publications A to I. In the following, we first we discuss strength and limitations of ABRE-QM, then of requirements smell detection. Afterwards, we go back to the quality criteria as discussed by current standards and analyze to which extent these quality criteria can be covered by automatic requirements smell detection. Lastly, we discuss whether the number of findings produced by requirements smell detection can predict the success or failure of projects.

5.1. Strengths and Limitations of ABRE-QM

The ABRE-QM meta model defines RE artifact quality as a set of quality factors, i.e. properties of an RE artifact entity with a defined impact onto activities of RE artifact stakeholders. This approach comes with several advantages which we want to discuss, before analyzing limitations of such a viewpoint.

5.1.1. Strengths of an ABRE-QM

We argue that ABRE-QM improves the status quo of RE artifact quality along the following paths.

ABRE-QM provides a more precise quality definition. We argue that ABRE-QM enables both researchers and practitioners to reason more precisely about RE artifact quality. For researchers, ABRE-QM provides a defined view onto RE artifact quality by taking activities into the focus of quality. This provides a simple rule whether something is of higher or lower quality: If it hinders a user of the artifact, it is of low quality; if it makes an activity more efficient or effective, it is of high quality. For example, in Fig 4.3, we could show how an ABRE-QM enables to reason whether requirements engineers should or should not avoid UI design details in their use case steps. This allows practitioners, as reported in Publication A, to question and validate their existing quality models in a simple and comprehensible manner.

ABRE-QM provides a more adequate quality definition. In many projects, the activities differ. Therefore, some activities, such as conducting impact analysis, might not be required in an iterative, short-cycled project. We consider this to be a strength of the ABRE-QM, to be able to taylor the quality model along the activities: If an activity is especially important, the quality factors of this activity should be taken more care of than others. Equally, if teams consider some activities irrelevant, the quality factors for these activities can be dropped. We can also see this in the aforementioned example in Fig 4.3, where it depends whether the RE artifacts are going to be maintained or not. In summary, we argue that the model takes the context (i.e. the activities in the project) into account and thereby models quality more adequately.

ABRE-QM provides a first notion of completeness of the quality definition. The quality model provides a guideline for understanding whether a given set of quality factors is complete. Through ABRE-QM, we can reason along the stakeholders and activities to understand if all relevant quality factors are covered. This means that if we consider all stakeholders, as well as all their activities with an artifact and understand what makes their activities efficient and effective, our quality model is complete. For example, in RQ 2 practitioners realized through an ABRE-QM that their use case guidelines did not take testers into account.

ABRE-QM provides a unified understanding and language. We argue that ABRE-QM provides a unified understanding, as well as a novel language to combine different pieces of knowledge for RE artifact quality. This could enable researchers and practitioners to collect, compare and combine hypothesis in this field. Steering the work in the community into the same direction, it would allow in the long run to build a combined theory of artifact quality in RE.

5.1.2. Limitations of an ABRE-QM

Projects in software development are based on human interaction. Thus, real processes are usually not straight, clean and defined, but instead iterative, complex and intertwined. This makes it hard to understand the dependencies and causalities in projects. This difficulty is also reflected in the limitations of ABRE-QM, which we describe in the following.

ABRE-QM does not provide a common quality definition. In this thesis, we do not provide a complete, widely applicable quality model. This is outside the scope of this work, since each quality factor and its impacts must be studied with care. Instead, we provide a language and show exemplarily how quality factors can be validated in RQ 2. In this direction, a large open question is whether such a widely applicable quality model exists or whether projects are too different to create a shared understanding of RE artifact quality. ABRE-QM and survey-like approaches as explained in RQ 2 enable future research to systematically study this question.

ABRE-QM requires trade-offs between precision and complexity. During our work for RQ 2, we often discussed how deep we should decompose entities, activities and quality factors. The question is very difficult to answer up-front, since it reflects the trade-off between the two purposes of the model. We can either detail the elements, which enables to explain more exactly where and why something is a quality defect. While this increases the validity and preciseness, it comes at a cost: We would argue that complex and detailed models are harder to communicate

and more difficult for practitioners to grasp and use. Section 4.3.2.2 shows this complexity, where we detail entities further and further down into single words of a language model. We argue that at this level of detail, the model could be difficult to be used company-wide (e.g. through guidelines in RQ 2).

Yet, when we started developing quality models based on guidelines, this trade-off was less problematic than expected. Whenever possible, we stayed on the abstract level, e.g. the testing activity. Yet, when discussions arose, we solved the conflicts by detailing the activities and quality factors. Therefore, the model remains abstract, unless validity is violated or questioned. However, the extent to which this trade-off between precision and complexity is a problem, remains an open question.

ABRE-QM does not model human compensation. Sometimes, impacts that exist on the paper, are solved through the individual human that performs the task. For example, if a requirement is hard to understand, the person might reflect on how the product was built last time. Or the person could ask for clarification in direct conversation. However, both examples also indicate that this compensation comes at cost and risk. Accordingly, the extent to which defects can be compensated (and at which cost and risk) is still an open question for future research. This also relates to the question of the relation between artifact and process quality: Under which circumstances does a low quality process lead to low quality artifacts and vice versa? This question is, unfortunately, still unanswered [Men15].

ABRE-QM struggles with transitive, feedback and non-linear impacts. Some impacts do not cause problems for the direct user of RE artifacts, but impacts stakeholders later in the process. For example, if the artifact contains requirements that do not reflect the customer's needs, this is not a problem for the developer or the tester. Unless confronted with the user, their activities are just as easy to perform. However, the product of their work is given to the user and will cause problems. Therefore, an issue in the RE artifact may cause problems only transitively. Such transitive effects are called second- or third-order effects, depending on the distance between cause and effect [HSW05].

We integrated second-order impacts into the model. For instance, the stakeholder `Customer` in Fig. 4.3 does not read the RE artifact, but uses the software that was built by someone else, who used the RE artifact. For an idea how to model these effects in more detail, see our discussions on second- and third-order effects in sustainable software engineering [PF13]. However, the real difficulties would become visible if we tried to evaluate such an impact, similar to the experiments in RQ 2. To take this thought further, systemic or third-order effects (such as changes in behavior, changes in the socio-economic environment), are even harder to reflect and study in such a model. The same holds for feedback and non-linear impacts. The open question remains: How can we study these types of effects of quality factors in RE artifact quality?

ABRE-QM does not solve limitations for empirical validation of impacts and limited knowledge in RE. This thesis shows a general approach how to validate quality factors in an ABRE-QM. However, empirical validation in RE has its own limitations and constraints, including, but not limited to, unmeasurable quality factors, generalization from instances to the whole, and further confounding factors. Related to this, we still have very limited knowledge about the impact of quality factors and the existing causalities. The quality model can steer future research through a common understand of quality, and structure future research through a common model.

5.2. Strengths and Limitations of Automatic Requirements Smell Detection

To bring QC of ABRE-QMs into practice, we propose requirements smells in order to more efficiently conduct RE artifact QC. At the core of our approach is automatic requirements smell detection, which enables cheap, fast and consistent QC of quality factors. In this section, we summarize strength and limitations of such an automatic approach, based on the results of RQ 4 in general, and RQ 4.3 in particular.

5.2.1. Strengths of Automatic Requirements Smell Detection

As discussed in Chapter 1.1, several arguments suggest to apply automation in QC tasks:

Automation is cheap. As we analyze in Publication H, Reviewing plus its coordination is an effective, yet time-consuming task. In contrast, after the initial setup, automatic tasks are cheap to execute (cf. the cost model for automatic versus manual system testing by Hauptmann et al. [HJE+14, Hau16]).

Automation is fast. Reviews can take multiple weeks until an author receives feedback (see our work in Publication H). In contrast, automatic feedback depends only on the runtime of the system. NLP tasks, such as part-of-speech tagging or syntactic parsing, are usually the most time-consuming part of requirements smell detection and take far below a second to execute per sentence [GE09, HEZ15].

Automation is consistent. If reviews are performed by using an ad-hoc reading technique (i.e. no checklists or similar as classified by Katasonov and Sakkinen [KS05]), they do not systematically evaluate RE artifacts against each quality factor. Since human reviewers are not good at simple, repetitive tasks, we argue that they tend to make more mistakes than automatic algorithms for these types of quality factors. Therefore, for certain quality factors, such as passive voice, we argue that automation is more consistent than reviews.

5.2.2. Limitations of Automatic Requirements Smell Detection

Based on the results of RQ 4.3 in Section 4.4.2 (see Publication G for more details), we see the following limitations to automating requirements smell detection. Some of the results are engineering problems that can be dealt with through development of more sophisticated solutions, yet others are logical boundaries that cannot be overcome.

Automatic requirements smell detection requires an explicit quality definition. First of all, in order to be able to detect a quality factor automatically, the factor must be known and defined. This is not the case when quality factors are subjective or fuzzy. Yet, as we describe in RQ 4.3 in Chapter 4.4.2, sometimes quality factors are fuzzy even to the reviewers themselves. For example, it can be hard to explain why a certain design is not appealing to certain users, but an experienced reviewer can still see this issue.

Automatic requirements smell detection struggles with noise in industrial data. Industrial data often comes with noisy data. For example, in our studies we had to deal with wrong grammar and incomplete sentences (see qualitative results in Publication G). Our approach is based on NLP, therefore the accuracy of the approach cannot be better than the accuracy of NLP. Yet NLP and therefore also our requirements smell detection is trained and based on the assumption that the RE artifact is in a solid, readable state, which produces unavoidable errors.

But even when confronted with correct grammar, NLP approaches often have their shortcomings. For example, we experienced low accuracy when we confronted our NLP libraries with short sentences (< 3 words). However, depending on the use case, one can circumvent some problems this with heuristics and combination with non-NLP methods, such as ontologies.

Automatic requirements smell detection has no deep semantic understanding of text. So far, deep automatic semantic parsing of text, such as understanding contradictions between sentences and understanding a sentence in its context, is still out of reach [CW14]. However, shallow semantic[1] parsing [PWH+04], takes a very first, but promising step in that direction. To this end, our automatic understanding of texts is limited to grammatical information and heuristics (e.g. the actor of a passive voice sentence is indicated through a *by*-phrase).

Automatic requirements smell detection has no knowledge of domain and common sense. Some issues in RE artifacts origin from semantic defects (What system is described?) instead of syntactic defects (How is the system described?). For example, a practitioner reported on an issue where the engineers had created a deployment plan, in which a certain device needed to be relocated into a different region. Under review, the domain expert explained that the proposed solution would not work, since the device was too large to be transported in the regular manner. Here, the reviewer found a conflict between his domain knowledge of the existing devices in the system and his understanding of the physical world.

But without further input, an algorithm cannot know the domain semantics. Therefore, to detect issues involving domain knowledge would need to include further information. For simple cases one option could be to include glossaries into the detection mechanism as it is presented by Hauptmann et al. [HJE+13] for test cases. For more complex examples, future work must assess whether there are rules that, one the one side, can be detected with semantic technologies in NLP, but, on the other side, are also widely applicable across projects.

Automatic requirements smell detection does not know the goal of the system and the current project status. It is not an engineering task, but a conscious decision to define which goals a system should fulfil, and accordingly which requirements must and must not be part of the system. Very often these goals are not explicitly stated in the RE artifact. Therefore an algorithm cannot understand missing requirements or requirements that are semantically correct, but are against the stakeholders goals. The same holds for the current project status. The status is a matter outside of the RE artifact and usually not accessible by automatic requirements smell detection.

1 Please note that in the field of natural language processing the term *semantic* refers to the understanding of words of a sentence and their relation to each other.

5.3. Which Quality Characteristics Can We Detect Automatically?

When discussing the question which defects can be detected automatically, we have to define the population of defects. There are two approaches towards defining this population: Either we take definitions of quality as the population, or we refer to their instances in the form of defects in RE artifacts. In Publication G, we performed the latter type of analysis, in Publication I, we performed the former (see Section 4.4.2). In the following, we want to further discuss the former view.

For the sake of the argument, we assume quality to be defined with the quality models of the ISO 29148 and the IREB curriculum, as displayed in Fig. 2.11 and discussed in depth in Section 2.4.3. In the following, for each characteristic, we provide a brief definition and discuss which types of defects cannot and which can be found automatically. We base the analysis upon the results to RQ 4.3 and the discussion in the previous section.

For those defects that cannot be automatically detected, we provide five reasons that make it hard or even impossible to develop an automatic requirements smell detection (see Section 5.2.2):[2]

R_1: Requires more explicit or precise quality definition.

R_2: Requires semantic understanding of text.

R_3: Requires domain knowledge.

R_4: Requires knowledge of the system's scope (e.g. goal).

R_5: Requires knowledge of the project's process and progress.

For those defects that can be automatically detected, automatic requirements smell detection requires a quality factor to be broken down to or approximated at the syntactic level. Therefore, as discussed in Section 4.3.2.2, we have to break down the problem (and also semantic aspects) to one or multiple of the three levels:

S_1: Lexical smells

S_2: Grammatical smells

S_3: Structural smells.

We start analyzing the characteristics on the RE artifact level (i.e. for a set of requirements), before looking at characteristics for individual requirements, as proposed by IREB and IEEE 29148.

5.3.1. Characteristics for Sets of Requirements

In the following, we discuss the characteristics at the RE artifact level (i.e. for a set of requirements) as proposed by IREB and IEEE 29148.

Consistency. A consistent RE artifact is free of redundancy and contradictions.
What we cannot detect: In general, consistency requires understanding the semantics of a text, including the semantics in context, e.g. pragmatics [CW14], or discourse analytics [JM14]. Unfortunately, existing technologies that could provide such information do not yet provide appropriate accuracy [CW14] (R_2).
What we can detect: However, to some extent, we can detect redundancy through clone detection and information retrieval: As discussed by Juergens et al. [JDF+10], clone detection can find consistent and to a certain extent also inconsistent reuse,

2 We drop the reason of data noise, since this analysis is performed on the definition level, not on the instance, i.e. data, level.

if it stems from copy-and-paste reuse. Furthermore, Falessi et al. [FCC13] present a systematic analysis that shows the potential to detect equivalent requirements with approaches based on information retrieval. Some approaches leveraging shallow semantic parsing [RMDP16] also show promising results in first examples. These approaches work on the lexical (S_1) or structural level (S_3).

Completeness. A complete RE artifact includes all relevant stakeholders, goals, and requirements, as well as all information necessary for the RE artifact stakeholders[3].
What we cannot detect: Completeness cannot generally be detected for three reasons: First, it is a matter of the system scope which stakeholders, goals etc. are in scope of the project (R_4). Second, if we knew the system scope, we would have to understand the semantics of the text to know whether the description is complete (R_2). Third, we would need a more precise definition of completeness for different types of artifact stakeholders (R_1, cf. [EVF16, EVFM16]).
What we can detect: However, if we can break down the completeness to a structural completeness, e.g. if a company sets up specific templates, one can detect whether all sections contain content. In addition, heuristics can check whether certain expected terms in this section are used or not (S_1, S_3).

Affordability. In an affordable RE artifact, the team members can create a system fulfilling all requirements within life cycle constraints (e.g. time and budget).
What we cannot detect: Affordability requires unambiguity and completeness, but is more than this. For automatic detection of violations to affordability, we would need an automatic approach to estimate the time required to create a system (such as an automatic version of use case point methods [ADSJ01]) including the project and system constraints and context. By judging from the current state of the even simpler problem of use case points, we argue that, currently, such an automatic detection is still far from reality (R_3, R_4).
What we can detect: However, an ambiguous or incomplete requirement inherently cannot be estimated either, which is the basis for affordability. Therefore, existing automatic approaches to improve unambiguity and incompleteness, also serve to improve affordability (see *Ambiguity* and *Incompleteness* in *Characteristics for Individual Requirements*, S_1, S_2).

Boundededness. In a bounded RE artifact, all requirements are within an identified scope, and within user goals and needs.
What we cannot detect: Boundedness refers to an externally identified scope (R_4). Therefore, there is no generic approach to detect unbounded requirements without adding external information.
What we can detect: Yet, some terms such as *including* or *etc.*, can generally hint for some types of scope creeps, i.e. imprecise definitions of the scope (S_1).

Clear Structure. This IREB criterion is not properly defined by IREB. We assume it contains two factors: In a clearly structured RE artifact, the artifact follows a common structure, and this structure can be understood efficiently and effectively by the readers.
What we cannot detect: We cannot automatically detect whether a structure can be understood efficiently and effectively. This requires structured, empirical analyses, as we describe in RQ 2 (R_1).

3 Please note that, according to our quality paradigm, an absolutely complete RE artifact is not necessarily advisable. It might not even be possible, as discussed by Glinz [Gli16].

What we can detect: However, if a structure is defined, we can automatically detect whether an artifact follows this structure (S_3).

Modifiability. A modifiable RE artifact allows that RE artifact changes can be performed quickly and error-free.
What we cannot detect: Modifiability, is an activity-based property and as such, its definition depends on the activity, i.e. the modification. In Publication C, we analyzed executed changes to clarify this quality characteristic. In our study, we found locally dispersed information, UI details and improper references to be particularly difficult to maintain. Another large set of changes was related to the taxonomy. We cannot give a perfect accuracy for detecting whether two parts of the document are related. However, information retrieval approaches (cf. Falessi et al. [FCC13]) can help us in this task. Furthermore, we cannot foresee future taxonomy changes and therefore it is difficult to judge whether an existing taxonomy will change in future (R_4).
What we can detect: To some extent we can detect locally dispersed information (see *consistent*), as well as UI details and improper references. However, all of these approaches are limited to syntactic aspects (S_1, S_3).

Traceability. In a traceable RE artifact, each requirement correctly defines its links to the origin, implementation and related requirements.
What we cannot detect: Whether two items should be linked, ultimately depends on the domain of the system (R_3) and, therefore, cannot be perfectly predicted in general. In addition, understanding the correctness of existing links requires also semantic understanding of texts (R_2).
What we can detect: However, if this traceability is part of the requirements structure, and every requirement must have certain types of traces, we can structurally detect whether these traces exist (S_3). To analyze whether links are missing, or whether the existing links are correct, various heuristics exist, mostly drawing on information retrieval methods (S_1). The current state of traceability is discussed, i.a. by De Lucia et al. [DFOT07], or more recently by Eder [Ede16].

Unambiguity. Ambiguity is the degree to which a document can be understood in multiple ways. We discuss this in *Characteristics for Individual Requirements*.

5.3.2. Characteristics for Individual Requirements

In the following, we discuss the characteristics on level of individual requirements as proposed by IREB and IEEE 29148.

Unambiguity and understandability. Ambiguity is the degree to which a document can be understood in multiple ways. We will discuss this factor on the individual requirements level. Berry et al. [BKK03] differentiate between lexical, syntactic, semantic and pragmatic ambiguity.
What we cannot detect: In contrast to lexical and syntactic ambiguity, semantic and pragmatic ambiguity refer to the meaning of the word, or even the meaning of the word in context. For both of these types of ambiguity, we are constrained by the current state in NLP (R_2).
What we can detect: Yet, for the other two levels, various approaches exist. For lexical ambiguity, such as synonyms [BKK03], we can make use of existing thesauri (S_1). For syntactic ambiguity (which Berry defines as *a sentence having more than one parse* [BKK03, p.10]), NLP can detect if multiple parses of a sentence are (grammatically, not semantically) possible (S_2).

Necessity. An individual requirement is necessary, if the removal leads to a deficiency, the requirement is not obsolete, and the planned expiration is clearly identified [ISO11b, p.11].
We cannot detect: To detect whether something is necessary is a question of the system scope and is therefore, in general, impossible to automate without additional input (R_4).
We can detect: However, it is possible to detect whether a requirement structurally contains information on a planned expiration (S_3). In addition, existing approaches (e.g. AQUSA [LDBvdW15]) aim at detecting additional, unnecessary information, such as fill words (S_1).

Completeness and verifiability. A complete individual requirement contains all information necessary for this requirement to be used in the consecutive process. Therefore, verifiability is one aspect of completeness of individual requirements. We detail this aspect in our related publications [EVF16, EVFM16].
What we cannot detect: As we detail [EVF16, EVFM16], completeness is very subjective to the usage context. Therefore, which information makes an RE artifact complete must be more precisely specified (R_1) and also depends on the context (R_3).
What we can detect: However, certain phrases and grammatical structures are inherently incomplete. Many of our smells address this aspect and can therefore be considered improving completeness or verifiability, such as the unverifiable terms smell or the superlatives smell (S_1, S_2).

Freedom from implementation. An implementation free specification is defined on the right level of abstraction. However, since RE is often an iterative process between the problem and the solution domain (according to, e.g. the twin-peaks-model [Nus01]), the *right* level of abstraction varies.
What we cannot detect: What is the right level of abstraction, what is problem and what is solution domain, cannot be generally detected since it depends on the context (R_3) and on the project scope (R_4). In addition, sometimes, e.g. for system constraints, we constrain the implementation. On this level, we intentionally dive down into the solution domain. Therefore, whether this level of abstraction is intentional or not, depends on the context (R_3).
What we can detect: However, if we know the intended level of abstraction, e.g. in use cases, or through the definition in company templates or guidelines, we can often identify certain violations. Common examples for this are references to the user interface, or also references to variables or signals (S_1).

Singularity. According to ISO 29148, an individual requirements statement is singular if it contains no more than one requirement [ISO11b, p.11].
What we cannot detect: It is still unclear what an individual requirements is (unless it is structurally defined) and consequently, how to separate individual requirements in general (R_1).
What we can detect: As proposed in the standard, we can detect conjunctions, nominalizations or other grammatical smells for violations of singularity (see e.g. Koerner et al. [KB09]).

Agreement. Although this characteristic remains undefined in the IREB glossary [Gli14], we assume that agreed requirements are accepted by all stakeholders.
What we cannot detect: There can be known and unknown disagreements. Unknown disagreements are part of the mental model, and therefore usually not manifested in the artifact, but a matter of the context (R_3).

What we can detect: However, the known disagreements are often denoted in the artifact in the form of open issue markers, such as todos, tbd's, etc. These can be detected on the lexical level (S_1).

Feasibility. See *affordable* in *Characteristics for Sets of Requirements.*

Consistency and traceability. See *Characteristics for Sets of Requirements.*

5.4. Relation of Findings to Project Success

As we discuss in Section 2.3.2, the impact of RE artifact defects onto projects are indirect and nondeterministic. This is due to RE's nature as a social process, which strongly depends on the individual's background and context. Consequently, the further away an effect is from the RE artifact defect, the harder it is to draw causal conclusions. Therefore, in our work, we focused on immediate effects, such as understanding or maintenance of RE artifacts (see RQ 2 in Section 4.2). In the following, we want to go beyond these limitations, and discuss whether the number of findings produced by automatic requirements smell detection provides indication for the project success and whether the introduction of requirements smell detection improves the probability of project success.

Smells as predictors for project success. To use smells as a predictor for project success requires that projects with fewer findings (or lower findings density) are more successful (to an operationalized definition of *success*) than projects with a larger number of findings (or higher findings density). If an RE artifact has few or no findings, this only reflects the automatically detectable quality factors. Consequently, the RE artifact could have a number of not-automatically detectable defects (see previous section). Since in RE any single defect can have tremendous consequences, any defect that the requirements smell detection missed could potentially decide about project success or failure (consider e.g. a forgotten stakeholder). The same holds for RE artifacts with many findings. One could imagine projects in which none of the findings turns out to be a serious RE defect, or in which all RE defects are compensated in the project. Therefore, any relationship between smells and any variable for project success can only be a fuzzy one, depending on the project, people, smells, and other context factors. Therefore, the potential of findings as an predictor are inherently limited.

However, there are two aspects why findings could be used as a predictor: First, if findings, such as instances of *passive voice*, have a certain probability to lead to a defect, as our studies indicate, this means that (in cases in which the probability is independent, i.e. not depending on the context etc.) the more smells we have, the higher is the probability that any of these leads to a defect. In addition, the more findings we have, the more do impacts of these defects culminate. For example, in RQ 4.4. in Publication G, a practitioner reports that removing smell findings from user stories could reduce the time spent on effort estimations. Therefore, the projects with more findings culminate more extra time spent on these activities, which increases the risk of a failing project. The second reason why findings could be used as a predictor is relating the visible findings with the invisible characteristics. It is reasonable to believe that the number of findings, which are detected by automatic requirements smell detection, correlates with quality of the general engineering of the project team, such as precision, structuredness, and professionalism. In this case, smells are just the visible predictor of a larger problem, which is a predictor

for project success. If any of these is the case, findings could be used as a predictor for project success.

Impact of smell detection. In addition to the passive application of smells as a predictor, we also have to consider the impact of introducing a technique, such as requirements smell detection, to a project. One could argue that the introduction of requirements smell detection has multiple effects on the team: First, the number of findings would decrease. Following the argumentation above, this should reduce the risk that any of the detectable defects impairs with the project. Second, the cumulating effects, such as the impact on readability, are decreased, which directly reduces efforts and costs. However, the introduction of a new quality control mechanism might also help the team by fostering discussions on quality. This should improve the quality awareness and, consequently, also the quality in general. Lastly, findings make people look again. When working with practitioners and requirements smell detection tools, we repeatedly found that, only after the tool indicated a finding, the practitioner started consciously rethinking the RE artifacts. This trigger then sometimes made the pracitioner discuss an issue that was not related to the issue that the requirements smell detection discovered. In this way, introducing requirements smell detection could have multiple side-effects to improve quality in general, and thereby increase the chances for project success.

However, in our opinion, these aspects should be empirically analyzed in order to derive conclusions. Future work should develop studies with a broad set of projects, as well as longitudinal and qualitative studies to understand whether these relationships in fact hold.

5.5. Summary

We argue that an activity-based viewpoint allows to create a shared understanding as well as a common language for RE artifact quality. Furthermore, we argue that ABRE-QMs enable to precisely reason about which quality factors exist and to precisely evaluate the consequences of a quality factor. We also argue that ABRE-QMs enable to more adequately define RE artifact quality and provide a first notion of a complete quality model.

However, open questions exist towards the existence of one common quality definition, the right granularity of model elements, human compensation, transitive, feedback and non-linear impacts, and, more fundamentally, the extent to which we (are able to) know about the impact of quality factors in RE artifacts.

Regarding automatic requirements smell detection, automatic approaches have their advantages, namely effort, speed, and consistency. However, they assume the existence of an explicit quality definition. They furthermore struggle compensating for noise and the limitations of the state-of-the-art NLP methods. In addition, automatic approaches lack the knowledge about the project scope (what do the stakeholders want?), as well as the domain knowledge that is not captured in RE artifacts.

Automatic requirements smell detection can detect violations of various quality criteria. However, for all of the criteria, there are also aspects that are not covered. This, again, suggests the need for combining automatic with manual approaches, as we propose in RQ 3. In addition, the question whether the number of findings is a accurate predictor of project success is still an open question.

CHAPTER 6

Conclusions and Outlook

Looking back at the problem statement, we summarize our results and provide an outlook to future work.

6.1. Conclusions

RE artifacts play a central role in software development. Their quality can be crucial for the project success. But, as discussed in Chapter 1, we currently have a limited understanding of what high quality RE artifacts are and need more efficient methods to control RE artifact quality in practice. This thesis approaches these two problems in two directions: First, we contribute a novel model to define RE artifact quality, and second, we present a more efficient method to execute QC for RE artifacts. In the following, we come back to the research questions and summarize our results.

6.1.1. Definition of RE Artifact Quality

As we argue in Chapter 1, existing definitions for RE artifact quality lack precise reasoning and adequacy for various contexts. In addition, whether existing sets of quality characteristics are complete is an open question.

We refined this problem into two research questions. To precisely define RE artifact quality in RQ 1, we contribute a meta model for RE artifact quality from a quality-in-use perspective. To create valid quality models for RQ 2, we contribute a validation framework and three applications thereof.

RQ 1: How can we precisely define quality for RE artifacts? This thesis contributes Activity-based RE Quality Models (ABRE-QM), an approach to define RE artifact quality, based on the understanding that high quality RE artifacts are those that are efficient and effective to use.

Based on this notion of quality-in-use, ABRE-QMs enable to precisely reason about quality factors and their impact on a stakeholder's activities. In addition, the quality-in-use perspective enables to model quality for a given context. Lastly, this

thesis provides a first understanding of completeness for an RE artifact quality model.

In summary, this thesis answers RQ 1 through a meta model that defines RE artifact quality from a quality-in-use perspective.

RQ 2: How can we create valid quality models? We argue that the main advantage of the ABRE-QM is its precise reasoning. In RQ 2, we contribute a framework for creating valid ABRE-QMs and show its use in an interview, a case study, and an experiment.

We also contribute a brief discussion of advantages and disadvantages according to our experience during these studies. We argue that interviews are best suited for validating a large set of known quality factors in a given context. For example, we contribute an interview study to validate an existing use case guideline. In contrast, case studies enable to study more unknown fields. In our case, we contributed a case study to better understand quality factors for maintenance of RE artifacts. Lastly, experiments can study individual quality factors in depth. However, due to their artificial setting and expensive setup, we argue to use this method for subtle, unclear quality factors. For example, we contributed an experiment that analyzed the effects of passive voice in RE artifacts.

In summary, this thesis answers RQ 2 through providing a framework for interviews, case studies and experiments. In addition, we discuss advantages, disadvantages and applications based on a set of contributing studies.

6.1.2. Efficient Methods for RE Artifact Quality Control

As a second problem, we argued that quality control of RE artifact quality struggles in practices, even when agreed upon quality factors. To more efficiently detect quality factors for RQ 3, we contribute requirements smells and automatic requirements smell detection. To understand benefits and limitations of requirements smell detection in RQ 4, we study the accuracy, the applicability, the benefits and the limitations of such an approach.

RQ 3: How can we efficiently ensure quality factors? We contribute a case study on RE artifact quality, indicating that besides a precise quality definition, the core problems lie in efficiency, speed and sustainability of the QC process. To address the issues, we furthermore contribute a definition of requirements smells as automatically detectable quality factors. In order to make QC more efficient we propose automatic requirements smell detection, for which we also provide a prototype as a technical validation. In addition, we contribute a smell taxonomy and a method that combines manual and automatic QA.

In summary, this thesis answers RQ 3 through combining efficient automatic requirements smell detection with manual detection of quality factors.

RQ 4: What are the benefits and limitations of requirements smell detection? To understand the precision, applicability, benefits and limitations, we contribute the results of a multi-case study with 12 industrial projects from three companies and an academic case, where we analyzed requirements of 51 different teams. In total, we analyzed requirements of the size of more than 265,000 words. To determine, how reliably the detected smells indicate quality problems, we measure the precision of our analysis. The requirements smell detection precision varies strongly between 0.26 and 0.96, averaging around 0.6. Furthermore, to understand the number of

undetected findings, we also analyzed the recall, leading to an average recall of 0.82. Practitioners report that several of the findings not only indicated quality problems, but were relevant for practitioners. We furthermore discuss potential applications with practitioners.

Based on analyzing review reports, we identified the following needs as main areas where automation is limited: stakeholder or domain knowledge, deep natural language understanding, knowledge of system scope or goal, knowledge of process information, and vaguely or subjectively defined criteria. Lastly, in an analysis of a large, industrial RE artifact guideline, we estimate that 52% of the criteria can be checked either perfectly or with a good heuristic. For detection of violations, most criteria require just simple heuristics. The main reason why criteria cannot be automatically detected are imprecise or unclear guideline definitions.

In summary, this thesis answers RQ 4 through a detailed analysis in multiple case studies and quality definitions. Based on a discussion of benefits and limitations, the results indicate that a combination of automatic and manual QA techniques is required, in which a requirements smell detection helps to solve simpler issues and leaves in-depth QA to reviews.

6.1.3. Summary of Contributions

We argue that this thesis provides practitioners with: a) a language to precisely and adequately define RE artifact quality, b) an efficient approach to detect quality factors, as well as c) a method to combine automatic with manual reviews.

Furthermore, this thesis provides researchers with a) an actionable notion of RE artifact quality, b) a framework to validate quality factors, and c) first evidence for quality factors for maintenance of RE artifacts and the negative impact of passive voice requirements onto understanding.

6.2. Outlook

This thesis is a first step into the direction of RE artifact quality and automatic requirements smell detection. In the following, we want to outline the various directions for future work and the questions that this thesis raises.

We see four directions for future work: First, extensions of details within our proposed approaches, namely extensions of the ABRE-QM meta model and extensions of requirements smells. Second, extensions of our empirical studies and our applications of the activity-based quality paradigm, which may lead to a common body of knowledge for RE artifact quality. Finally, one could extend the concept beyond RE artifact quality to RE quality in general.

6.2.1. Extending the ABRE-QM Meta Model

The first path of changes are extensions to the ABRE-QM meta model, which would allow usage of ABRE-QMs in activities further than requirements smell detection.

Combining ABRE-QMs with existing artifact and process models. In the presented studies, we created artifact and process models in an ad-hoc manner as far as required for defining the ABRE-QM.

Another option would be to combine ABRE-QMs with existing and comprehensive process models (e.g. as known from the V-Modell XT, see [BR05]), which would

enable to relate the quality models to the process. Similarly, one could reuse existing artifact models for ABRE-QMs, which would allow to resort to existing content items as entities (as described e.g. in [MPKB10, MF11, MWL+12, PEM13, MP14]).

These combinations enable to integrate process and artifact models with quality models. This would not only enable to reuse the models and therefore save time during the creation of the ABRE-QM, a fine grained modelling of, for example, activities enables to reason as precisely in the ABRE-QM.

Extending the ABRE-QM meta model with criticality and probability. To keep the model simple and also as widely applicable as possible, the ABRE-QM meta model provides only rough guidance for defining positive and negative impacts.

However, whether a quality factor makes the reading activity less efficient or less effective, can potentially have vast differences on the overall project costs. And even there, known issues in effectiveness (i.e. misunderstandings) might be less risky than implicit, unknown misunderstandings. To address this, future work should extend impacts, as well as activities, with cost or risk models to express such aspects.

The model could be extended to serve such requirements through additional attributes of the impact element. This would allow for structured analysis through the quality model, e.g. in cost or risk models (see e.g. [Dei09, WLH+12, HJE+14, Hau16]). In consequence, practitioners could use such a model to, i.a. precisely argue which quality factors to focus on. It would finally enable to define the costs and risks of bad quality. However, it is still a very open question whether we can elicit valid costs and probabilities given the complex domain of RE, as we also discuss in the limitations to ABRE-QM in Chapter 5.1.2.

6.2.2. Extending Research on Requirements Smells

In the second part of this thesis, we propose and discuss requirements smells. These requirements smells could be further extended in terms of requirements smell detection, but also further requirements smell definitions.

Improving precision of requirements smell detection. For accuracy, other approaches by Krisch and Houdek [Kri13, KH15] have shown the potential to increase precision of automatic detection mechanisms. Future work could develop further heuristics to understand to which extent the precision can be improved, based on the false positives that we detected in Chapters 4.4. In this field, we see strong applicability for artificial intelligence and machine learning, similar to the approaches by Yang et al. [YRG+11, YRG+12], yet based on a detailed analysis of word, sentence and document context.

In addition, we understand the smells that we presented in Publication G as first steps towards a catalogue. The taxonomy described in Chapters 4.3 provides a framework for this endeavor. Also, natural language processing is continuously advancing [CW14]. So far, approaches towards understanding semantics of unstructured natural language texts still lack precision. However, in future, we should discuss which smells exist that can detect quality problems through deeper understanding of the semantics of a text. The same holds for understanding discourse semantics, i.e. a stateful understanding of sentences in their linear context [CW14]. This would allow for more precise analysis, e.g. of the smell vague pronouns.

Together, all of these will change what can and what cannot be detected through requirements smells, as well as our understanding thereof.

Smells for other natural language artifacts. Besides RE, the proposed requirements smell detection techniques could be also applied to further similar artifacts such as legal texts, or various types of specifications. For example, in the area of security (requirements) engineering, engineers create vulnerability descriptions. Just as with RE artifacts, these descriptions are used as an input in the software engineering process and therefore, their quality has similar impacts on the process.

In a first step into this direction, we analyzed missing content in vulnerability description in a work with Allodi et al. [ABBF17] (see Fig. 6.1). Future work should analyze in how far our results also hold in this domain.

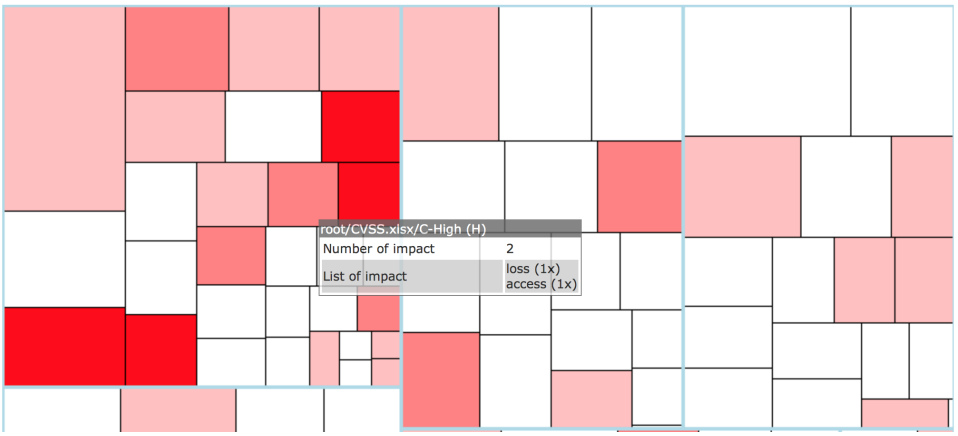


Figure 6.1.: This figure shows a tree map resulting from the analysis of vulnerability descriptions. Each box represents one Common Vulnerability Scoring System (CVSS) description. Red boxes contain one or more keywords describing the impact. White boxes lack impact descriptions. See [ABBF17] for a detailed explanation.

Smells for semi-formal or formal RE artifacts. In this work, all smells are generic and can be applied to various types of RE artifacts, such as sentence requirements, use cases or user stories. With the exception of user stories in RQ 4, we do not analyze the different types specifically.

However, the more structured an RE artifact is documented, the more we can analyze automatically. This holds not only for user stories, for which we analyzed whether or not a user story contained a rationale in RQ 4, but also for requirements that are created through sentence patterns (e.g. [EVF16]). In a first step in this direction, we define and detect specific requirements smells for use cases together with Munich Re. Further requirements smells could be defined for other representations than natural language, such as UML diagrams.

6.2.3. Extending Our Studies

In this thesis, we not only propose new approaches, but also empirically evaluate these approaches. To increase internal and external validity, and to extend their scope, we suggest to replicate and extend our studies.

Conducting further studies on quality factors. In this work, we evaluate quality factors for maintenance, passive voice, and a first set of requirements smells. However, as we discuss in the publication in depth, these studies come with their threats

to validity. Therefore, future work should analyze these quality factors in other contexts with other methods to further increase the validity of the results.

Conducting longitudinal studies for RE QC approaches. In our contributions, we analyze the quality of RE artifacts at a given point in time. Ideally, we assume that if a team receives constant feedback from an approach such as requirements smell detection, we should see a trend in quality of the artifacts.

To analyze this aspect in depth, we need longitudinal studies with steady academic supervision to understand the consequences of RE artifact QC, of requirements smells, and of requirements smell detection. These longitudinal studies come with additional technical requirements towards requirements smell detection, e.g. the tracking of findings over time (as is known, e.g. from source code [HHS14, Ste16]).

Conducting studies on second and third-order effects of RE artifact quality. In our contributions, we focused mainly on direct effects of RE artifact quality. However, as explained in Chapter 2.3.2 and discussed in Chapter 5.1.2, impacts of RE artifact quality can produce effects also on further artifacts, which can have serious consequences on the project. For example, a missing performance requirement in an RE artifact does not have negative consequences on understanding, developing and testing directly, but only on the produced output (i.e. test and code). The negative consequences only turn out as a so-called second order effect [HSW05]. Along this line of reasoning, even more difficult are systemic or third-order effects. For example, a badly maintainable RE artifact might impact the reputation of the company and thus have long term consequences out of the scope of an individual project. Our work on cause and effect relation for RE variables [MMFV14] is a first step in that direction (see Fig. 6.2).

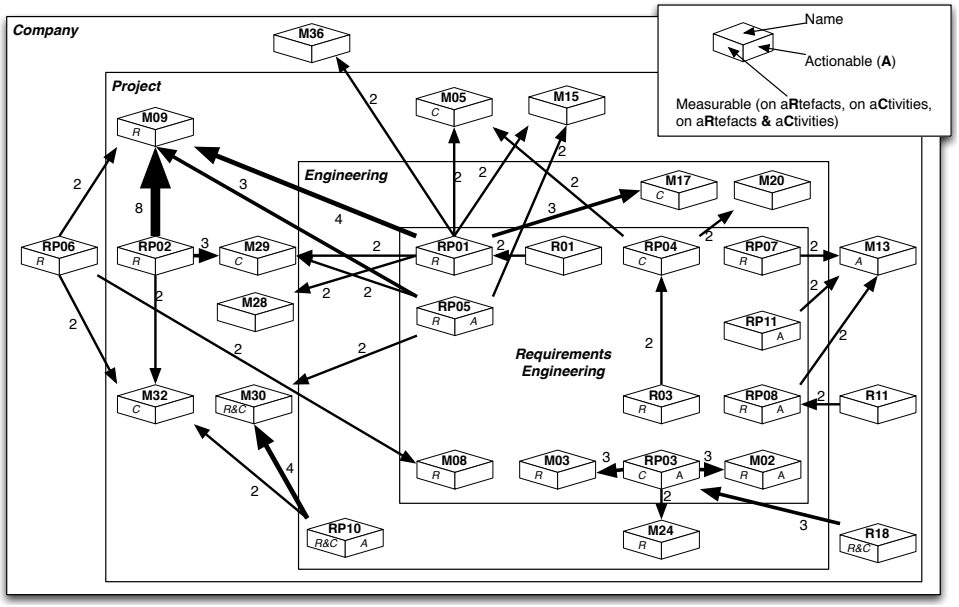


Figure 6.2.: This figure shows an excerpt of the variable relations. Each item is one variable in one of the four dimensions (RE, engineering, project, company). Each arrow means that n>1 participants of the survey argued that one of variable is a consequence of the others. On the side of the variables we denoted whether or not these variables are measurable. Figure taken from [MMFV14].

6.2.4. Extending Applications of Activity-based Quality

We have applied the proposed meta model for different purposes. The meta model proved beneficial in several contexts that are discussed in the following.

1. Activity-based understanding of RE artifact quality: First and foremost, the concept of activity-based RE artifact quality enables to systematically understand the so-far fuzzy concept of RE artifact quality. For this, ABRE-QMs help to structure the argumentation why and in which context a quality factor is problematic, as we show in Publication A. In addition, the activities can be used to systematically analyze quality factors in their consequences, as we show in Publications D and C.
2. Activity-based RE guidelines: Many companies nowadays have guidelines to help employees improve their requirements and to create a baseline for quality. However, as stated before, these guidelines are often incomplete and imprecise.

   We argue that guidelines that are defined in an activity-based manner could help to make these guidelines more complete and precise. In Publication A, practitioners reported that a translated guideline helps to both discuss validity of the existing rules and to create more complete guidelines.
3. Activity-based requirements elicitation and documentation: Requirements templates are blueprints that determine the syntactic structure of a single requirement. One reported advantage of requirements templates is that they facilitate more *complete* specifications of requirements. However, what complete actually means depends on how the requirement is used: The information that needs to be provided in a requirement is determined by the activities that are performed based on the requirement. For example, when performance tests will be conducted for the system, the requirement templates must enforce to augment requirements with detailed information about the desired reaction times and assumed conditions [EVF16]. In a recent paper [EVFM16], we used activity-based models to tailor requirements templates in a way that the information they demand for a requirement fit the actual usage in a specific development context. The result is a set of requirements templates that are more specific and expressive than the general templates that are proposed to fit every situation.
4. Activity-based quality control: The presented paradigm also has strong implications on quality control. This is both for constructive aspects, such as tailoring guidelines to requirements use, but also analytical approaches, such as requirements smells (see Publications F and G), where ABRE-QMs could enable us to decide which quality aspects should hold in a certain context. In addition, techniques, such as manual reviews, can benefit from an activity-based understanding, as we point out in Publication H and as is also discovered through perspective-based reading, by, inter alia, Basili, Shull and He [BGL+96, SRB00, HC06].
5. Activity-based decision making: We used the proposed meta model to develop cost models to enable an informed decision making process: In a recent study [VFJ16], we used an instance of the meta model to characterize the cost and benefits of refactoring functional parts that reoccur in several functions of a system specification. The decision whether a refactoring pays off heavily depends on the context in which the respective system specification is used. Therefore, we identified activities that are performed with the system specification, and we identified cost factors that affect these activities in the original and the refactored version. Cost factors are a specific form of quality factors as apparent in the meta model. As a result, the decision whether to refactor a specification or leave it as it is can be assessed with respect to the

usage context. For one context, the refactoring pays off because functions must be tested frequently and recurring parts in the functions makes testing more expensive. For another context, the refactoring does not pay off because the responsibility for implementing the refactored parts is in a different department and, thus, we need to consider costs for knowledge transfer. A similar approach is taken by Hauptmann et al. [HJE+14, Hau16] to decide when and if test step automation pays off.

We assume that these are only the first of many applications. We argue that in future work, activity-based viewpoints can show benefits in many further applications, in particular when requirements engineers must decide in a trade-off.

6.2.5. Extending RE Artifact Quality by Building a Common Body of Knowledge

In this work, we provide a syntax for defining and reasoning about RE artifact quality. In addition, we provide first insights into RE artifact maintenance and the impact of passive voice in RE artifacts.

But these are just small pieces of the puzzle. Future work should collect existing evidence in the RE community and translate the existing knowledge into an ABRE-QM. This could require extensions to the ABRE-QM impacts in the meta model as discussed in this chapter. However, based on such a theory, we could identify blind spots in this domain and extend the existing knowledge. It is an open question, however, to which extent such a generic quality definition exists.

If the research continues along this theory, the community can, together, create a generic ABRE-QM, which resembles the existing knowledge on RE artifact quality. The precision of such a theory would allow researchers to systematically discuss results in the field and its focus on activities would enable practitioners to understand and weigh consequences of bad quality in a structured manner. In the long run, this paradigm could even be extended beyond artifacts to create a general RE quality theory. For this goal, we must take five steps:

1. We must understand users and their usage of RE artifacts. Activity-based RE artifact quality is based on the idea that good quality is defined by RE artifact users and their usage. However, these aspects are not yet precisely understood in RE. In particular, we found that in practice RE artifact usages vary, depending on the goal of RE in the respective context (see goals of RE in Section 2.2).

2. We need detailed artifact and language models for different RE approaches. We must additionally validate the artifacts and entities, as described in the model: This involves questioning which artifacts exist, which artifacts are used, and how they relate to each other. One could extend this step into a comprehensive artifact model, which would allow us to model quality factors and their relationships in depth (as described i.a. in [MPKB10, MF11, MWL+12, PEM13, MP14]). In addition to artifact models, we must also extend the language model (see Chapter 4.3.2.2) with improvements in NLP.

3. We must understand existing quality factors and their impacts. After we understood how stakeholders use the RE artifacts, and with detailed artifact and language models, we can then conduct systematic literature reviews to understand all existing work in this area. We can then frame their studies within an ABRE-QM,

which will lead to a first draft of the existing quality factors and their impacts. Further literature reviews in the area of psycholinguistics (see Rayner et al. [RPACJ11] for an introduction) and related fields should supplement this research.

4. We must understand context factors and how they confound the impacts of quality models. In the ABRE-QM meta model, we assume that impacts take place depending on the context (i.e. human, tool, or process context, see Chapter 4.1. In the remaining work, to a large extent, we left the context out of scope. Future work must understand to which extent these context factors are tangible enough to be measured, understood, and taken into account. Alternatively, we could model impacts in a more probabilistic manner and thus abstract from the detailed contexts. This discussion will then also answer the next question that is raised by this thesis, which is: To which extent is quality project-, company-, or process dependent.

There are two options towards this problem: On the one hand, stakeholder roles, activities, and artifacts enable to tailor the quality model. In this case, tailoring means just taking the relevant quality factors and leaving out the irrelevant. Tailoring would not impact the general body of knowledge. On the other hand, if the impacts and context factors themselves change between projects, and thus, if we need to manually adapt the model, this would render the application cumbersome. However, based on the results from Chapter 4.2, we would estimate that many aspects, especially those based on linguistic understanding of texts, are to a large extent generic and thus hold for most contexts.

5. We must maintain this body of knowledge. Over time, the existing body of knowledge will change. This includes removal of unused artifacts, addition of new artifacts and quality factors, and new activities and stakeholders, in case processes change.

6.2.6. Extending ABRE-QM Towards Activity-based RE Quality

Lastly, the topic of this thesis is RE artifact quality. Therefore, we model RE artifact quality instead of RE quality in general (see Fig 6.3). In order to be able to draw more generic conclusions about RE quality, we should leverage the activity-based quality paradigm from artifacts to all activities that are conducted based on RE results. As a first step into this direction, we must try to understand which of the context factors as described in the ABRE-QM influence which activities directly (i.e. independent from any artifacts). Examples for this could be team factors, background knowledge, or also tool quality factors, such as important tool features.

This would enable us to also reflect quality in both document centric, but also rather agile projects, which do not depend as thoroughly on RE artifacts for communications. We think that the resulting theory could help understanding the complex reality of RE quality in more depth.

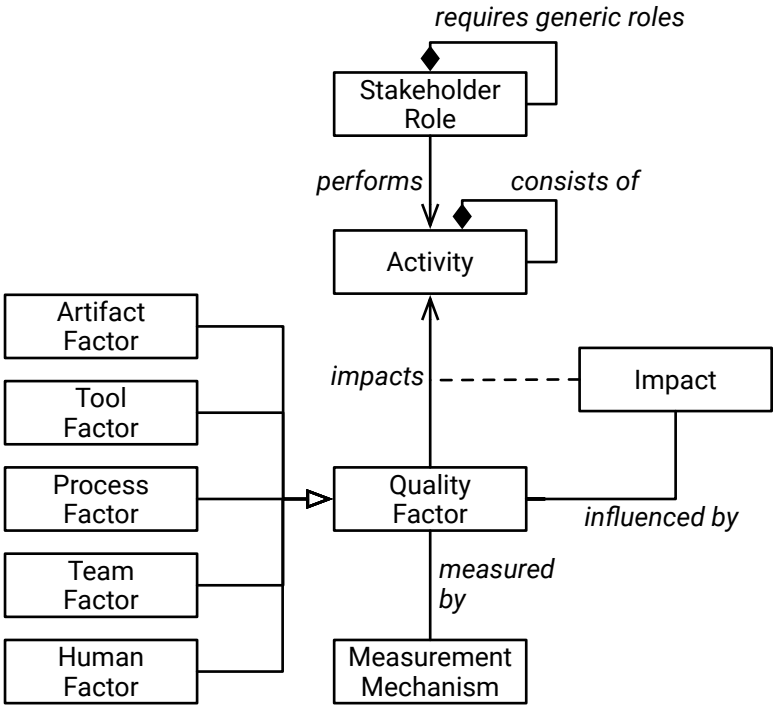


Figure 6.3.: This figure shows an extension of the meta model for RE quality in general. In this case both artifact and context factors have an impact on the activity.

Bibliography

[ABBF17] Luca Allodi, Sebastian Banescu, Kristian Beckers, and Henning Femmer. Identifying relevant information cues for vulnerability assessment using CVSS. In *Submitted to the International Symposium on Empirical Software Engineering and Measurement*, ESEM. ACM, 2017.

[ADSJ01] Bente Anda, Hege Dreiem, Dag IK Sjøberg, and Magne Jørgensen. Estimating software development effort based on use cases—experiences from industry. In *International Conference on the Unified Modeling Language*, UML, pages 487–502. Springer, 2001.

[AriBC] Aristotle. *Categories*. Edghill, E. M., The University of Adelaide, 2013 edition, 350 B.C.

[AS02] Bente Anda and Dag I. K. Sjøberg. Towards an inspection technique for use case models. In *International Conference on Software Engineering and Knowledge Engineering*, SEKE, pages 127–134. ACM, 2002.

[BBG+06] Daniel M. Berry, Antonio Bucchiarone, Stefania Gnesi, Giuseppe Lami, and Gianluca Trentanni. A new quality model for natural language requirements specifications. In *Requirements Engineering: Foundation for Software Quality*, REFSQ, pages 1–12. Springer, 2006.

[BFE+15] Mohammad R. Basirati, Henning Femmer, Sebastian Eder, Martin Fritzsche, and Alexander Widera. Understanding changes in use cases: A case study. In *International Requirements Engineering Conference*, RE, pages 352–361. IEEE, 2015.

[BGL+96] Victor Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sørumgard, and Marvin V. Zelkowitz. The empirical investigation of perspective-based reading. *Empirical Software Engineering Journal*, 1:133–164, 1996.

[BHH+03] K. Buhr, N. Heumesser, F. Houdek, H. Omasreiter, F. Rothermel, R. Tavakoli, and Zink T. DaimlerChrysler demonstrator: System specification. Technical report, EMPRESS Project, 2003.

[BK04] Daniel M. Berry and Erik Kamsties. Ambiguity in requirements specification. In Julio Cesar Sampaio do Prado Leite and Jorge Horacio Doorn, editors, *Perspectives on Software Requirements*, chapter 2, pages 7–44. Springer, 2004.

[BKK03] Daniel M. Berry, Erik Kamsties, and Michael M Krieger. From contract drafting to software specification: Linguistic sources of ambiguity. Technical report, University of Waterloo, 2003.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, 1st edition, 1981.

[Bos07] Boston Scientific. Pacemaker system specification. Technical report, Boston Scientific, 2007.

[BP88] Barry W. Boehm and Philip N. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 14(10):1462–1477, 1988.

[BR05] Manfred Broy and Andreas Rausch. Das neue V-Modell® XT (in German). *Informatik-Spektrum*, 28(3):220–229, 2005.

[Bro06] Manfred Broy. Requirements engineering as a key to holistic software quality. In *Computer and Information Sciences*, ISCIS, pages 24–34. Springer, 2006.

[CD09] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.

[Cho57] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2nd edition, 1957.

[CMM06] CMMI Product Team. CMMI® for development, version 1.2. Technical report, Carnegue Mellon Software Engineering Institute (CMU/-SEI), 2006.

[Coc98] Alistair Cockburn. Basic use case template. Technical report, Humans and Technology, 1998.

[Coh04] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[CW14] Erik Cambria and Bebo White. Jumping NLP curves: A review of natural language processing research. *IEEE Computational Intelligence Magazine*, 9(2):48–57, 2014.

[DAEE08] Joerg Doerr, Sebastian Adam, Michael Eisenbarth, and Michael Ehresmann. Implementing requirements engineering processes: using cooperative self-assessment and improvement. *IEEE Software*, 25(3):71–77, 2008.

[dCG14] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Workshop on Open Infrastructures and Analysis Frameworks for HLT*, OIAF4HLT, pages 1–11, 2014.

[Dei09] Florian Deissenboeck. *Continuous Quality Control of Long-Lived Software Systems*. Dissertation, Technische Universität München, 2009.

[DFOT07] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *Transactions on Software Engineering and Methodology*, 16(4):13:1–13:50, 2007.

[DJLW09] Florian Deissenboeck, Elmar Juergens, Klaus Lochmann, and Stefan Wagner. Software quality models: Purposes, usage scenarios and requirements. In *International Workshop on Software Quality*, WoSQ, pages 9–14. IEEE, 2009.

[DOJ+93] Alan M. Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, Anh Ta, and Mary Theofanos. Identifying and measuring quality in a software requirements specification. In *International Software Metrics Symposium*, METRICS, pages 141–152. IEEE, 1993.

[Dru92] Colin Drury. Activity-based costing. In *Management and Cost Accounting*, pages 273–288. Springer, 1992.

[DWP+07] Florian Deissenboeck, Stefan Wagner, Markus Pizka, Stefan Teuchert, and Jean-Francois Girard. An activity-based quality model for maintainability. In *International Conference on Software Maintenance and Evolution*, ICSM, pages 184–193. IEEE, 2007.

[Ede16] Sebastian Eder. *Exploiting Execution Profiles in Software Maintenance and Test*. Dissertation, Technische Universität München, München, 2016.

[EVF16] Jonas Eckhardt, Andreas Vogelsang, and Henning Femmer. An approach for creating sentence patterns for quality requirements. In *International Workshop on Requirements Patterns*, RePa, pages 1–8. IEEE, 2016.

[EVFM16] Jonas Eckhardt, Andreas Vogelsang, Henning Femmer, and Philipp Mager. Challenging incompleteness of performance requirements by sentence patterns. In *International Requirements Engineering Conference*, RE, pages 1–10. IEEE, 2016.

[FCC13] Davide Falessi, Giovanni Cantone, and Gerardo Canfora. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Transactions on Software Engineering*, 39(1):18–44, 2013.

[FFGL00] Fabrizio Fabrini, Mario Fusani, Stefania Gnesi, and Guiseppe Lami. Quality evaluation of software requirements. In *Software and Internet Quality Week Conference*, pages 1–18, 2000.

[FGLM02] Alessandro Fantechi, Stefania Gnesi, Guiseppe Lami, and Alessandro Maccari. Application of linguistic techniques for use case analysis. *Requirements Engineering Journal*, 8(3):161–170, 2002.

[FHEM16] Henning Femmer, Benedikt Hauptmann, Sebastian Eder, and Dagmar Moser. Quality assurance of requirements artifacts in practice: A case study and a process proposal. In *International Conference on Product-Focused Software Process Improvement*, PROFES, pages 506–516. Springer, 2016.

[FKSJ14] Henning Femmer, Marco Kuhrmann, Joerg Stimmer, and Joerg Junge. Experiences from the design of an artifact model for distributed agile project management. In *International Conference on Global Software Engineering*, ICGSE, pages 1–5. IEEE, 2014.

[FKV14] Henning Femmer, Jan Kučera, and Antonio Vetrò. On the impact of passive voice requirements on domain modelling. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 21:1–21:4. ACM, 2014.

[FMJ+14] Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer. Rapid requirements checks with requirements smells: Two case studies. In *International Workshop on Rapid Continuous Software Engineering*, RCoSE, pages 10–19. ACM, 2014.

[FMM15] Henning Femmer, Jakob Mund, and Daniel Méndez Fernández. It's the activities, stupid! A new perspective on RE quality. In *International Workshop on Requirements Engineering and Testing*, RET, pages 13–19. IEEE, 2015.

[FMWE17] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017.

[FN99] Norman E. Fenton and Martin Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999.

[FUG17] Henning Femmer, Michael Unterkalmsteiner, and Tony Gorschek. Which requirements artifact quality defects are automatically detectable? A case study. In *Fourth International Workshop on Artificial Intelligence for Requirements Engineering*, AIRE, pages 1–7. IEEE, 2017.

[FV17] Henning Femmer and Andreas Vogelsang. Requirements quality is quality in use – a novel viewpoint –. *Submitted to IEEE Software*, 2017.

[Gar84] David A. Garvin. What does "product quality" really mean? *Sloan Management Review*, pages 25–43, 1984.

[GD08] Tony Gorschek and Alan M. Davis. Requirements engineering: In search of the dependent variables. *Information and Software Technology*, 50(1-2):67 – 75, 2008.

[GE09] Eugenie Giesbrecht and Stefan Evert. Is part-of-speech tagging a solved task? An evaluation of POS taggers for the German web as corpus. In *Web as Corpus Workshop*, WAC5, pages 27–35, 2009.

[GF15] Martin Glinz and Samuel A. Fricker. On shared understanding in software engineering: an essay. *Computer Science - Research and Development*, 30(3):363–376, 2015.

[Gla98] Robert L. Glass. Maintenance: Less is not more. *IEEE Software*, 15(4):67–68, 1998.

[Gli14] Martin Glinz. A glossary of requirements engineering terminology. Technical report, International Requirements Engineering Board and University of Zurich, 2014.

[Gli16] Martin Glinz. How much requirements engineering do we need? *Softwaretechnik-Trends*, 36(3):19–21, 2016.

[GW07] Martin Glinz and Roel J Wieringa. Stakeholders in requirements engineering. *IEEE Software*, 24(2):18–20, 2007.

[Hau16] Benedikt Hauptmann. *Reducing System Testing Effort by Focusing on Commonalities in Test Procedures*. PhD thesis, Technische Universität München, 2016.

[HC06] Lulu He and Jeffrey Carver. PBR vs. checklist: A replication in the n-fold inspection context. In *International Symposium on Empirical Software Engineering*, ISESE, pages 95–104. ACM, 2006.

[HEZ15] Tobias Horsmann, Nicolai Erbs, and Torsten Zesch. Fast or accurate? – a comparative evaluation of pos tagging models. In *International Conference of the German Society for Computational Linguistics and Language Technology*, pages 22–30. German Society for Computational Linguistics and Language Technology, 2015.

[HHS14] Lars Heinemann, Benjamin Hummel, and Daniela Steidl. Teamscale: Software quality control in real-time. In *International Conference on Software Engineering*, ICSE, pages 592–595. ACM, 2014.

[HJE+13] Benedikt Hauptmann, Maximilian Junker, Sebastian Eder, Lars Heinemann, Rudolf Vaas, and Peter Braun. Hunting for smells in natural language tests. In *International Conference on Software Engineering*, ICSE, pages 1217–1220. IEEE, 2013.

[HJE+14] Benedikt Hauptmann, Maximilian Junker, Sebastian Eder, Christian Amann, and Rudolf Vaas. An expert-based cost estimation model for system test execution. In *International Conference on Software and System Process*, ICSSP, pages 159–163. Springer, 2014.

[HL01] Hubert F. Hofmann and Franz Lehner. Requirements engineering as a success factor in software projects. *IEEE Software*, 18(4):58–66, 2001.

[HSW05] Lorenz M. Hilty, Eberhard K. Seifert, and Jennifer Wetzel, editors. *Information Systems for Sustainable Development*. Idea Group Publishing, 2005.

[IEE98] IEEE Computer Society. IEEE recommended practice for software requirements specifications. Technical report, IEEE Computer Society, 1998.

[Int15] International Requirements Engineering Board. IREB certified professional for requirements engineering - foundation level - syllabus - version 2.2. Technical report, International Requirements Engineering Board e.V., 2015.

[ISO05] ISO 9000:2005. Quality management systems— fundamentals and vocabulary. Technical report, ISO, 2005.

[ISO10] ISO/IEC/IEEE 24765:2010. Systems and software engineering - Vocabulary. Technical report, ISO/IEC/IEEE, 2010.

[ISO11a] ISO/IEC 25010:2011. Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models. Technical report, ISO/IEC, 2011.

[ISO11b] ISO/IEC/IEEE 29148:2011. Systems and software engineering - life cycle processes - requirements engineering. Technical report, ISO/IEC/IEEE, 2011.

[ISO12] ISO/IEC 17024:2012 . Conformity assessment — general requirements for bodies operating certification of persons. Technical report, ISO/IEC, 2012.

[JB98] Joseph M. Juran and A. Blanton Godfrey. *Juran's Quality Handbook*. McGraw-Hill, 5th edition, 1998.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Reading, 1st edition, 1999.

[JDF+10] Elmar Juergens, Florian Deissenboeck, Martin Feilkas, Benjamin Hummel, Bernhard Schaetz, Stefan Wagner, Christoph Domann, and Jonathan Streit. Can clone detection support quality assessments of requirements specifications? In *International Conference on Software Engineering*, ICSE, pages 79–88. ACM, 2010.

[JM14] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Education, 2nd edition, 2014.

[Jon96] Capers Jones. Activity based software costing. *IEEE Computer*, 29(5):103–104, 1996.

[Jon00] Capers Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[Kal13] Georg Kalus. *Projektspezifische Anpassung von Vorgehensmodellen*. Dissertation, Technische Universität München, 2013.

[KB09] Sven J. Körner and Torben Brumm. Natural Language Specification Improvement With Ontologies. *International Journal of Semantic Computing*, 3(4):445–470, 2009.

[KH15] Jennifer Krisch and Frank Houdek. The myth of bad passive voice and weak words: An empirical investigation in the automotive industry. In *International Requirements Engineering Conference*, RE, pages 344–351. IEEE, 2015.

[KLS95] John Krogstie, Odd Ivar Lindland, and Guttorm Sindre. Towards a deeper understanding of quality in requirements engineering. In *International Conference on Advanced Information Systems Engineering*, CAiSE, pages 82–95. Springer, 1995.

[Kof07] Leonid Kof. Treatment of passive voice and conjunctions in use case documents. *Natural Language Processing and Information Systems*, 4592:181–192, 2007.

[KP96] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: The elusive target. *IEEE Software*, 13:12–21, 1996.

[Kri13] Jennifer Krisch. *Identifikation kritischer Weak-Words aufgrund ihres Satzkontextes in Anforderungsdokumenten*. Diploma thesis, Universität Stuttgart, 2013.

[Kro98] John Krogstie. Integrating the understanding of quality in requirements specification and conceptual modeling. *ACM SIGSOFT Software Engineering Notes*, 23(1):86–91, 1998.

[KS05] Artem Katasonov and Markku Sakkinen. Requirements quality control: a unifying framework. *Requirements Engineering Journal*, 11(1):42–57, 2005.

[Lam09] Axel Van Lamsweerde. *Requirements Engineering*. John Wiley & Sons, 2009.

[LDBvdW15] Garm Lucassen, Fabiano Dalpiaz, Sjaak Brinkkemper, and J.M.E.M. van der Werf. Forging high-quality user stories: Towards a discipline for agile requirements. In *International Requirements Engineering Conference*, RE, pages 126–135. IEEE, 2015.

[LDvdWB16] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering Journal*, 21(3):383–403, 2016.

[Loc13] Klaus Lochmann. *Defining and Assessing Software Quality by Quality Models*. PhD thesis, Technische Universität München, 2013.

[LSS94] Odd Ivar Lindland, Guttorm Sindre, and Arne Solvberg. Understanding quality in conceptual modeling. *IEEE Software*, 11:42–49, 1994.

[LWE01] Brian Lawrence, Karl Wiegers, and Christof Ebert. The top risks of requirements engineering. *IEEE Software*, pages 62–63, 2001.

[Lyu96] Michael R Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

[MCH+12] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data. `http://promisedata.googlecode.com`, 2012.

[Men15] Daniel Mendez Fernandez. *Artefact-based Requirements Engineering Improvement*. Concluding essay of the habilitation procedure, Technische Universität München, 2015.

[MF11] Daniel Méndez Fernández. *Requirements Engineering: Artefact-Based Customisation*. PhD thesis, Technische Universität München, 2011.

[MFME15] Jakob Mund, Henning Femmer, Daniel Méndez Fernández, and Jonas Eckhardt. Does quality of requirements specifications matter? combined results of two empirical studies. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 144–153. ACM, 2015.

[MFNI04] Luisa Mich, Mariangela Franch, and Pier Luigi Novi Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering Journal*, 9(1):40–56, 2004.

[MK09] Thilo Mende and Rainer Koschke. Revisiting the evaluation of defect prediction models. In *International Conference on Predictor Models in Software Engineering*, PROMISE, pages 1–10. ACM, 2009.

[MMFV14] Daniel Méndez Fernández, Jakob Mund, Henning Femmer, and Antonio Vetrò. In quest for requirements engineering oracles: Dependent variables and measurements for (good) RE. In *International Conference on Evaluation and Assessment in Software Engineering*, EASE, pages 3:1–3:10. ACM, 2014.

[MoD11] MoDRE2011. Case study: Canal monitoring and control system (cmcs). Technical report, Model-Driven Requirements Engineering (MoDRE) workshop, 2011.

[MP14] Daniel Méndez Fernández and Birgit Penzenstadler. Artefact-based requirements engineering: the AMDiRE approach. *Requirements Engineering Journal*, 20(4):405–434, 2014.

[MPKB10] Daniel Méndez Fernández, Birgit Penzenstadler, Marco Kuhrmann, and Manfred Broy. A meta model for artefact-orientation: fundamentals and lessons learned in requirements engineering. In *International Conference on Model Driven Engineering Languages and Systems*, MODELS, pages 183–197. Springer, 2010.

[MSB+14] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[MW15] Daniel Méndez Fernández and Stefan Wagner. Naming the pain in requirements engineering: A design for a global family of surveys and first results from germany. *Information and Software Technology*, 57:616–643, 2015.

[MWL+12] Daniel Méndez Fernández, Stefan Wagner, Klaus Lochmann, Andrea Baumann, and Holger de Carne. Field study on requirements engineering: Investigation of artefacts, project parameters, and execution strategies. *Information and Software Technology*, 54(2):162–178, 2012.

[Nus01] Bashar Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–119, 2001.

[PEM13] Birgit Penzenstadler, Jonas Eckhardt, and Daniel Méndez Fernández. Two replication studies for evaluating artefact models in re: Results and lessons learnt. In *International Workshop on Replication in Empirical Software Engineering Research*, RESER, pages 66–75. IEEE, 2013.

[PF13] Birgit Penzenstadler and Henning Femmer. A generic model for sustainability with process- and product-specific instances. In *International Workshop on Green In Software Engineering and Green By Software Engineering*, GIBSE, pages 3–8. ACM, 2013.

[PG12] Carla Pacheco and Ivan Garcia. A systematic literature review of stakeholder identification methods in requirements elicitation. *Journal of Systems and Software*, 85(9):2171–2181, 2012.

[PGH+08] Reinhold Plösch, Harald Gruber, Anja Hentschel, Christian Körner, Gustav Pomberger, Stefan Schiffer, Matthias Saft, and Stephan Storck. The EMISQ method and its tool support-expert-based evaluation of internal software quality. *Innovations in Systems and Software Engineering*, 4(1):3–15, 2008.

[Poh93] Klaus Pohl. The three dimensions of requirements engineering. In *International Conference on Advanced Information Systems Engineering*, CAiSE, pages 275–292. Springer, 1993.

[Poh10] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[PWH+04] Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Daniel Jurafsky. Shallow semantic parsing using support vector machines. *Human Language Technology Conference/North American chapter of the Association for Computational Linguistics annual meeting*, pages 233–240, 2004.

[RMDP16] Alejandro Rago, Claudia Marcos, and J. Andres Diaz-Pace. Identifying duplicate functionality in textual use cases by aligning semantic actions. *Software & Systems Modeling*, 15(2):579–603, 2016.

[RPACJ11] Keith Rayner, Alexander Pollatsek, Jane Ashby, and Charles Clifton Jr. *Psychology of Reading*. Taylor & Francis Ltd, 2nd edition, 2011.

[Sal13] Frank Salger. Requirements reviews revisited: Residual challenges and open research questions. In *International Requirements Engineering Conference*, RE, pages 250–255. IEEE, 2013.

[SB13] Florian Schneider and Brian Berenbach. A literature survey on international standards for systems requirements engineering. In *Conference on Systems Engineering Research*, CSER, pages 796–805. Elsevier, 2013.

[Sch04] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[SK13] Wolfgang Seeker and Jonas Kuhn. Morphological and syntactic case in statistical dependency parsing. *Computational Linguistics*, 39(1):23–55, 2013.

[Som11] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9th edition, 2011.

[SRB00] Forrest Shull, Ioana Rus, and Victor Basili. How perspective-based reading can improve requirements inspections. *IEEE Computer*, 33(7):73–79, 2000.

[Ste16] Daniela Steidl. *Cost-Effective Quality Assurance For Long-Lived Software Using Automated Static Analysis*. PhD thesis, Technische Universität München, 2016.

[Ter13] John Terzakis. The impact of requirements on software quality across three product generations. In *International Conference on Requirements Engineering*, RE, pages 284–289. IEEE, 2013.

[VFJ16] Andreas Vogelsang, Henning Femmer, and Maximilian Junker. Characterizing implicit communal components as technical debt in automotive software systems. In *Working IEEE/IFIP Conference on Software Architecture*, WICSA, pages 31–40. IEEE, 2016.

[VFW16] Andreas Vogelsang, Henning Femmer, and Christian Winkler. Take care of your modes! an investigation of defects in automotive requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ, pages 161–167. Springer, 2016.

[Wag07] Stefan Wagner. *Cost-Optimisation of Analytical Software Quality Assurance*. PhD thesis, Technische Universität München, 2007.

[Wag13] Stefan Wagner. *Software Product Quality Control*. Springer, 2013.

[WDW08] Stefan Wagner, Florian Deissenboeck, and Sebastian Winter. Managing quality requirements using activity-based quality models. In *International Workshop on Software Quality*, WoSQ, pages 29–34. IEEE, 2008.

[Wie05] Karl Wiegers. *More About Software Requirements: Thorny Issues and Practical Advice*. Microsoft Press, 2005.

[WJKT05] Stefan Wagner, Jan Jürjens, Claudia Koller, and Peter Trischberger. Comparing bug finding tools with reviews and tests. In *Testing of Communicating Systems*, TestCom, pages 40–55. Springer, 2005.

[WLH+12] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb, and Jonathan Streit. The quamoco product quality modelling and assessment approach. In *International Conference on Software Engineering*, ICSE, pages 1133–1142. IEEE, 2012.

[WMMY12] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye. *Probability & Statistics for Engineers & Scientists*. Prentice Hall, 9th edition, 2012.

[WRH97] William M. Wilson, Linda H. Rosenberg, and Lawrence E. Hyatt. Automated analysis of requirement specifications. In *International Conference on Software Engineering*, ICSE, pages 161–171. ACM, 1997.

[WRH+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2012.

[YRG+11] Hui Yang, Anne De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering Journal*, 16(3):163–189, 2011.

[YRG+12] Hui Yang, Anne De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. Speculative requirements: Automatic detection of uncertainty in natural language requirements. In *International Requirements Engineering Conference*, RE, pages 11–20. IEEE, 2012.

APPENDIX A

Publications

Table A.1.: Author publications included this thesis

Ref.	Publication
A: [FMM15]	Henning Femmer, Jakob Mund, and Daniel Méndez Fernández. It's the activities, stupid! A new perspective on RE quality. In *International Workshop on Requirements Engineering and Testing*, RET, pages 13–19. IEEE, 2015
B: [FV17]	Henning Femmer and Andreas Vogelsang. Requirements quality is quality in use – a novel viewpoint –. *Submitted to IEEE Software*, 2017
C: [BFE+15]	Mohammad R. Basirati, Henning Femmer, Sebastian Eder, Martin Fritzsche, and Alexander Widera. Understanding changes in use cases: A case study. In *International Requirements Engineering Conference*, RE, pages 352–361. IEEE, 2015
D: [FKV14]	Henning Femmer, Jan Kučera, and Antonio Vetrò. On the impact of passive voice requirements on domain modelling. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 21:1–21:4. ACM, 2014
E: [FKSJ14]	Henning Femmer, Marco Kuhrmann, Joerg Stimmer, and Joerg Junge. Experiences from the design of an artifact model for distributed agile project management. In *International Conference on Global Software Engineering*, ICGSE, pages 1–5. IEEE, 2014
F: [FMJ+14]	Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer. Rapid requirements checks with requirements smells: Two case studies. In *International Workshop on Rapid Continuous Software Engineering*, RCoSE, pages 10–19. ACM, 2014
G: [FMWE17]	Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017
H: [FHEM16]	Henning Femmer, Benedikt Hauptmann, Sebastian Eder, and Dagmar Moser. Quality assurance of requirements artifacts in practice: A case study and a process proposal. In *International Conference on Product-Focused Software Process Improvement*, PROFES, pages 506–516. Springer, 2016
I: [FUG17]	Henning Femmer, Michael Unterkalmsteiner, and Tony Gorschek. Which requirements artifact quality defects are automatically detectable? A case study. In *Fourth International Workshop on Artificial Intelligence for Requirements Engineering*, AIRE, pages 1–7. IEEE, 2017

Publication A: It's the Activities, Stupid! A New Perspective on RE Quality

Authors Henning Femmer, Jakob Mund, Daniel Méndez Fernández

Venue 2nd International Workshop on Requirements Engineering and Testing (RET) at the 37th International Conference on Software Engineering (ICSE), 2015

Abstract [Context] Requirements Engineering (RE) artifacts are central items in software development: Their quality is of essential importance for development, testing and other software engineering activities. However, as requirements artifacts are used differently in different processes, the proper definition of what is good quality depends on the context under consideration. [Problem] So far, no methodology exists that enables to define context-specific RE artifact quality in a precise manner. [Principle Idea] We define context-specific RE artifact quality by how quality attributes of an RE artifact impact on the activities of the software development process in which this artifact is used. [Contribution] In this paper, we introduce a methodology to define RE artifact quality specifically to a project- or process context. Furthermore, we provide a preliminary technical validation as well as an industrial validation on the application of our approach. Our studies indicate that the activity-based approach enables defining and validating RE quality in a precise and systematic manner. The industrial validation furthermore suggests the applicability of the approach in practical use.

Results This paper is summarized in Sections 4.1.1 and 4.2.3.1.

Authors Contributions I co-designed the meta-model, designed and conducted the empirical validation, as well as the provided examples. I reported on the results.

2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing

It's the Activities, Stupid! A New Perspective on RE Quality

Henning Femmer, Jakob Mund, Daniel Méndez Fernández
Technische Universität München, Germany,
Email: {femmer,mund,mendezfe}@in.tum.de

***Abstract*—[Context] Requirements Engineering (RE) artifacts are central items in software development: Their quality is of essential importance for development, testing and other software engineering activities. However, as requirements artifacts are used differently in different processes, the proper definition of what is good quality depends on the context under consideration. [Problem] So far, no methodology exists that enables to define context-specific RE artifact quality in a precise manner. [Principle Idea] We define context-specific RE artifact quality by how quality attributes of an RE artifact impact on the activities of the software development process in which this artifact is used. [Contribution] In this paper, we introduce a methodology to define RE artifact quality specifically to a project- or process context. Furthermore, we provide a preliminary technical validation as well as an industrial validation on the application of our approach. Our studies indicate that the activity-based approach enables defining and validating RE quality in a precise and systematic manner. The industrial validation furthermore suggests the applicability of the approach in practical use.**

I. Introduction

Requirements engineering (RE) artifacts are a basis for communicating the stakeholders' demands. Based on these artifacts, developers produce source code, testers create test cases and customers accept or reject the result. Consequently, RE artifact quality can be crucial for the success of the software engineering endeavor.

Various proposals define a concept of RE artifact quality, most prominently standards such as the IEEE-830 family [1] or its successor, the ISO/IEC/IEEE-29148 [2]. These standards provide a set of characteristics that define a notion of good RE artifacts, e.g. artifacts need to be unambiguous, implementation-free, verifiable, etc. We can classify the characteristics into two types:

First, some characteristics, such as unambiguity and implementation-free[1], describe properties of the RE artifacts. These characteristics can, after further definition and clarification, be diagnosed directly in the artifact. However, the rationale (where does this cause which problems in which situation?) remains implicit and, thus, imprecise. Second, some characteristics, such as verifiability, name activities to be performed with the artifacts [2]. Although in this case the rationale behind this characteristic is clear (e.g. a violation has negative consequences on verification activities), the concrete property of the artifact remains fuzzy, since it strongly depends on the process; for instance, how suited the requirements are for testing also depends on whether the project applies either in-house, explorative testing or a formal testing process.

Consequently, in quality assurance (QA) for RE artifacts we face the problem of a missing guidance of why and how particular characteristics should be inspected in a certain context.

Problem Statement: We are lacking a methodology for defining RE artifact quality for a specific project- or process context in a precise manner. **Contribution:** In this paper, we suggest to analyze how RE artifacts are used in order to define RE artifact quality. To define quality in a certain context, we propose the concept of activity-based RE quality models (ABRE-QM). Quality engineers can use such models to define a context-specific notion of quality. In this work, we define an approach towards the development of such a quality model, and provide a technical validation as well as an empirical evaluation in an industrial setting. **Impact:** Our results support researchers to foster the discussions on how to precisely capture RE quality. In addition, our proposed notion should support practitioners to analyze (and maybe also question), as well as align their RE artifacts with a notion of quality that fits their individual processes.

Outline: The following section shows related work on RE quality and activity-based quality modeling. Afterwards, we introduce ABRE-QM and a methodology how to create an ABRE-QM. This methodology is followed by a technical validation, as well as an industrial validation, in which we exemplarily show how an ABRE-QM is applied in practice and analyze the experiences. The paper concludes with a summary and an outlook on future work.

II. Fundamentals and Related Work

The notion of quality has been widely discussed in RE research and different RE quality models have been proposed so far. Besides the widely-known standards defining quality in terms of a set of attributes demanded of a requirements specification, e.g., [1], [2], several models question quality in RE in a more fundamental way. Lindland et al. [3], [4], [5] model RE quality based on semiotic theory. Syntactic quality is concerned with the absence of errors regarding the languages used, semantic quality with the completeness and correctness (or, validity), and pragmatic quality with the degree to which a specification is understood by its audience. Pohl et al. [6] model RE quality along three fundamental dimensions,

[1] *Implementation-free* refers to the rule that requirements specifications should report on the problem- and not the solution domain.

978-1-4673-7073-8/15 $31.00 © 2015 IEEE
DOI 10.1109/RET.2015.11

namely specification (degree of completeness), representation (degree of formalization), and agreement (degree to which a common view was obtained). Furthermore, depending on the representation used, specific quality models are introduced, e.g. for natural-language specifications, Berry et al. [7] relate manifestations in terms of linguistic defects to understandability, consistency, completeness, and correctness. All these models have in common to focus on intrinsic properties of artifacts rather than on its usage for the engineering endeavor.

In contrast, the basic idea of activity-based quality models is to define quality by how well properties of a product support the activities carried out by the use of the product (see also [8]). Those models, as introduced by Wagner et al. [9] are originally intended to enable the precise definition and evaluation of code quality aspects against maintainability [10] and other non-functional requirements [11], [12]. In order to precisely define context-specific quality, this work transfers the basic concepts of the activity-based quality model to the domain of RE (and its quality assurance) where we support building up a context-specific notion of RE artifact quality.

III. Activity-based RE Quality Models

We strictly understand RE as a supporting means for software engineering, with the goal to produce working software products in a systematic and predictable way. Therefore, the value of the outcome of RE cannot be assessed on its own but must be evaluated in its use as a function to the rest of the engineering endeavor. In this work, we take an artifact-based view on RE where we concentrate on the artifacts rather than on the methods used to create and modify the artifacts. In our understanding, an artifact is any document or data set required in the RE process in its intermediate of final form [13].

A. Metamodel

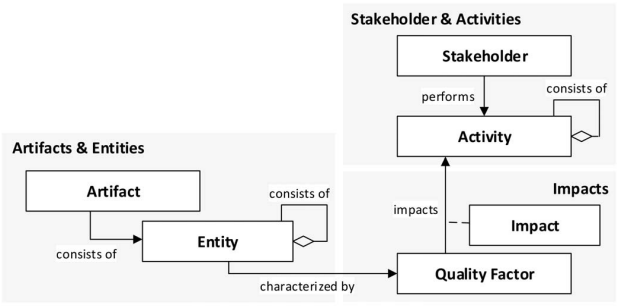


Fig. 1. Metamodel of the activity-based RE quality model based on [9]

To define the notion of the quality of RE artifacts, we adopt the approach of activity-based quality as defined by Wagner et al. [9] by investigating the activities in which the artifact is potentially used as an input. This approach, which results in what we call a *activity-based requirements engineering quality model (ABRE-QM)*, investigates all *stakeholders* (or roles) and the *activities* that use an artifact, and identifies characteristics, called *quality factors* that *impact* the stakeholder's ability to perform his specific activity efficiently and effectively. The meta-model is illustrated in Fig.1.

Furthermore, for each quality factor, the model must provide the following information: a) One or more *Activities* on which this quality factor has an influence; Consequently, we can obtain a structured quality model from those activities, e.g., if the activity is testing, the quality factor will be part of testability in the model. b) Furthermore, for each quality factor and each activity, a rationale has to be given. The rationale includes a reason, i.e., an argumentation why the possession of a specified characteristic (the quality factor) of an artifact impacts the associated activity, consequences on costs, schedule or quality of the developed system if the quality factor is not met, and a source from which this impact was derived and which can provide further information, such as a requirements quality standard or corporate guidelines[2]. c) Lastly, the *Entity* in which the information described by the quality factor is contained. This can be an high-level artifact, such as e.g. `use cases` or more specific definition of the exact content location, such as e.g. `the first step in each use case`.[3]

This metamodel enables us to define quality for specific activities and purposes in a profound manner. However, instantiating this model for a specific purpose requires a thorough analysis of the impacts and activities, which we will present in the following section.

B. Defining an ABRE-QM

In order to give a deeper understanding on how to define an activity-based RE quality model, we describe a process that can be applied to a specific context (i.e. for a certain process, project or company). This process is by no means prescriptive; yet we argue that a systematic methodology towards defining RE quality increases the chance to create a more complete quality model since all stakeholders and artifacts are systematically included in the analysis.

The process contains four steps, following the main concepts of the meta model provided in Fig. 1. These four steps are most probably not executed sequentially, but iteratively until all stakeholders are content with the result.

1. Define RE entities in the project. First, we must analyse which artifacts are used in the context of this ABRE-QM, and determine for which artifacts we define quality. Usually, project repositories (e.g. in version management or file systems or also project tracking systems such as *JIRA*) give a good overview, but sometimes artifacts are also transferred directly between stakeholders. The elicited artifacts are then broken down into their entities. Good candidates for entities are fields or self-containing sections in the artifacts.

2. Elicit stakeholders. Depending on how requirements are used and needed in a project, various stakeholders work with the requirements artifact. These stakeholders have direct

[2]For simplification, we did not include these in Fig. 1.

[3]The ABRE-QM focuses on the *impact* relation. Therefore, we intentionally omitted modelling further existing relations (such as the relation between stakeholders and artifacts).

needs to the requirements artifact and thus must be involved in the definition of the ABRE-QM. Accordingly, missing stakeholders as well as unnecessary stakeholders can lead to a suboptimal definition of the quality model. A project lead is usually a good starting point for finding out who interacts with the RE artifact.

3. Elicit activities with an interface to RE. A good opportunity to find out how RE artifacts are used is to ask the elicited stakeholders how they use the requirements artifacts. This leads to certain, usually coarse-grained activities. These coarse-grained activities have to be broken down into smaller, until we can pinpoint to how a specific stakeholder interacts with the RE artifact.

4. Determine quality factors and impacts. We now have to identify quality factors (properties of the elicited entities) that affect the elicited activities. Currently, we only see heuristics to determine the quality factors: Generally, we need to analyze those activities where the RE entities serve as input artifacts: What helps or hinders to execute this activity and why? What helps or hinders to create an output for this activity efficiently and effectively? For example, there are reports on various quality factors in literature, e.g. in standards (e.g. the requirements language criteria in [2]) or in specific research areas (e.g. the work on requirements ambiguity in [14]). Furthermore, some companies have specific guidelines for their projects. Other sources for quality factors include defect reports, questionnaires in the project or retrospectives. These quality factors are then explicitly linked to the respective activities via impacts. However, we must carefully validate our impacts and determine whether each impact really holds in the context under consideration.

C. Example

The following short example illustrates these ideas.

1. Artifacts and entities: A `Use Case` (e.g. [15]) is a common artifact for specifying functional requirements to software systems. A use case usually contains a `basic flow`, which is a sequence of steps that describes how the user interacts with the system. **2. Stakeholders:** For the sake of simplicity, in this example we will consider only `test engineers`. **3. Activities:** When we analyze how a test engineer in a specific project processes the use case document, we find out that in some contexts the test engineers goes through the steps and `creates test steps` for each element in the sequence. **4. Quality factors:** It is considered good practice in use cases to `enumerate` these steps one by one instead of describing the interaction in a text block. With the aforementioned context and activity in mind, we understand why a use case inhibiting this quality factor is better: The test engineer's task of creating a test sequence can be executed more effectively (and maybe also more efficiently) when the factor is present in the use case. In the remainder of the paper, we will denote this positive ('+') impact with the following shorthand:

$$\textit{Explicit step enumeration @basic flow} \xrightarrow{+} \textit{Create test steps}$$

IV. Preliminary Technical Validation

The goal of this technical validation is to demonstrate the feasibility of the proposed concepts, and to gather experiences in applying our approach to software processes. Therefore, we instantiate a model by applying the ABRE-QM approach to a standardized software process. This serves as a preparation for the application in a real-world scenario described in Sect. V.

The set of activities and artifacts we rely on are taken from the iterative and incremental software development process *unified process* (UP), based on Jacobsen et al. [16]. We chose UP, because it is widely known in both academia and industry, its activities are refined to detailed tasks and described in detail, and so are the expected output artifats created by the activities. The entities in UP therefore provide us with the information necessary to conduct step 1–3 of the procedure described above. Note that although the model is based on activities of the UP, it does not strictly depend on the actual order of how those activities are performed, and thus may be applicable to various other process model variants having same or similar activities.

A. Defining an ABRE-QM

In the first step, we obtain the RE entities, stakeholders, and activities as defined in the UP (step 1–3). In order to determine the quality factors and their impacts (step 4), we analyze the descriptions of the activities and their output artifacts, marking all occurrences where a characteristic of an RE artifact can impact elementary tasks of the corresponding activity. Each quality factor is formulated in such a way that the impact on the associated activity is positive, and a rationale for it is given.

Example. We demonstrate the derivation of quality factors on the activity `design system test` performed by the `test engineer`. According to the UP, one elementary step in designing a system test is to `identify and describe test cases`, producing `test case` artifacts. For both the aforementioned activity and the output artifacts, the definitions provided by the UP need to be analyzed for potential quality factors. For instance, the UP states that test cases shall be created for "requirements whose implementation justifies a test" [16]. One important factor for this is the requirements importance, e.g., in terms of associated priorities[4]. Consequently, we derive the quality factor

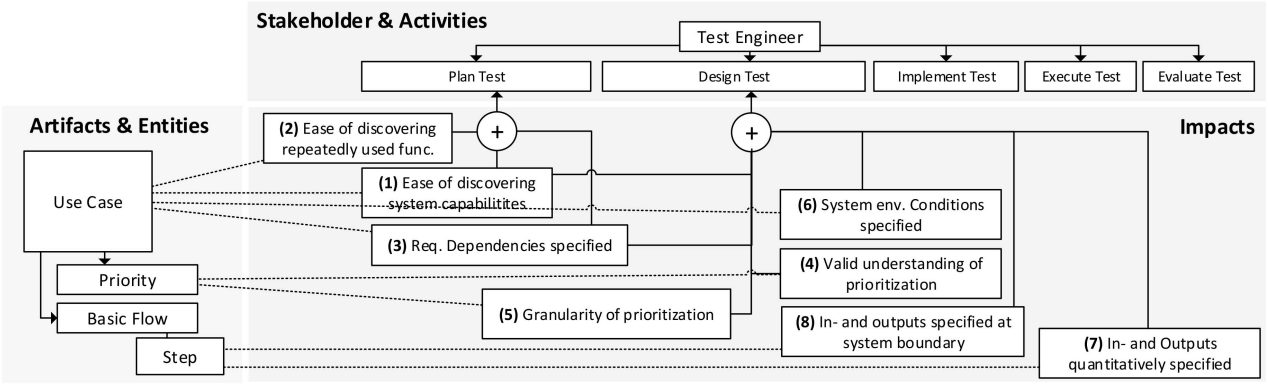
$$\textit{Valid understanding of prioritization @use case} \xrightarrow{+} \textit{Design Test}$$

B. Resulting Model

In the remainder, we identify quality factors that are relevant for the activities in the exemplary discipline (system) `testing`. We opt for this discipline because of its relevance, its strong dependencies with RE, and because of its general representativeness for many engineering processes.

The obtained quality model is illustrated in Fig. 2 together with the rationale for each identified quality factor. According to the UP, RE artifacts are extensively used for planning

[4]Other factors include technical factors, such as the requirements error-proneness.



Q. Factor	Rationale
(1) Ease of discovering system capabilities	In order to estimate the efforts in testing, the system's desired functionality can serve as an indicator. Hence, the system capabilities must be known to the test engineer During the identification of test cases, the engineer must be able to obtain the system's capabilities in order to choose the test cases to be created.
(2) Ease of discovering repeatedly used functionality	Choosing between automated and manual tests is also an economic question. It can be argued that (parts of) system's functionality often used among different scenarios and use cases may be appropriate candidates for test automation.
(3) Requirement dependencies specified	The test plan includes a test schedule, which must reflect the logical order of different req., e.g, that login-functionality preceedes data manipulation scenarios. During the identification of test cases, the engineer must be understand how the requirements influence each other, e.g., that some functionality is always run in parallel, in order to create more approriate testing conditions.
(4) Valid understanding of prioritization	Cf. Prioritization specified. (In addition, prioritization often lack an unambiguous specification of what an associated priority means)
(5) Granularity of prioritization	Cf. Prioritization specified. (The more fine-grained the prioritization, the better the test engineer could optimize his test budgets)
(6) System Environment Conditions specified	Part of the test cases is to describe the environment of the test as close as possible to the productive (end) system. Thus, the engineer must understand under what conditions the system will operate, e.g., regarding amounts of productive data, temperature, workload.
(7) Inputs and Outputs qualitatively specified	In order to be able to produce executable test cases, the system requires quantiatively precise inputs. Furthermore, the system emits quantitative results, where an acceptable range thereof must be defined in a test case.
(8) Inputs and Outputs specified at system boundary	A test case has to be applicable directly on the system, i.e., inputs are submitted and outputs obtained at the system's interface. Inputs or outputs which happen in the environment of the system are not sufficient to create test cases.

Fig. 2. Quality model (with rationales) for system testing according to the Unified Process.

the test strategy (output artifact: `test plan`) and designing system tests (output artifacts: `test cases` and `test procedures`), while for implementation, execution and evaluation, the SRS is not explicitly used at all and, thus, has solely indirect influence. Furthermore, some quality factors influence more than one activity, e.g., requirement dependencies are used both during planning (to arrange a feasible schedule) and designing test cases (to specify appropriate testing conditions).

C. Experiences from the Technical Validation

We were able to derive quality factors (and their impacts) from UP activities and their corresponding output artifacts. However, a pure analysis of the process model description is not sufficient, but a more liberal interpretation is required. That is, a static (idealized) description of a process is not sufficient as the obtained quality factors (and their practical value) highly depend on the person deriving it, especially regarding the expertise in the field and, since definitions are vague, experiences how the activities are carried out in practice. From our experiences, we obtained more and more precise quality factors when asking experienced testers compared to students.

In contrast to the more traditional quality attributes advocated by standards (e.g., [1], [2]), the obtained quality factors are quite different in nature. They are more specific in the sense that they focus on a rather small part of the whole SRS, e.g., the priorities attached to requirements, and could thus demand quite precise characteristics of them, whereas traditional quality attributes are rather cross-cutting concerns in nature, e.g., *all* information must be precise.

V. Industrial Validation

The goal of this validation is to evaluate whether an activity-based quality model can represent a valid, context-specific quality model in practice. Therefore, we conducted a real-world validation with our industry partners. In this study, we translated a set of company-specific use case guidelines, which are used throughout a large re-insurance company, into an ABRE-QM and validated the resulting model with practitioners. The purpose of this validation is to receive qualitative practitioners' feedback on the resulting model in order to understand applicability, and thus let practitioners' evaluation steer the further development of our approach.

A. Study Design

Study Objects. We performed the study at Munich Re, which is one of the worlds leading reinsurance companies with more than 47,000 employees in reinsurance and primary insurance worldwide. For their insurance business, they develop a variety of custom software systems. To elicit the artifacts, stakeholders and activities of a regular Munich Re project, we inspected

the development of a large software project that has passed its initial development and is currently in the maintenance phase. For the impacts, we referred to Munich Re's "use case authoring and review guide" (*guidelines* in the remainder of this paper), which is a 28 pages document that gives detailed instructions on how to describe use cases at Munich Re.

Data Collection. Following the process as described in Sec. III-B, the approach contained four phases: First, we received and analysed a full set of 51 requirements engineering artifacts that were created in the project, including use cases, business rules and others. In the first, 90-minutes workshop with the project lead, we eliminated artifacts that were irrelevant to the guidelines and broke the remaining artifacts down into entities. Furthermore, when the project artifacts did not follow the guideline rules, we extended the model by the entities that were mentioned in the guidelines. Then, the project lead explained the current process of the project, including the users of each artifact and the activities that are performed with these artifacts (Steps 2 and 3 in the process). Furthermore, we defined general activities, such as `find` or `trace` that are generic activities which are independent from the specific roles or which are a basic foundation for each activity, such as `understand`. After the meeting, the authors inspected each of the 50 rules of the guidelines and determined (a) the quality factors that the rule describes, (b) the entities that this rule affects, and (c) the activities that this rule impacts either according to the guideline, if explicitly mentioned, or according to our own experience. This step took approximately 1.5 days.

Although we conducted these steps one-by-one, we iteratively refined over the different phases whenever it was evident that information was missing. This happened particularly often with the activities. For instance, when we discussed the impact of a quality factor, we realized that we needed to add certain activities.

Validation. We validated the model in the second, 90-minutes workshop with an RE lead as well as an experienced developer at Munich Re. This resulted in adding one activity ($1/19 \approx 5\%$) and changing (i.e. adding, removing or altering) 11 impacts ($11/79 \approx 14\%$). In the remaining section, we describe the resulting model after validation.

B. Resulting Model (Exerpt)

The resulting model, after validation with the two experts, contains 36 entities, 5 stakeholders, 19 activities and 79 impacts. In this section, we provide an overview of the resulting model and give some insights into examples. Please note, that it is not our intention to discuss the reasoning in the guideline rules nor the impacts that were given by Munich Re. The goal in this work is to understand whether the meta-model of ABRE-QM enables a definition of quality in this context.

Stakeholders and Activities. The stakeholders involved were requirements engineers, architects, developers, testers, and the department that requests the IT system under development (*customer* in the remainder of this section).

On the activity side, the model defines five activities that form the basis for the remaining, i.e. `find` (successfully searching an information) and `understand` (transferring the intended meaning from the author to the recipient). By extracting this abstract information, we were able to keep impacts on e.g. implementing a use case only if the impact was really specific to this activity and not due to a generalized one of the above. That way we take only direct impacts into account and avoid the model to be cluttered with indirect impacts. The remaining activities range from standard SE activities, such as `derivate test`, to more specific activities, such as `update use case`.

Artifacts and Entities. Munich Re applies use cases with the entities proposed by Cockburn [15]. They furthermore add a small number of semi-formal language constructs, such as references for extension points and subflows.

Quality Factors and Impacts. Instead of presenting all quality factors, we provide three typical examples:

$$\textit{Presence of UI design details @step} \xrightarrow{+} \{\textit{Understand, Implement}\}$$
$$\textit{Presence of UI design details @step} \xrightarrow{-} \{\textit{Maintain, Implement, Use}\}$$

As one example, the experts discussed a common defect in RE artifacts which violates the rule that requirements artifacts should be *implementation-free*, i.e. the artifact should describe the problem domain instead of the solution domain, see also [2]. In the specifications of Munich Re, however, we could observe various violations of this quality attribute, especially in use case artifacts. In the example in Fig. 3, we show the resulting ABRE-QM for UI Design Details (i.e. details on the system's look) on the software development process at Munich Re: The Artifact under consideration is the `use case`, which contains different entities, such as a `name` or `pre-/postconditons` and `basic flow`, which in turn consists of a set of `steps`. For the activities, we show only the most relevant activities in this figure. Regarding the impacts of UI design details, the model argues that they make understanding a use case more efficient, as the visual support can increase the understanding of how the use case is executed. In addition, the UI details might help the tester to run the test case. However, UI details tend to change more often, which can lead to additional maintenance for the requirements engineers. Lastly, these details might lead to a non-optimal solution, which might be rejected by the end-user.

$$\textit{Contains unique ID @name} \xrightarrow{+} \{\textit{Find, Overview, Trace}\}$$
$$\textit{Contains unique ID @name} \xrightarrow{-} \textit{Read}$$

The second rule in the Munich Re guidelines states that each use case should have a unique identifier in the name, e.g. at Munich Re use cases are named in the pattern `<ProductPrefix>-UC<nn> <name>`. The ID in the middle obviously serves purposes of identification and traceability. Furthermore, the experts explained us that the number enables to keep an overview as most file browsers thus display the use cases in a defined order. However, they agreed that the ID makes texts sometimes less efficient to read, a trade-off they are willing to accept for the benefits.

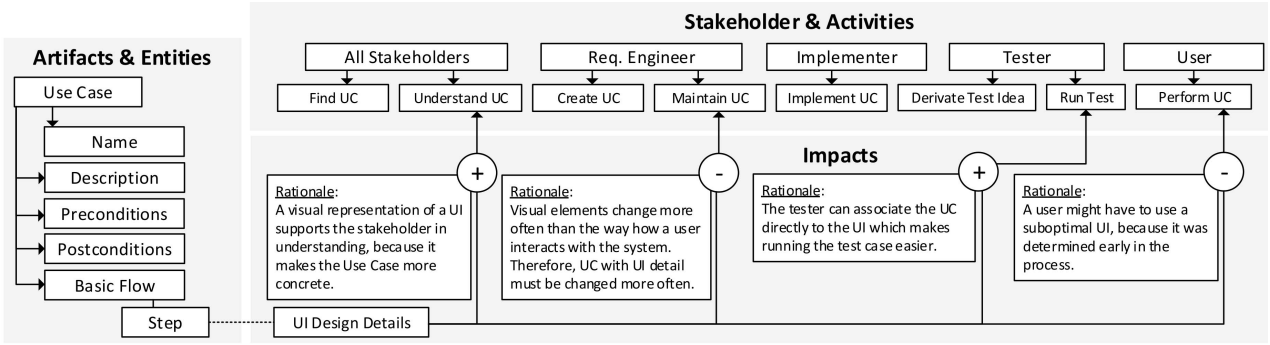


Fig. 3. An example of an activity-based quality definition with implementation details

$$\textit{Is present @subflow} \xrightarrow{+} \{\textit{ Update, Trace}\}$$
$$\textit{Is present @subflow} \xrightarrow{-} \{\textit{ Read, Understand}\}$$

In a final example, we heavily discussed the usage of subflows with the experts. Subflows are mechanisms for reuse that enable the author of a use case to extract a certain set of steps into a reusable subflow to prevent copy-and-past reuse (so-called *cloning* [17]) in the use cases. That way, if parts of the flow change, they only need to be changed in one location (the subflow), and not in each use case [17], hence the positive impact (efficiency) in the `Update` activity. This furthermore creates an explicit `trace` link between the use cases. However, in Microsoft Word use case documents, as currently used by Munich Re, subflows force the reader to jump between different positions in the text in order to `read` through the use case, which can be argued to lead to less readable use cases that are harder to `understand`.

C. Experiences from the Industrial Validation

Our experiences in the industrial validation can be divided into a company and a researcher's perspective.

In order to understand the companies perspective, we asked the project and RE lead after the creation of the model, whether they considered this quality model a useful approach. They responded that they considered this definition of impacts a relevant analysis for understanding guideline correctness and completeness: First, discussing each quality factor and its impacts on activities was seen as a **validation** of the company guidelines. They considered re-evaluating guideline rules that do not have a broad impact on the development process. Also, rules that have positive as well as negative impacts should be debated within the company again, especially if use case authors have issues following them. Second, they considered the model a good starting point for improving guideline **completeness**, through analysis of their process from activities to quality factors: What do the testers need from the requirements specification? What additional quality factors do we need for the developers, etc. The model might enable to formalize these discussions. We will focus on these goals in future validating research.

From a researcher's perspective, we realized during the creation of the ABRE-QM, that the most difficult part is **understanding the impacts** of a certain quality factor: Discussions on the impact often resulted in one example in which the impact was present and a contradicting example in which it was not. Hence, whether there was an impact depended on the context. We see two solutions to this problem: First, further refinement of quality factors and activities can enable to more precisely define the quality factor or the particular activity that is affected. For example, whereas the impact of `UI details` in use cases on the implementation is unclear, it is obvious that `unnecessary UI details` have a negative impact. Similarly we can split up activities to further understand which activities are affected in what way. Second, if the refinement as described before, does not help to clarify the impact, a quality factor may also include multiple impacts on an activity with a defined context describing when a certain impact will hold. This more fuzzy solution prevents that impacts are removed due to conflicting opinions. As future work, we need to understand whether the opinion of different experts intersect regarding the impact of quality attributes.

VI. Discussion

So far, we presented a new approach to derive a quality model based on activities that need RE artifacts. In this section, we reflect on strengths and limitations of such a model regarding the validity and completeness of the notion of quality and the notion of context-specificity, and discuss its use for constructive quality assurance.

A. Model Completeness

Applied to the quality modeling approach in this paper, we consider a quality definition model to be complete if all quality factors which (significantly) impact the activities are identified. Regarding both the technical and industrial validation, we face one fundamental problem; we only consider the impact on the *engineering* in terms of pure creation of a software system. However, RE results may be used for activities beyond engineering, e.g., cost-estimation, project management, maintenance. Therefore, the obtained quality model cannot be complete unless those activities are also considered. Furthermore,

the derivation of quality factors is a cognitive task based on expertise and experience of individuals, and as such, imperfect by definition. Also, the set of quality factors which have *some* effect on the activity outcome may very well be infinite. For practical purposes, we would advise to derive quality factors by several experts independently, and than use consolidation techniques to obtain a final set of quality factors, however, we have no scientific evidence for such claims and it remains future work.

B. Applicability for Quality Assurance

We envisioned the activity-based quality modeling approach to be embedded in practical quality assurance. Therefore, further techniques and tools are required. For quality assessment, we could imagine that our definition of quality might lead to new metrics to measure quality factors, and for constructive quality assessment, techniques such as the one of Requirements Smells [18] for quality factors could provide valuable feedback when writing the requirements specification.

C. Threats to Validity

Our approach, in particular the industrial validation, offers some threats to validity, let alone those inherent to case study research [19] such as subjectivity. However, we were particularly interested in (subjective) practitioners feedback and qualitative insights that would allow us to actively steer the further improvement of ABRE-QMs. This subjectivity also means that we need to expand our investigations in the future to analyse the risks arising from those subjective facets relating to quality modeling for requirements engineering: Do people agree on the impact of certain defects in their requirements specifications?

Finally, our findings can not be generalized as we made our investigations in one company context only involving two subjects. The objective of the validation was, however, not of confirmatory nature, but of exploratory where we wanted to reveal first qualitative insights. Needed are, therefore, further independent investigations for which we have provided the basis. During those independent investigations, it has also to be shown if the approach can be used by others and how exactly it is used (in which particular quality assurance context).

VII. Conclusion and Future Work

In this work, we proposed activity-based quality models (ABRE-QM) that support practitioners in defining a context-specific notion of RE (artifact) quality. We defined a meta-model and suggested a coarse procedure to build such an individual ABRE-QM. Our first technical validation and the industrial evaluation in collaboration with practitioners both indicate that the concepts can be applied in practice. Our results furthermore strengthen our confidence that the approach yields substantiated and detailed quality factors. Practitioners suggested that the approach could improve their requirements guidelines in terms of correctness and completeness, which forms future research.

We identified the proper validation of impacts as a key challenge in the development of ABRE-QMs and we see potential of the quality model to enable a cost-benefit analysis of conflicting (positive as well as negative) impacts through weighting of the different factors. Finally, another question that is currently unanswered is how much the application of the quality model depends on the involvement of us researchers. As we could only provide the first step in this direction, we cordially invite further researchers and especially practitioners to critically discuss our approach, and to join us in (independent) evaluations of our approach to eventually further explore the full spectrum of quality modeling in RE.

Acknowledgments. We thank Jonas Eckhardt, Sebastian Eder, Klaus Lochmann, Sabine Teufl, Antonio Vetro', Benedikt Hauptmann as well as Rudolf Vaas und Alexander Widera from Munich Re for their feedback.

References

[1] "IEEE Recommended Practice for Software Requirements Specifications," IEEE Computer Society, Tech. Rep., 1998.

[2] ISO, IEC, and IEEE, "29148:2011-Systems and software engineering - Life cycle processes - Requirements engineering," Tech. Rep., 2011.

[3] J. Krogstie, "Integrating the understanding of quality in requirements specification and conceptual modeling," *ACM SIGSOFT Software Engineering Notes*, 1998.

[4] J. Krogstie, O. I. Lindland, and G. Sindre, "Towards a deeper understanding of quality in requirements engineering," in *CAiSE*. Springer, 1995.

[5] O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *IEEE Software*, 1994.

[6] K. Pohl, "The three dimensions of requirements engineering: a framework and its applications," *Information Systems*, 1994.

[7] D. M. Berry, A. Bucchiarone, S. Gnesi, G. Lami, and G. Trentanni, "A new quality model for natural language requirements specifications," in *REFSQ*, 2006.

[8] ISO and IEC, "IEC 25010: Systems and Software Quality Requirements and Evaluation (SQuaRE)," Tech. Rep., 2011.

[9] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit, "The Quamoco Product Quality Modelling and Assessment Approach," in *ICSE*, 2012.

[10] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard, "An activity-based quality model for maintainability," in *ICSM*, 2007.

[11] K. Lochmann, "Engineering Quality Requirements Using Quality Models," in *ICECCS*, 2010.

[12] S. Wagner, F. Deissenboeck, and S. Winter, "Managing quality requirements using activity-based quality models," in *WoSQ*, 2008.

[13] Méndez Fernández, D. and Penzenstadler, B. and Kuhrmann, M. and Broy, M., "A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering," in *MoDELS*, 2010.

[14] D. M. Berry, E. Kamsties, and M. M. Krieger, "From Contract Drafting to Software Specification : Linguistic Sources of Ambiguity," Tech. Rep., 2003.

[15] A. Cockburn, "Basic use case template," *Humans and Technology, Technical Report*, vol. 96, 1998.

[16] I. Jacobson, G. Booch, J. Rumbaugh, J. Rumbaugh, and G. Booch, *The unified software development process*. Addison-Wesley Reading, 1999.

[17] E. Juergens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schaetz, S. Wagner, C. Domann, and J. Streit, "Can Clone Detection Support Quality Assessments of Requirements Specifications?" in *ICSE*, 2010.

[18] H. Femmer, D. Méndez Fernández, E. Juergens, M. Klose, I. Zimmer, and J. Zimmer, "Rapid Requirements Checks with Requirements Smells: Two Case Studies," in *RCoSE*, 2014.

[19] P. Runeson and M. Höst, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *EMSE*, 2009.

Publication B: Requirements Quality is Quality in Use – A Novel Viewpoint

Authors Henning Femmer and Andreas Vogelsang

Venue Submitted to IEEE Software

Abstract The quality of requirements engineering artifacts is widely considered a success factor for software projects. Currently, the definition of high-quality or good RE artifacts is often provided through normative references, such as quality standards, text books, or generic guidelines. We see various problems of such normative references: (1) It is hard to ensure that the contained rules are complete, (2) the contained rules are not context-dependent, and (3) the standards lack precise reasoning why certain criteria are considered bad quality. To change this understanding, we postulate that creating an RE artifact is rarely an end in itself, but just a means to understand and reach the project's goals. Following this line of thought, the purpose of an RE artifact is to support the stakeholders in whatever activities they are performing in the project. This purpose defines high-quality RE artifacts. To express this view, we contribute an activity-based RE quality meta model and show applications of this paradigm. Lastly, we describe the impacts of this view onto research and practice.

Extended Summary This paper is summarized in Sections 4.1 and 6.2.

Authors Contributions I co-designed and co-reported the work.

Requirements Quality is Quality in Use – A Novel Viewpoint –

Henning Femmer and Andreas Vogelsang

***Abstract*—The quality of requirements engineering artifacts is widely considered a success factor for software projects. Currently, the definition of high-quality or good RE artifacts is often provided through normative references, such as quality standards, text books, or generic guidelines. We see various problems of such normative references: (1) It is hard to ensure that the contained rules are complete, (2) the contained rules are not context-dependent, and (3) the standards lack precise reasoning why certain criteria are considered bad quality. To change this understanding, we postulate that creating an RE artifact is rarely an end in itself, but just a means to understand and reach the project's goals. Following this line of thought, the purpose of an RE artifact is to support the stakeholders in whatever activities they are performing in the project. This purpose defines high-quality RE artifacts. To express this view, we contribute an activity-based RE quality meta model and show applications of this paradigm. Lastly, we describe the impacts of this view onto research and practice.**

***Index Terms*—Quality, quality standards, requirements, documentation, roadmap**

Three Actionable Insights:

- Always remember: Requirements engineering artifacts are *a means, not an end*.
- Therefore, before writing your requirements, think about *the readers* and *how they use the artifacts* first.
- Use this simple model to define who is using RE artifacts, what RE artifacts are used for, and how RE artifacts should therefore look like.

I. Current Standards are Incomplete, Inadequate and Imprecise on Requirements Quality.

Requirements Engineering (RE) artifacts are central entities in the software engineering process. Based on these artifacts, project managers estimate effort, designers create architectures, developers build the system, and test managers set up a test-strategy. Consequently, quality defects in RE artifacts can cause expensive consequences in subsequent software development activities. Therefore, quality control of RE artifacts is key for successful software development projects.

The definition of high-quality or good RE artifacts is often provided through normative references, such as quality standards or text books (e.g., ISO/IEEE/IEC-29148 [1]). We see various problems of such normative references.

Quality standards are incomplete. Several quality standards describe quality through a set of abstract criteria. When analyzing the characteristics in detail, we see that there are two different types of criteria: Some criteria, such as ambiguity, consistency, completeness, and singularity are factors that describe properties of an RE artifact itself. In contrast, feasibility, traceability and verifiability state that activities can be performed with the artifact. This is a small, yet important difference: While the former can be assessed by analyzing just the artifact by itself, the latter describe a relationship of the artifact in the context of its usage. Yet this usage context is incompletely represented in the quality standards: For example, why is it important that requirements can be implemented (feasible in the terminology of ISO-29148) and verified, but other activities, such as maintenance, are not part of the quality model? Therefore, we argue that normative standards do not take all activities into account systematically, and thus, are missing relevant quality factors.

Quality standards are not context-dependent. One could go even further and ask about the value of some artifact-based properties such as singularity or formality. Still widely cited quality models of the past [2] proclaimed that (all) projects should strive towards formalized requirements. What is the purpose and reason behind such a property? A normative approach does not provide rationales. This is different for activity-based properties, such as verifiability, since these properties are defined by their usage: If we need to verify the requirements, properties of the artifact that increase verifiability are important. In particular, we need to understand up-front how we want to verify the requirements. For a formal verification, formalized requirements are a reasonable approach. For manual testing, however, formalized requirements might actually make them harder to understand and, therefore, harder to test. This example shows that, in contrast to the normative definition of quality in RE standards, RE quality usually depends on the usage context.

Quality standards lack precise reasoning. For defining most of the aforementioned criteria, the standards remain abstract and vague. For some criteria, such as ambiguity, the standards provide detailed lists of factors to avoid. However, these criteria have an imprecise relation to both the abstract criteria mentioned above as well as to any kind of reasoning. Consequently, the harm that these criteria might cause remains unclear.

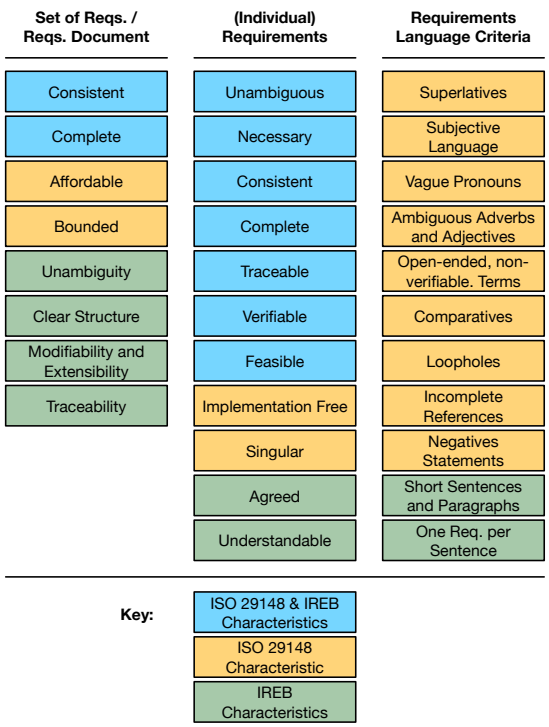


Fig. 1: This figure depicts the quality characteristics of ISO 29148 and the IREB syllabus in comparison. Blue characteristics are shared characteristics, orange and green characteristics appear only in one of the standards. Please note that, as we discuss in the text, some characteristics are shared between the standards by their name, but vary in the precise meaning of the characteristics.

II. Sidebar: Comparison of RE quality standards

To get a taste of current RE quality standards, we compare the ISO/IEC/IEEE-29148 [1] quality standard with the definition of quality attributes from the curriculum of the International Requirements Engineering Board (IREB), a certification also widely used in industry.

Both standards define a quality model through a simple list of characteristics. According to the standards, good requirements documents are those in which these characteristics are present. The standards share nine of the characteristics (see Fig. 1), mostly those characteristics defined in earlier literature and standards, such as the IEEE 830. However, the standards disagree on more characteristics than they agree on. In particular, the standards completely disagree when it comes to concrete language criteria. And even when the standards agree on the characteristics, as soon as they define the characteristics, their interpretations differ significantly. Take, for example, consistency. While the IREB definition only considers disagreeing requirements as inconsistent, the ISO 29148 definition of inconsistency also includes duplication issues and terminological deficiency. In addition, the IREB assesses quality characteristics on a continuous scale, whereas the definitions of the ISO standard suggests a boolean interpretation.

At a glance, both standards share the same approach towards quality, but their details differ tremendously. This is especially true for the concrete, assessable language criteria. We argue that these differences indicate two problems. First, the missing agreement on the level of concrete language criteria indicates that we do not yet know what is good or bad quality, and that we have little to no established understanding of the impacts of concrete language criteria. Second and even more problematic, the missing agreement at the level of abstract quality characteristics indicates there is neither an established understanding about nor an established approach towards quality for RE artifacts as a whole.

III. Goals of Requirements Engineering

Let us take a step back. If we want to get to the bottom of RE artifact quality, we need to reconsider the goals of requirements engineering itself since RE artifacts should eventually support the goals of RE. Following the definitions of the goals of RE as understood by Glinz [3, p.18], we understand quality in RE as the degree to which the following goals are sufficiently fulfilled for system stakeholders as well as the project team:

① **Understand stakeholders' needs**: In our understanding, high quality in RE is the degree of correct and complete understanding of the goals, expectations and constraints of the system stakeholders.

② **Achieve agreement**: In addition, high quality in RE is the degree of agreement on a system that manifests the consensus of all system stakeholders. To this end, high quality in RE correctly prioritizes requirements, and ensures that a best-possible solution is derived for the system stakeholders' needs (iteration between problem and solution space, see twin-peaks model [4]).

③ **Create the same mental model between all system stakeholders**: Furthermore, high quality in RE is the degree to which these system stakeholders' needs and the derived consensus is correctly and completely communicated between all involved system stakeholders in the project.

④ **Structure & manage requirements-based activities**: Lastly, many project activities are structured along the system stakeholders' needs, e.g. in the form of requirements. Some exemplary activities are estimating costs and schedule of the system, developing the system or testing the system. Consequently, high quality in RE is the degree to which engineers working with the requirements (i.e. the information) can efficiently and effectively use the requirements to execute their requirements-based

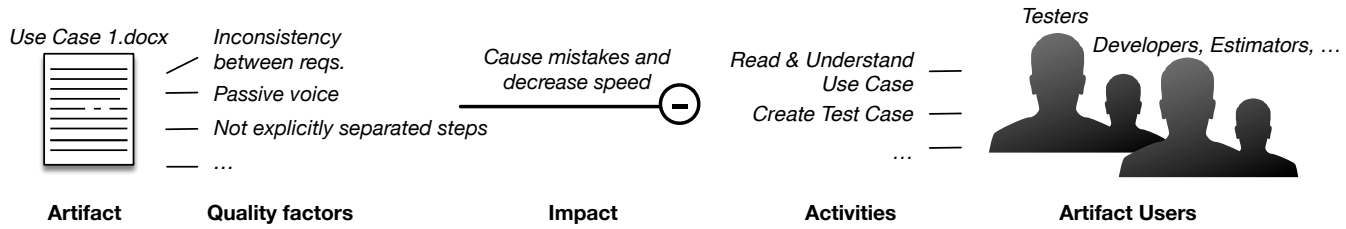


Fig. 2: This figure shows the general idea: High or low quality means that the artifact shows quality factors which impact activities of the user of the artifact.

activities. This can include being able to handle changing requirements over time, if necessary in the project.

These goals could be achieved without RE artifacts. However, RE artifacts support achieving these goals of RE, especially if a project is difficult to overview due to its size or product structure complexity. In these cases, projects can benefit from relying on RE artifacts to effectively and efficiently fulfill the purpose of RE. Some projects try to execute requirements engineering without RE artifacts.

IV. A Different View on Requirements Quality

In the following, we want to describe our novel approach towards requirements artifact quality. For this, we first describe the basic concepts, then the detailed model and afterwards an exemplary quality factor.

A. The Idea

We postulate that creating an RE artifact is rarely an end in itself, but just a means to reach the project's goals. In particular, they are a tool to reach the goals of RE, as described in the previous section. Following this line of thought, the purpose of a requirements artifact is to support the stakeholders in whatever activities they are performing in the project (see Fig. 2). This change of view means that it is unreasonable to talk about good or bad RE artifacts in general. What is good and what is bad must always be assessed with respect to the given context. More specifically, good quality depends on the RE artifact stakeholders and the activities that they conduct with the RE artifacts. In fact, we argue that common quality criteria, even completeness and correctness, have to be rethought from a quality-in-use perspective. This contributes a novel view on requirements engineering artifact quality, which discusses RE artifact quality from a quality-in-use viewpoint.

B. The Model

To precisely define RE artifact quality, we designed activity-based RE artifact quality models (ABRE-QMs). First, to describe the structure of ABRE-QMs, we provide an ABRE-QM meta model that introduces the concepts needed to describe an ABRE-QM.

The ABRE-QM meta model adapts and extends the QUAMOCO meta model [5]. The QUAMOCO meta model is used to explicitly define quality-in-use characteristics of source code, such as maintainability [6]. We simplify, but also extend the meta model to adapt it to RE artifact quality.

ABRE-QMs define quality as an instance of the following elements (see Fig. 3):

An artifact is a documented collection of requirements entities, which is produced during an RE process. An example for an artifact is a *use case document*.

An entity is a coherent documented information. An entity can be an information content item, but can also be further decomposed, e.g. into the linguistic components of such a content item. Examples for entities are a *use case*, an *alternative flow* or a *step* within the flow.

A stakeholder role is the role of someone with an interest in the RE artifact [7], such as a test engineer. Each role can include more generic roles. For example, both *test engineer* and *developers* are also *readers* of the requirements artifact. Therefore, quality factors that affect the activity *read*, affect all *readers* of the artifact, including *test engineers* and *developers* through their included generic role *reader*. This allows combining shared activities that multiple stakeholders must execute.

An activity is an invested effort, which involves one or more of the aforementioned artifacts, such as creating test cases, and one or more of the aforementioned stakeholder roles, such as the *test engineer*. An activity can be broken down into subactivities. For example, the testing activity is decomposed into *creating*, *running*, and *maintaining* test cases.

A quality factor is a property that is or is not present in an entity. This property must be objectively assessable through a measure to be used for quality control.

An impact is an explicit relation between a quality factor and an activity. The impact influences either *effectiveness* or *efficiency* of that activity. This impact is explicitly discussed through: First, a reason, i.e. an argumentation why the presence of a specified characteristic (the quality factor) of an artifact impacts the associated activity; second, consequences on costs, schedule or quality of the developed system; and third, a source from which this impact was derived and which can provide further information, i.e. a requirements quality standard or corporate guidelines.

A context factor influences the impact of a quality factor. For example, the problematic impact of a *passive voice* requirement varies, depending on the background of the reader. If the reader has no or few domain knowledge, the passive voice has a stronger impact. In contrast, in cases where the reader is well aware of the domain and the ideas of the system, the impact can be less problematic. Context factors can be human, process or tool factors.

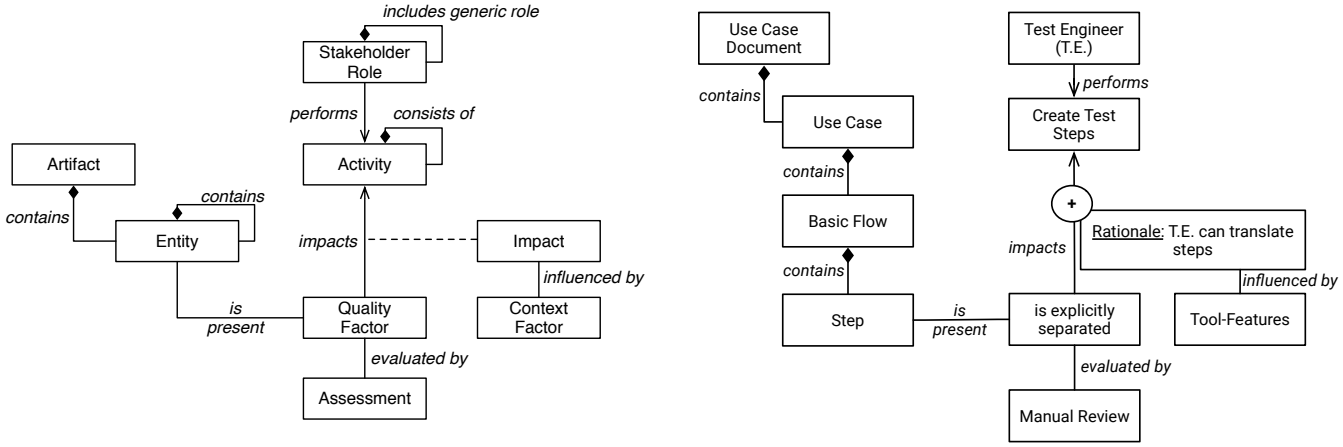


Fig. 3: This figure shows the ABRE-QM meta model and a simple example of a quality factor in an ABRE-QM. The meta model consists of artifacts and their decomposition into entities, quality factors and their impact on activities, which are performed by certain stakeholder roles. Impacts are influenced by context factors. Lastly, quality factors are evaluated by assessments. The example discusses why explicitly separated steps in basic flows of use cases are considered good quality. In this example, we discuss the impact on creating test steps, i.e. explicitly separated steps in basic flows allow more efficient and effective creation of test steps through reuse.

An assessment is a description for evaluating an entity against a quality factor. The application of an assessment against an entity (analogue to Deissenboeck [6]) results in a (potentially empty) set of quality defects. Just as Deissenboeck describes, we see three potential categories of assessments: manual, automatic, and semi-automatic assessments.

C. An Exemplary Quality Factor

To foster understanding, this section provides an exemplary excerpt of an ABRE-QM (see Fig. 3). The example shows the definition of one quality factor, namely the presence of explicit steps in a use case flow.

1. Artifacts and entities: A *use case* is a common artifact for specifying functional requirements to software systems. A use case usually contains a *basic flow*, which is a sequence of steps that describes how the user interacts with the system.

2. Stakeholder roles: For the sake of simplicity, in this example, we consider only *test engineers*.

3. Activities: When we analyze how a test engineer processes the use case document in a specific project, we discover that, among other activities, the test engineers goes through the use case steps and *creates test step(s)* for the use case's basic flow.

4. Quality factors: It is considered good practice in use cases to *explicitly separate* each step instead of describing the whole basic flow in one text block. With the aforementioned context and activity in mind, we understand why a use case with this quality factor is considered higher quality: The test engineer can directly translate the use case steps to test steps. Therefore, the test engineer's task of creating a test sequence can be executed more effectively (and maybe also more efficiently) when the factor is present in the use case. Fig. 3 explicates this reasoning through a positive ('+') impact in an ABRE-QM. Please note, that for simplicity, we only discuss one of the impacts of this quality factor here.

5. Context factors: One could consider the applied tool environment to be a context factors. Depending on the concrete tools in use, the translation is more or less efficient.

6. Assessment: One could discuss various types of assessments, depending on the tools used. An easy-to-apply assessment is a manual review, which can spot this quality defect. In addition, for various requirements management tools, one could discuss automatic (or at least semi-automatic) methods through automatic analysis of the use case's structure.

This example shows the definition of one quality factor. An ABRE-QM is a composition of a set of such quality factors with their respective relations. RE artifact quality is thus defined through an ABRE-QM.

This model enables researchers to provide practitioners with a precise definition of what they consider to be bad quality, why (i.e., due to which consequences) and in which context (i.e., based on which activities). Practitioners can then use such a precise quality model and, based on artifacts, activities and impacts, decide which quality characteristics are relevant for their context.

V. Applications in Research and Practice.

We have applied the proposed meta model for different purposes. The meta model proved beneficial in several contexts that are discussed in the following.

Activity-based RE Guidelines. Nowadays, many companies have generic guidelines to help employees improve their requirements and to create a baseline for quality. We argue that guidelines that are defined in an activity-based manner could help to make these guidelines more complete, precise, and specific for their context. In a first study [8], practitioners reported that a translated guideline helps to both discuss validity of the existing rules and to create more complete guidelines.

Activity-based Tailoring of Requirements Templates. Requirements templates are blueprints that determine the syntactic structure of a single requirement. One reported advantage of requirements templates is that they facilitate more *complete* specifications of requirements. However, what complete actually means depends on how requirements are used: The information that needs to be provided in a requirement is determined by the activities that are performed based on the requirement. In a recent paper [9], we used activity-based models to tailor requirements templates in a way that the information they demand for a requirement fit the actual usage in a specific development context. The result is a set of requirements templates that are more specific and expressive than general templates that are proposed to fit every situation.

Activity-based Cost Estimation. We used the proposed meta model to develop cost models to enable an informed decision making process: In a recent study [10], we used an instance of the meta model to characterize the cost and benefits of refactoring functional parts that reoccur in several functions of a system specification. The decision whether a refactoring pays off heavily depends on the context in which the respective system specification is used. Therefore, we identified activities that are performed with the system specification, and we identified cost factors that affect these activities in the original and the refactored version. Cost factors are a specific form of quality factors as present in the meta model. As a result, the decision whether to refactor a specification or leave it as it is can be assessed with respect to the usage context.

Activity-based Quality Assurance. The presented paradigm also has strong implications on quality assurance. This is both for constructive aspects, such as tailoring guidelines to requirements use, but also analytical approaches, such as requirements smells [8], where ABRE-QMs enabled us to decide which quality characteristics should hold in a certain context.

Activity-based Impact of RE Quality in a Common Theory. Lastly, this paradigm helps steer and unite research by providing it with a common theory: Research can be structured along quality factors and thus focus on which activities are impacted by a certain quality factor. That way, both defining the quality factor and understanding its impact follows a precise structure. We followed this example in our experiment on the impact of passive voice on understanding requirements [11].

VI. So What?

Based on our activity-based model, we can categorize where the research in the field should be heading towards.

A. What Should Practitioners Do?

If you are a practitioner trying to improve the RE process in your company by increasing the artifact quality, we would argue that these steps help to create more efficient and effective requirements artifacts:

1) Always remember: Requirements engineering artifacts are *a means, not an end*.
2) Therefore, before writing requirements, we suggest to think about *the readers* and *how they use the artifacts* first: Which information do they need? Afterwards, we advice to create a model of the RE artifacts produced in the company, the stakeholders that use these artifacts, and the activities that the stakeholders perform. The meta-model presented in this article can help to structure this model.
3) What is most important in our opinion is to talk to the stakeholders who use an artifact to assess what helps or hinders them in performing their tasks. Their experience can be included as quality factors to the quality model.
4) After the basic model of artifacts, activities, and quality factors is established, one can start introducing company-wide guidelines or quality assurance measures based on this model. We argue that these guidelines and measures are more complete, specific, and justified than normative references, because they have a direct relation to activities that directly profit from them. Requirements patterns may help to remind the engineers of the information they need to document (see [12]).
5) Lastly, it is important to evolve and maintain this model over time. The stakeholders, activities, and therefore also the notion of artifact quality may change over time or there might be new quality factors that should be added. Additionally, a change in some context factors (e.g., tooling improvements) may mitigate negative impacts. This could mean that maintaining some quality factors may no longer be relevant.

B. What Is Left For Research?

The activity-based approach for quality definitions strongly benefits from a unified and well-tested body of quality factors and related impacts. If the research continues along this theory, the community can, together, create a generic ABRE-QM, which will resembles the existing knowledge on RE artifact quality. The precision of such a theory would allow researchers to systematically discuss results in the field and its focus on activities would enable practitioners to understand and weigh consequences of bad quality in a structured manner. In the long run, this paradigm could even be extended beyond artifacts to create a general RE quality theory.

To accomplish this vision, researchers should work on the following topics:

- Create a **reference artifact and usage model** that serves as a complete list of possible stakeholders, their most important activities, and typical artifacts that are used in the activities. Practitioners may use this reference model as a starting point for a company-specific model.
- Create a **taxonomy of quality factors** that serves as a body of knowledge of quality factors. To observe these, one can look through produced artifacts, e.g.:
  - Review protocols that indicate the effort that was invested during QA of the RE artifacts (as we, for example, did in an earlier paper [13])
  - Incorrect test cases or incorrect test results that show that the test case engineer misunderstood the RE artifacts

  - Requirements change requests, or defects in bug tracking systems that can be traced back to RE artifacts, to understand defects in the RE artifact that are discovered during development or further activities
  - Concrete changes that have been performed on the RE artifacts to understand maintenance efforts (as for example performed by Basirati et al. [14])
- Create a **taxonomy of impacts** that provides a list of well examined effects of quality factors on activities. To evaluate impacts, one may use interviews (see [15],), case studies (see [14]), or experiments (see [11])

VII. Conclusions

The strength of our approach, as reported by practitioners [15], lies its clarity of reasoning. Taking activities as the basis provides a simple rule whether or not something is of better or worse quality: *If it hinders someone, it is bad quality*. This rule, at the same time, generates falsifiable hypotheses for each postulated rule for good or bad quality. We argue that this quality model enables research to both argue more clearly about their results, but also conduct better studies, with a clearer research focus. We furthermore argue that this model provides practitioners with a precise and valid approach to understand: What are good requirements artifacts in my case? We defined follow-up steps to be conducted by researchers, in order to built a precise common understanding of RE artifact quality,

Acknowledgements

This work was performed within the project Q-Effekt; it was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS15003 A-B. The authors assume responsibility for the content.

PLACE PHOTO HERE

Henning Femmer Henning Femmer is a Ph.D. candidate at Technical University Munich (TUM) and co-founder of the requirements consulting company Qualicen. His research focusses on improving the efficiency and effectiveness of requirements quality control, with a particular focus on automatic methods. He publishes at academic venues, such as ICSE, RE, ESEM, but also speaks at industy-focussed events, such as ReConf or Embedded World. In both his research and practical work he aims to combine scientific rigor with industrial applicability in order to efficiently deliver high quality.

PLACE PHOTO HERE

Andreas Vogelsang Andreas Vogelsang is a professor for systems engineering at the Berlin Institute of Technology (TU Berlin). He is also a research director at the Daimler Center for Automotive IT Innovations. His research interests comprise model-based requirements engineering, requirements specification quality, and software architectures for software-intensive systems. He has published over 20 publications in international journals, conferences, and workshops including ICSE, RE, and REFSQ. Additionally, he participated in several research collaborations with industrial partners especially from the automotive domain. Most of the research collaborations focused on how to create high-quality requirements artifacts and how to improve the RE process.

References

[1] ISO/IEC/IEEE, "Systems and software engineering – Life cycle processes – Requirements engineering," International Organization for Standardization, Geneva, Switzerland, ISO/IEC/IEEE 29148:2011(E), 2011.

[2] K. Pohl, "The three dimensions of requirements engineering," in *International Conference on Advanced Information Systems Engineering*, ser. CAiSE, 1993, pp. 275–292. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-56777-1_15

[3] M. Glinz, "A glossary of requirements engineering terminology," International Requirements Engineering Board and University of Zurich, Tech. Rep., 2014.

[4] B. Nuseibeh, "Weaving together requirements and architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115–119, 2001.

[5] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit, "The quamoco product quality modelling and assessment approach," in *International Conference on Software Engineering*, ser. ICSE, 2012, pp. 1133–1142. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337372

[6] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard, "An activity-based quality model for maintainability," in *International Conference on Software Maintenance and Evolution*, ser. ICSM, 2007, pp. 184–193. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4362631

[7] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[8] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, 2016.

[9] J. Eckhardt, A. Vogelsang, and H. Femmer, "An approach for creating sentence patterns for quality requirements," in *International Workshop on Requirements Patterns (RePa)*, 2016.

[10] A. Vogelsang, H. Femmer, and M. Junker, "Characterizing implicit communal components as technical debt in automotive software systems," in *Working IEEE/IFIP Conference on Software Architecture*, ser. WICSA, 2016, pp. 31–40. [Online]. Available: http://www4.in.tum.de/ vogelsan/publications/WICSA16.pdf

[11] H. Femmer, J. Kucera, and A. Vetrò, "On the impact of passive voice requirements on domain modelling," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2014.

[12] J. Eckhardt, A. Vogelsang, H. Femmer, and P. Mager, "Challenging incompleteness of performance requirements by sentence patterns," in *International Requirements Engineering Conference (RE)*, 2016.

[13] A. Vogelsang, H. Femmer, and C. Winkler, "Take care of your modes! An investigation of defects in automotive requirements," in *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2016.

[14] M. R. Basirati, H. Femmer, S. Eder, M. Fritzsche, and A. Widera, "Understanding changes in use cases: A case study," in *International Requirements Engineering Conference (RE)*, 2015.

[15] H. Femmer, J. Mund, and D. Méndez Fernández, "It's the activities, stupid!: A new perspective on RE quality," in *International Workshop on Requirements Engineering and Testing (RET)*, 2015.

Publication C: Understanding Changes in Use Cases: A Case Study

Authors Mohammad R. Basirati, Henning Femmer, Sebastian Eder, Martin Fritzsche, Alexander Widera

Venue 23rd IEEE International Requirements Engineering Conference (RE), 2015

Abstract Requirements change and so (should) do requirements artifacts, such as use cases. However, we have little knowledge about which changes requirements engineers actually perform on use cases. We do not know what is changing, at which locations use cases change and need a deeper understanding of which changes are problematic in terms of difficult or risky.

To explore these challenges from an industrial point of view, we conducted a mixed methods case study in which we analyze 15 month of changes in use cases in an industrial software project.

The study provided interesting observations for both practitioners and researchers involved: First, the most frequently changing use cases had an issue in their structuring. Second, alternative flows (i.e., variations or extensions of the main flow) were especially prone to changes. Third, changes in content (semantic changes) and in presentation of the content (syntactic changes) happen similarly frequently. Last, a qualitative and quantitative analysis aiming at a deeper understanding of problematic changes identified taxonomy changes, as well as locally or temporally dispersed changes as particularly difficult and risky.

In this paper, we contribute a first empirical inquiry for understanding the maintainability of use cases: The presented study provides empirical evidence that there are particular maintenance risks and suggests to continuously analyze local and temporal dispersion.

Extended Summary This paper is summarized in Section 4.2.3.2.

Authors Contributions I co-designed the case study, I executed the interviews, I co-analyzed and reported the results.

Understanding Changes in Use Cases: A Case Study

Mohammad R. Basirati, Henning Femmer, Sebastian Eder
Technische Universität München,
Germany
m.basirati@tum.de, {femmer, eders}@in.tum.de

Martin Fritzsche, Alexander Widera
Munich Re,
Germany
{MFritzsche, AWidera}@munichre.com

***Abstract*—Requirements change and so (should) do requirements artifacts, such as use cases. However, we have little knowledge about which changes requirements engineers actually perform on use cases. We do not know *what* is changing, *at which locations* use cases change and need a deeper understanding of which changes are *problematic* in terms of difficult or risky.**

To explore these challenges from an industrial point of view, we conducted a mixed methods case study in which we analyze 15 month of changes in use cases in an industrial software project. The study provided interesting observations for both practitioners and researchers involved: First, the most frequently changing use cases had an issue in their structuring. Second, alternative flows (i.e., variations or extensions of the main flow) were especially prone to changes. Third, changes in content (semantic changes) and in presentation of the content (syntactic changes) happen similarly frequently. Last, a qualitative and quantitative analysis aiming at a deeper understanding of problematic changes identified taxonomy changes, as well as locally or temporally dispersed changes as particularly difficult and risky.

In this paper, we contribute a first empirical inquiry for understanding the maintainability of use cases: The presented study provides empirical evidence that there are particular maintenance risks and suggests to continuously analyze local and temporal dispersion.

***Index Terms*—Requirements Engineering, Artifacts, Use Cases, Change, Maintainability**

I. Introduction

Requirements Engineering (RE) artifacts play a central role in many software development projects [12]: RE, Customers and other stakeholders use it to communicate stakeholder's needs, developers create software and testers create systems tests based on these artifacts, to name only a few roles. One common way to document requirements, are use cases (see, e.g., [16]).

In long living software systems, the requirements to the system change over time. With changing requirements, changing the use case artifacts is inevitable: If the use cases are not maintained over time, they become useless to stakeholders, since they do not reflect the (currently intended) system behavior [2].

Changes to requirements artifacts can be of different nature: Some changes target the functionality (the semantics) of the system, leading to changes in other artifacts, such as test cases or source code. Other changes alter only the presentation (on a syntactical level) of a requirement, for example restructuring use cases or improving the understandability of the written text.

In both cases, changing a use case can require a high effort. The reason for this are interconnected use cases, where a change in one use case leads to changes in various other use cases. For example, changing the user interaction for storing certain data might require changes in use cases for creating, reading, updating and deleting this data and, furthermore, in other downstream artifacts such as test cases or source code.

Additionally, changing use cases is risky, since a change might introduce wrong requirements or requirements that are hard to understand. The latter leads to difficulties in subsequent software engineering activities, such as testing or implementing the system. In the case of interconnected requirements artifacts, changes might introduce inconsistencies between these artifacts, which leads to possibly contradicting requirements.

How difficult it is to keep the use cases up-to-date with the changing requirements, depends on the use case's *maintainability*: The risk and the efforts should decrease with an increasing maintainability of the requirements documents.

Problem: There is a lack of knowledge how use cases change over time. In order to be able to manage requirement artifacts change and foster an understanding of maintainability of requirements, we need to empirically investigate requirements change in real world projects.

Approach: This paper aims at exploring maintenance in use cases. In an industrial case study, we manually inspect more than 400 changes in a corpus of 32 use cases over the time of 15 months. In order to validate our results and gain additional insights we pair with industry practitioners through interviews.

Our approach is split into three steps: We first identify change hotspots in the form of often changing use cases and use case parts and analyze the reason for these changes. Afterwards, we inspect the changes in depth by developing a taxonomy of use case changes, and by classifying all changes in an industrial project of the reinsurance company Munich Re. We furthermore investigate, which of the changes impose higher risk or effort.

Contributions: The main contribution of this study are: a) the identification of use cases and parts of use cases, where maintenance happens most frequently, b) a taxonomy of concrete changes in use cases, and c) an elaboration of which changes to use cases lead to problems in our study object.

Impact: The taxonomy presented in this paper enables a more structured investigation of the maintenance activities in use cases, which can be helpful for researchers in order to understand maintenance and can be helpful for practitioners to understand the state of their project. Furthermore the

978-1-4673-6905-3/15/$31.00 © 2015 IEEE

RE 2015, Ottawa, ON, Canada
Industry Paper

identification of problematic changes points to systematic problems that have to be investigated in detail. The study shows cases in which maintainability of use cases can have a severe impact on the risks and efforts imposed by changes in use cases.

Structure of this work: The paper organized as follows: After discussing related work, we present the design of our study and its results. Then, we describe the threats to our study's validity and conclude with a summary and future research directions.

II. Related Work

Change in software has been studied as a software maintenance and evolution issue for many years. The work of Swanson [8] and later Briand et al. [4] built the foundation for further researchers. Chapin et al. [9] propose a comprehensive redefinition of the change types in software maintenance and evolution, and Buckley et al. propose a taxonomy of software change based on characterizing the mechanisms of change and its influencing factors [7]. Moreover Benestad et al. [3] present a systematic literature review on software maintenance research based on analyzing individual changes which gives us a comprehensive picture of this perspective.

In the domain of RE, most researchers agree that also change in software requirements is a constant phenomenon: Harker et al. recommend to consider the characterization of changes and their nature to improve our insight on their impact [15]. Following this view, researchers analyze changes based on different attributes to reach a better characterization of requirement changes.

There are many studies that analyze changes based on addition, deletion, and modification of the software requirement [14, 24, 21, 10]. Most of the studies focus on classifying the reasons (sources) which trigger changes in requirements [14, 21, 22, 10, 18, 20, 19]. McGee and Greer provide a generic change source taxonomy for better managing requirement changes in a series of studies [18, 20, 19]. They distinguish between uncertainty and trigger as reasons for change and introduce five generic sources of change: Market, Customer Organization, Project Vision, Requirement Specification, and Solution.

A few studies go further and distinguish between the reason of a change and its origin [15, 21, 22]. Nurmuliani et al. identify the reasons of a change as rationales behind the proposed changes, such as Design Improvement, and the origin of a change as where it is originated, such as Design Review [21].

Type of Requirement: Some researches classify changes based on the requirement, on which the changes are applied to [14, 24, 15]. Ghosh et al. classify changes based on five requirement categories: Non Functional, two groups of Functional, User Interface, and Deliverable Requirements [14].

Summarized, a wide range of studies analyze different factors of requirement changes. However, our goal is to understand maintainability of the artifact itself. Therefore, we focus on change within the artifacts from a requirements engineer's point of view instead of its reasons and origins, or its effects and consequences later in the software development process.

III. Study Design

To close the aforementioned gap, we designed a case study investigating on the nature of use case changes in practice.

A. Goal and research questions

The goal of this study (formally defined in Table I) is to understand changes in use case artifacts and the modifications that are performed on them. To accomplish the stated goal, we aim at finding out what is changed particularly often (which of use cases and in which part), what these changes are and what types of changes are problematic to requirements engineers.

TABLE I: Research goal

Analyze changes in contents of use case artifacts
with respect to frequency, location, change types, and level of risk
from the point of view of requirements engineers in practice
in the context of a large business information system in maintenance.

From this goal definition, we conduct an exploratory study based on the following research questions:

RQ1. Which use cases change and in which part? To understand the changes in the project, we first analyze the distribution of changes *over use cases* individually: Are there single use cases that change more than others? If so, why?

We furthermore inspect the relation of changes to the *structure* of use cases in general. In consistency with, e.g., [13], we refer to the use case structural elements as content items (e.g., basic flow, preconditions, etc.): Do some content items form hotspots in use cases that are particularly prone to changes? If we understand these hotspots, they might point at certain types of bad maintenance. In addition, for parts of use cases that change particularly often, maintenance might be especially important.

RQ2. Which types of changes exist and occur in use cases? After understanding which use cases change and in which parts, we can then investigate into the contents of the changes, i.e., we need to create a *taxonomy* of these changes. This enables us to understand whether certain *types of changes occur more frequently* than other.

RQ3. Which changes are the most problematic? Lastly, our ultimate goal is to understand and handle problematic changes. We want to provide a first understanding what are problematic changes from a practitioner's perspective and analyze their nature according to the classifications in RQ2. In the long term, this might enable us foresee and prevent some of these problematic changes.

B. Case and Subjects Selection

The case was selected with the goal to study realistic models under realistic conditions, i.e., a real project from industry; Out of these, the selection was performed opportunistically. We invite other researchers to reproduce the study in different contexts.

For the subject selection, we carefully selected those practitioners who performed the analyzed changes and thus worked or work with the use cases.

C. Data Collection and Analysis

We retrieved the use cases of each iteration from the companies versioning system. For this, we extracted the major version of each iteration.

We identified a change as *a block of added, removed or modified words*. We compared the textual contents between the iterations, based on the built-in diff function of Microsoft Word. Even though this adds manual effort for extracting the changes out of Microsoft Word for quantitative statistics, it eases classification, since changes are displayed in the context of changes in the whole use case. When relying on the built-in diff was not technically possible, e.g., due to a difference in layout, we manually compared the text. When use cases changed their name, we identified the corresponding use case and compared them nevertheless. The first version of newly appearing use cases is not counted as a change, but only the following changes, since our study focuses on maintenance.

We filtered graphical changes, layout changes, as well as changes in the document meta data (table of contents, authors, etc.), since we focused on the changes of the textual content in use case artifacts. We manually inspected the changes to filter out single, extreme outliers (e.g., in change size) that would otherwise skew the data.

To answer **RQ1**, we proceeded as follows (more detailed explanation and definitions of the metrics can be found in Table II):

1) Frequency of changes can be understood in three ways: The total number of changes (modified text blocks), $size_{count}$, the number of words changing (often called $churn$ in source code metrics), $total\_churn$, or the number of iterations in which a use cases changes, $size_{iter}$. To identify frequently changing use cases, we aggregate $size_{count}$, $total\_churn$ and $size_{iter}$ per use case.
2) To identify frequently changing content items, we aggregate, for each content item, the average number of changes over all iterations, $avg\_changes$. However, since the content items vary strongly in size (e.g., a list of actors section versus the basic flow section), we also normalize the number of changes by the size of the content item, $relative\_churn$.
3) We discuss the three most frequently changing use cases as well as the most frequently changing content items according to all aforementioned metrics with practitioners, in order to understand why these use cases and content items might change more often.

To answer **RQ2** we proceeded as follows:

4) We perform iterative, open coding on the changes and translate the codes into a taxonomy.
5) We classify all changes according to the created taxonomy.
6) We calculate the $size_{count}$ as well as the $size_{words}$ for each taxonomy item.
7) We discuss the taxonomy, as well as its distribution with practitioners.

To answer **RQ3** we proceeded as follows:

TABLE II: Used Definitions and Metrics

$change$	A block of added, removed or modified words
$changes(uc, it)$	The set of changes for a use case in a given iteration
$size_{count}(change)$	The count-size of a single change is 1
$size_{count}(changes)$	The number of changed text blocks
$size_{words}(change)$	The number of words removed, added or modified within a single change
$size_{words}(changes)$	$\sum_{\forall c \in changes} size_{words}(c)$
$size_{iter}(changes)$	The number of distinct iterations of the changes
$churn(uc, it)$	The number of words changed in a use case in an iteration: $size_{words}(changes(uc, it))$
$total\_churn(uc)$	$\sum_{\forall it \in iterations} churn(uc, it)$
$relative\_churn(ci)$	$\frac{\sum_{\forall it \in iterations} \frac{\sum_{\forall c \in changes(ci,it)} size_{words}(c)}{size_{words}(ci,it)}}{n(iterations)}$
$changes(ci, it)$	The set of changes in a given content item in a given iteration
$avg\_changes(ci))$	The average $size_{count}(changes(ci, it))$ of a content item over all iterations

8) We interview practitioners about problematic changes in use cases from their perspective.
9) We group syntactically similar (*coupled*) changes and calculate dispersion of a group over use cases as well as the dispersion over time (number of different iterations).
10) We compare the largest groups with the categories of RQ2 and discuss results with practitioners.

D. Validity Procedure

To control threats to internal validity due to errors or mistakes, all data analysis and interpretation is executed by at least two researchers, and validated with practitioners involved in the case study.

In order to control the risk that misunderstanding or ambiguity of the taxonomy impacts the distribution for Step 5-7, two researchers independently classify a random subset of 10% of the changes and calculate the inter-rater agreement (Cohen's kappa).

IV. Results

In the following, we report on the results of a first execution of the study at Munich Re.

A. Case and Subjects Description

We performed the study at Munich Re, which is one of the world's leading reinsurance companies with more than 43,000 employees in reinsurance and primary insurance worldwide. For their insurance business, they develop a variety of custom software systems. To elicit the changes of a regular Munich Re project, we inspected the development of a medium-sized industrial software project (around 102,000 SLoC in about 1,900 files), which went live 5 years ago and is thus currently in the maintenance phase. The system forms a complex interface to the Munich Re calculation of life insurance probabilities and conditions. The system behavior is currently described in 32 use cases with a total size of around 35,000 words. The system is furthermore described through supplementary requirements artifacts, such as business rules, business specifications, security

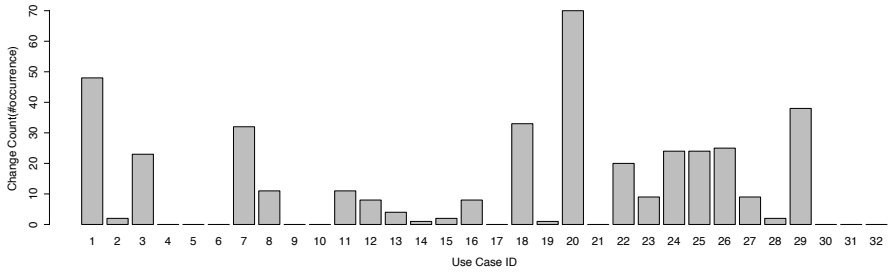


Fig. 1: Total Number of Changes in Use Cases

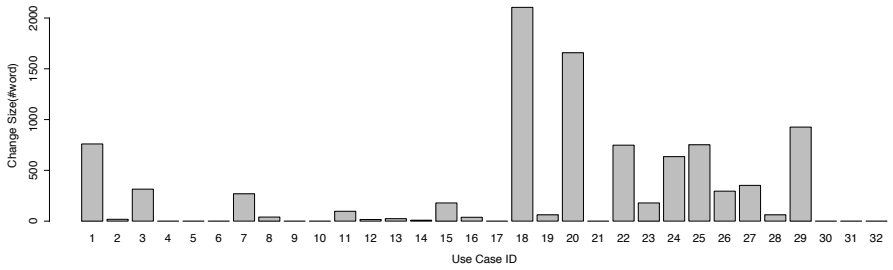


Fig. 2: Total Size of Changes in Use Cases

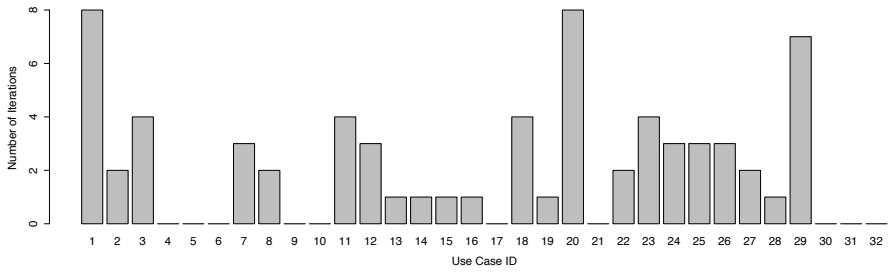


Fig. 3: Number of Iterations Use Case Changed in

plans and guidelines, as well as UI specifications, all of which were not subject of this analysis.

The analyzed use cases were tracked in twelve iterations over the time of 15 months from February 2012 until May 2013. All changes were performed by the RE team of the project. We consider the Munich Re use cases a very common interpretation of the established Cockburn template (e.g. [11]). They contain the following content items: A *brief description* over the contents of the use case, the *actors* and their authorizations in this use case, *pre-* and *post conditions*, as well as the *basic flow* ("main success scenario" in Cockburn [11]) and a set of *alternative flows* ("extensions" or "subvariations" [11]).

In total, our change set included 485 changes. Of these, we filtered 33 changes in the document history, 23 graphical changes, and one outlier, which extracted all business rules into another separated document, and which would have skewed our size metrics. Hence, the following analysis is based on the remaining 405 changes.

Due to non-disclosure agreements and sensible information, in the following, we removed some details of the software system under analysis, focusing on the research questions.

In the following, we answer each research question, by providing our results, explaining practitioners feedback and validation and finally interpreting the results.

RQ1: Which use cases change and where?

We first analyze which use cases change and then analyze the content items changing.

Which use cases change most often? As explained in the study design, the size of a set of changes can be understood in three different ways: Either by taking each change as an atomic unit and simply counting the number of changes for each use case, or by counting the number of words that are

changed within each single change, or, lastly, by counting the number of iterations in which the use cases change.

Figure 1 depicts changes in use cases in terms of count. Use Cases 1, 20 and 29 have the largest number of changes, with between 38 and 70 changes over all iterations. Figure 2 depicts changes in use cases in terms of number of words. Use Cases 18, 20 and 29 have the largest amount of changes with between 920 and 2100 words changed. Lastly, we want to know which use cases were the most volatile over time and changed during larger number of iterations. Figure 3 depicts the number of different iterations in which each use case has changed. The most volatile ones are Use Cases 1 and 20 with changes in 8 iterations and Use Cases 29 with changes in 7 iterations.

Practitioners Feedback: Practitioners explained that these results were not unexpected, since there is an inherent dependency between Use Cases 1, 20, and 29 due to the business workflow. Practitioners told us that this is a structural problem based on the slicing of use cases and, if they would have to write the use cases again, they would put these three use cases into a single use case. Regarding the fact that these use cases changed among 7 or 8 iterations, they told us that this represents Munich Re's process of sometimes incrementally adding functionality. In one case the use cases were changed over two to three iterations until they were implemented into code. The explanation for the large changes in Use Case 18 was a major extension that was executed in this year in an incremental style over 4 iterations.

Interpretation: The analysis revealed three problematic use cases that are constituents of one business flow. In other words, there was one challenging volatile business flow that needed the most attention to take care of. This analysis shows that in our case study the analysis of most changing use cases was indicating at a dependency between use cases.

Which content items of use cases change most often?

In a second part of this research question, we inspect the change within different content items of use cases, measured by the *churn* (see Table II): Figure 4 depicts the average number of changes occurring in each content item per iteration. Absolutely, most changes (avg. of 24 changes per iteration) appear in the basic flow, followed by the alternative flows (avg. of 7 changes per iteration). The other content items hardly change at all.

However, assuming that change is statistically dependent with size, we normalize the change frequency through the size in words, as the *relative churn*, resulting in the distribution shown in Figure 5. This figure shows a different view: The alternative flow is, normalized by the size, changing more than the basic flow.

When inspecting the changes in alternative flows in more depth, we could see that most of changes in this section are adding new flows from scratch or moving concepts from basic flow to alternative flows.

Practitioners Feedback: Practitioners were surprised to see that the alternative flow has a greater relative churn than the basic flow. They saw two reasons for this: First, it could be caused due to common flow restructuring in

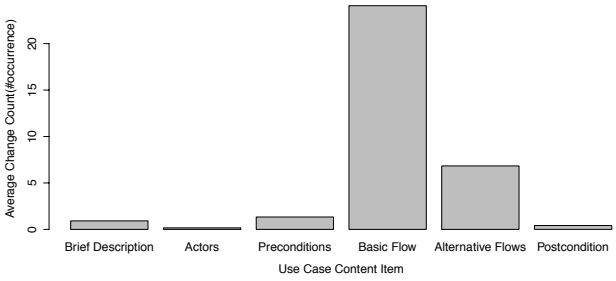


Fig. 4: Average Number of Changes in Use Case Content Items per Iteration

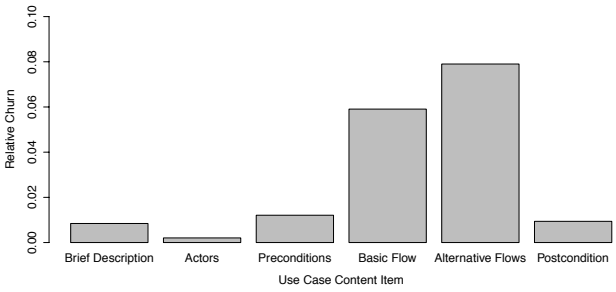


Fig. 5: Change Probability of Use Case Content Item per Word

alternative flows. Second, since this is a project with a long life span, new requirements often are extensions to existing use cases. This could lead to the fact that new requirements manifest themselves as new alternative flows. Regarding the non-changing content items, they agreed, since changes in actors (including authorizations) for such a mature project is very rare.

Interpretation: In this case study, the basic flow is changed most often. However, looking at the content per word, the alternative flow has a higher change rate. Based on the feedback from practitioners we interpret the data as follows: Alternative flows of a use case are not completely clear at the beginning; there are alternative flows which initially cannot be recognized neither by customer nor by requirement engineers and it takes time until they get exposed. Another reason for high relative churn in alternative flows can be lack of attention on alternative flows at first, caused by the fact that basic flow matters more than alternative flows.

In summary, we interpret these results as showing, that the maintenance of basic and alternative flows should be watched carefully, since these content items will change more often over time. This implies for practitioners, that proper slicing of use cases early on can potentially save effort later in the process.

RQ2: Which types of changes exist and occur?

To answer this research question, we developed a taxonomy of use case changes. In contrast to related work, we focused

on concrete changes in use cases without considering the high level reason or source of a change.

Which types of changes exist? In order to create a taxonomy that adequately represents the changes under analysis, and not limit ourselves with any constraints or assumptions, we applied a bottom-up approach, i.e., an open classification of a subset of changes. We iteratively extended and groomed this taxonomy, until we resulted in the categories given in Table III. In the following we will not explain each category in detail, but the structure and motivation of the categories.

The taxonomy is based on the differentiation between semantic and syntactic changes: When a semantic change is executed, the system that is described in the use case changes. This implies, that semantic changes necessarily lead to subsequent changes in other artifacts. In contrast, syntactic changes only change how the system is described in the use cases. Syntactic changes are similar to refactorings, as known from source code.[1]

All semantic changes are, from an RE perspective, to add, modify or remove requirements. All differentiation here would lead to an analysis of the reasons for use case changes, which is out of scope of this research. In contrast, the syntactic changes scatter over various types of use case refactorings. Please refer to Table III for more details.

Practitioners Feedback: The taxonomy was successfully validated with practitioners: Practitioners were not aware of further categories, and advised against removing any of the introduced categories.

Which types of changes occur? To understand which changes occur, we made a closed classification of all 405 changes, applying the bottom-up taxonomy discussed before. Figure 6 shows the total number of changes in each category. `Adding Requirement`, `Modifying Requirement`, and `Clarifying Taxonomy` have the largest number of changes among all change categories respectively. They are followed by `Flow Management`, `UI`, and `Sentence Enhancement`.

Figure 7 depicts a box plot for change sizes in each category, ordered by total size in terms of words of changes in each category. `Adding Requirement`, `Modifying Requirement`, and `Flow Management` have the largest total size of change, while `Document Management`, `Flow Management`, and `Adding Requirement` have the greatest mean size of change respectively. The results show that after `Adding Requirement` and `Modifying Requirement`, `Flow Management` changes were the most common on use case documents. Furthermore striking is the large number of `Clarifying Taxonomy` changes occurred, all of which had very small sizes.

[1]We use the terms 'semantic' and 'syntactic' here in the context of system behavior, in contrast to the semantics of natural language. Obviously, nearly every addition, modification or deletion of words changes the semantics of the natural language text, but not each change also changes the semantics of the described system. For example, if we clarify the meaning of a term appearing in the use case, we still describe the same system behavior, but with more precise language (thus the semantics of the text changes).

Practitioners Feedback: Practitioners were surprised by the high number of `Adding Requirement` and `Modifying Requirement` changes. They were unsure whether this was typical for their projects. They expected the high number of `Clarifying Taxonomy` changes, due to major changes that we will describe in detail in the RQ3.

Interpretation: In our case study, `Adding Requirement` and `Modifying Requirement`, as well as `Clarifying Taxonomy` are the most common change types. Future work needs to investigate whether this is also common for other projects or a case specific result.

How are semantic and syntactic changes distributed? Besides the detailed analysis of each category by itself, we also analyze how changes are distributed over the categories of semantic and syntactic changes. To identify semantic and syntactic changes we used the classification based on our detailed categories of changes, in which we classify `Adding Requirement`, `Modifying Requirement`, or `Removing Requirement` as semantic changes and other categories as syntactic changes.

TABLE IV: Syntactic vs Semantic Changes

	Semantic	Syntactic
$size_{count}$	47%	53%
$size_{words}$	71%	29%

The results in Table IV show that 53 percent of all changes are syntactic changes and 47 percent are semantics. When taking the size in words into account 29 percent of the total size of all changes are syntactic and 71 percent are semantic. Hence, semantic changes are larger than syntactic changes (avg. size of 13 words for syntactic changes vs. avg. size of 35 words for semantic changes).

In more depth, Figure 8 and Figure 9 show the count and size percentage of semantic and syntactic changes in use case content items respectively. Actors had only syntactic changes, however only few changes occurred in this content item. Actors, Preconditions, and Postconditions are more prone to syntactic changes than other content items. Figure 11 depicts that the Brief Description had been changed mostly because of a semantic change in use cases. The relation between semantic and syntactic changes in Basic Flow and Alternative Flows represents roughly average distribution (see Table IV).

Figure 10 shows the distribution of semantic vs syntactic changes over all 12 iterations. We consider Iteration 7 an outlier, since it only contained 4 changes. Iteration 1, 5 and 6 show the highest percentage of syntactic changes, with around 60%. Other iterations show around 20% to 40% of syntactic changes.

Practitioners Feedback: Practitioners agreed with classifying changes into semantic and syntactic and immediately had an intuitive understanding. However, from their practical standpoint, they could not relate to the relative numbers from Tbl. IV, since the size and count of changes does not necessarily represent effort spent. Future work should look deeper into a change-effort relation for RE. However, they considered inspecting the trend of the relation over time reasonable: In

TABLE III: Use Case Change Taxonomy

Semantic Changes	Adding Requirement	Adding a text which adds something new to the system	"System sets the validation status to invalid"
	Modifying Requirement	Adding or modifying a text which leads to a change in the system	"five different" to "the maximum number is 12"
	Removing Requirement	Removing a text in order to delete a part of or even a complete requirement	"In this case the latter action will be performed without changing the status"
Syntactic Changes	Flow Management	Reorganizing the basic or alternative flows of use case in which the actions are taking place	Dividing, merging, moving a part or whole of a step
	Document Management	Extracting a text or figures into another document or vice versa	Extracting GUI figures to another document.
	Sentence Enhancement	Enhancing the structure of a sentence to make it more clear without changing its meaning	"the permitted" to "those that the user is authorized for"
	Typos	A typographical error	"fore" to "for"
	Clarifying Taxonomy	Adding or modifying some words in a phrase to distinguish it from a changed or new concept in a clearer way. If this rephrasing also changes a requirement it is classified as modifying requirement	"business" to "entity X"
	Formatting	Changing the presentation style of a requirement by adding, modifying or removing text	Representing some rules in a table instead of lines of text.
	Adding Supplementary Reference	Adding a reference to a context to make it more clear or connect it to other required documents	"(see BR_XX)"
	Updating Reference	Updating a reference to new version	"(apply UC_BRXX)" to "(apply LRXX_BRXX)"
	Adding Details	Adding a text to make something more clear without adding or changing the context of a requirement	"both fields" to "the text field and the drop down box"
	Removing Useless	Removing a useless or outdated text	Deletion of an outdated version of a rule
	UI	Adding, modifying, or removing a text or figure which exclusively speaks on user interface	"If an element has the status X is highlighted" design and features

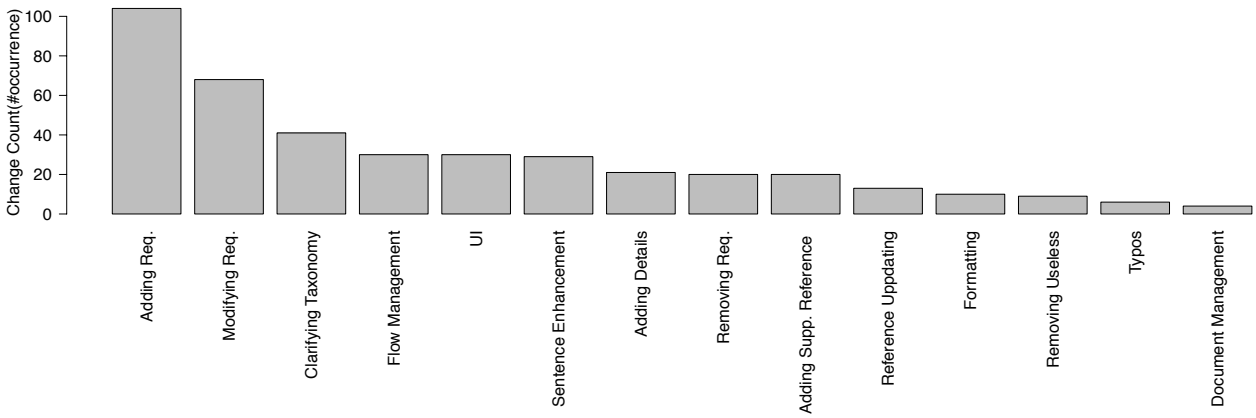


Fig. 6: Total Number of Changes per Category

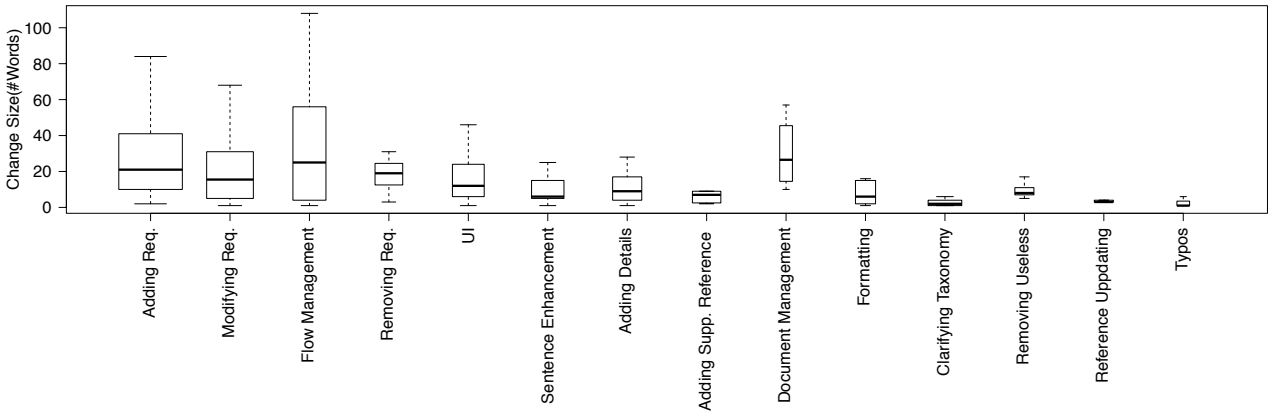


Fig. 7: Word-Size of Changes per Category, Ordered by Total Size, Width∼Sample Size

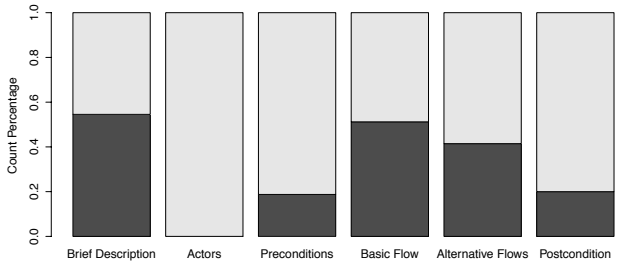


Fig. 8: Syntactic (Light Gray) VS Semantic (Dark Gray) Change Size Count Percentage in Content Items

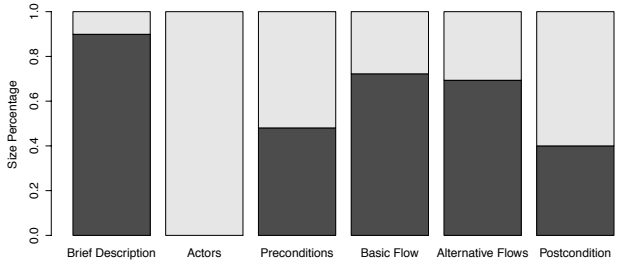


Fig. 9: Syntactic (Light Gray) VS Semantic (Dark Gray) Change Size Word Percentage in Content Items

the first sprints, requirements engineers had time to deal with syntactic refactorings in the use cases. The same holds for Iteration 6.

Interpretation: This case study showed a proportion of 53% of syntactic changes on RE artifacts, which, due to their smaller size, account for 29% of the size of changes. For source code, studies show that, e.g., after reviews only around 25% (e.g. [1]) of changes are semantic changes. Even though this provides us with a rough benchmark, unfortunately, due to the high granularity of iterations in our case study, we cannot detect which changes were triggered through reviews. Future work should reproduce our results with review changes.

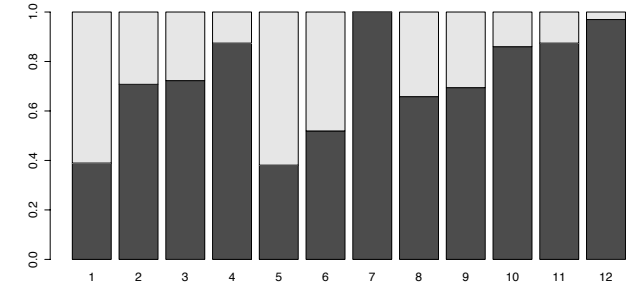


Fig. 10: Syntactic (Light Gray) VS Semantic (Dark Gray) Changes Size Word Over Iterations

TABLE V: Five largest change groups

		#Occurrence	# Iterations	# Use Cases
Group 1	Clarifying Taxonomy	18	3	6
	Modifying Requirement	1		
	Removing Requirement	1		
Group 2	Clarifying Taxonomy	3	3	4
	UI	9		
Group 3	Clarifying Taxonomy	2	1	1
	Modifying Requirement	5		
	Adding Requirement	5		
Group 4	Modifying Requirement	1	1	2
	Adding Requirement	5		
Group 5	Clarifying Taxonomy	7	1	1

In our study preconditions and postconditions showed a higher proportion of syntactic changes. It remained unclear where this difference results from.

We could furthermore observe that the percentage of syntactic changes reflected rather calm periods in the project. We imagine that tracking the distribution of syntactic vs semantic changes could indicate for some sort of requirements technical debt (cf. e.g. [6]), indicating periods of pressure and refactoring in the project.

RQ3: What are problematic types of changes?

In our last research question, we wanted to understand which changes are difficult to conduct within the requirements artifact[2].

What are problematic types of requirements artifact changes from an industry perspective? In order to receive open, unrestricted feedback, we openly asked the practitioners what they considered problematic changes for use cases. This resulted in two types of problematic changes:

The first group of problematic changes are semantic changes that are inherently complex (*essential complexity* in the words of Brooks [5]), due to the nature of the domain. This is especially common in the insurance domain, since the systems build on a complex mathematical and legal background.

The second group of problematic changes are changes that are not inherently complex, but particularly risky, since there is a dependency between multiple changes. We would consider these changes *accidentally complex*, since the complexity results from the RE instead of the domain. The practitioners took our taxonomy as reference and identified `Clarifying Taxonomy`, `Flow Management`, `Adding Details` or also locally dispersed semantic changes as particularly risky, since these changes might have a higher probability of introducing inconsistencies or unintentionally introduce other incorrect behavior (c.f. the risk of dispersed software clones leading to inconsistencies in code [17]).

[2]We only consider the maintainability of the requirements artifact itself, not the impact that the change triggers. This has been studied by previous works, cf. Section II.

Can we find problematic changes through syntactically coupled changes? In order to detect risky changes from dependencies, we searched for *locally dispersed changes*, i.e., changes that are syntactically similar (or *coupled*), but appear in different locations, either within a document, between documents, or even over time (*temporally dispersed*). Syntactically coupled changes are a group of changes which can be recognized explicitly by their terms and phrases as the same change in different spots in use cases and iterations. Table V shows the five largest change groups in terms of highest number of occurrences, including their change types and number of occurrence, number of dispersed iterations and use cases. In the following, we provide details and explain practitioners feedback on the rationale behind each group.

Group 1 was a fundamental taxonomy change due to a change in the requirements: At first, the whole system was built around a single type of entity, the *model point*, which is added, changed, calculated etc. However, at a certain iteration in the middle of the project, the system needed to be changed so that it could handle two different types of model points. This lead to fundamental changes over 6 use cases, which needed three iterations until finished (one change was conducted several iterations later, a fact that could hint at an inconsistent/forgotten change). This group showed a risky temporally and locally dispersed change.

Group 2 contained a set of changes in the taxonomy, which lead to changes in multiple use cases and iterations. The problem here was that the use cases also contained UI references, which needed to be updated subsequently. Practitioners told us that they considered this bad practice and, if possible, would move UI design specification to a separate artifact. This indicated bad requirements artifact maintainability resulting from UI details in use cases.

Group 3 was a set of reference to an enumeration that needed to be continuously updated when a new item is added to the list, e.g., B1, B2, etc. They stated that these types of numbered references are very hard to maintain and can easily lead to wrong references.

Group 4 was an essentially complex requirements change from the business domain.

Group 5 was a clarification of taxonomy that was always implicitly clear for insiders, but became obvious when new people joined the team.

Interpretation: 49% of changes in the top five change groups were in the category `Clarifying Taxonomy`. It's an inherent property of Taxonomy Changes to spread among different use cases, since terms are usually used orthogonally through all use cases. We interpret these change groups as advice to have the terms clear at the start of writing use cases, since changes in taxonomy in late phases causes dispersed and thus potentially problematic changes. Furthermore, we found evidence that UI details as well as enumerated references can cause dispersed changes and thus decrease the maintainability of use cases.

V. Threats to Validity

Regarding our answers to the research questions, two major threats could constrain our internal validity: First, elaborating our change taxonomy was a creative process, hence, it could be ambiguous, incorrect or incomplete. We analyzed the ambiguity through independent reclassification of a subset of 10% of the changes, leading to a substantial inter-rater agreement (Cohan's kappa: 0.65). We therefore consider this threat negligible. Second, our change taxonomy could also be incorrect (internal validity) or incomplete (external validity). We control these threats, as well as threats regarding potential bias in interpretation of practitioner's feedback, through validation with practitioners.

Regarding external validity, case study research inherently comes with advantages, but limited generalizability, since it always answers research questions for a limited set of cases [23]. Our study intentionally focused on an industrial project in the maintenance phase, hence, the results might not be generalizable for requirements artifacts in elaboration. Furthermore, we intentionally analyzed changes per iteration instead of more fine-grained changes. This could also create a different picture, such as more typos. Lastly, our results show that changes depend on the maintainability of the use cases. Therefore, we are expecting to see different results for use cases in different quality.

In addition, our study intentionally focused on expert opinions, which can only provide some facets of (bad) maintainability. Thus, we invite other researchers to reproduce this case study in order to confirm or refute our observations and extend the validity onto other change granularities, project settings, and maintainability facets.

VI. Conclusion and Outlook

Changes in requirements artifacts are common in software projects, since outdated artifacts are not useful to stakeholders. However, there is little existing knowledge on maintainability of use cases.

This paper presents an analysis of use case changes based on a case study in an industrial software project in maintenance. Applying qualitative and quantitative methods to more than 400 changes and discussing the results with practitioners, we answer our research questions in this case as following:

RQ1: Which use cases change and where? Our analysis revealed that the most frequently changing use cases in our case, are in a strong dependency with each other (belonging to the same workflow). We also found that although most of changes occur in the basic flow, alternative flows are most prone to change relative to their size. We observe that improper slicing of use cases forms one way of bad maintainability and that most maintenance changes go into alternative and basic flows. This indicates that these two content items have a stronger need for use case maintainability.

RQ2: Which types of changes exist and occur? We developed a detailed change taxonomy for use cases, with which we found out 50% number of all changes and 30% of the total size of all changes are syntactic. Over time we

see that phases with higher proportion of syntactic changes coincide with rather calm periods in the project. We conclude that tracking the proportion of syntactic and semantic changes over time can indicate the effort going into quality assurance of use cases. Therefore, future work should analyze the potential of this metric for project and QA monitoring.

RQ3: What are problematic types of changes? Practitioners report that problematic changes origin from essential complexity, i.e., complexity in the domain, and accidental complexity, i.e., complexity in the requirements artifacts themselves. For the latter, we identified dispersed changes as particularly risky. This study has shown the particular difficulty of changes in the domain taxonomy. We furthermore identified UI details and improper referencing as other causes for risky, dispersed changes. This motivates to monitor dispersed changes, but also provides first empirical evidence towards factors for bad maintainability, such as UI details or improper references.

Future work: We found these factors for bad maintainability in our case study. Future work should dig deeper This study was performed on major versions of use cases. An analysis on minor versions of use cases can gives us deeper insight on the way use cases change, including which changes are triggered by use case reviews in particular.

ACKNOWLEDGMENTS

This work was performed within the project Q-Effekt; it was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS15003 A-B. The authors assume responsibility for the content.

REFERENCES

[1] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. Modern code reviews in open-source projects: which problems do they fix? In *MSR*, 2014.

[2] E. Ben Charrada, A. Koziolek, and M. Glinz. Identifying outdated requirements based on source code changes. In *RE*, 2012.

[3] Hans Christian Benestad, Bente, and Erik Arisholm. Understanding software maintenance and evolution by analyzing individual changes: a literature review. *Journal of Software Maintenance and Evolution: Research and Practice*, 2009.

[4] Lionel C. Briand, Victor R. Basili, and Yong-Mi Kim. A change analysis process to characterize software maintenance projects. In *ICSM*, 1994.

[5] Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 1987.

[6] Nanette Brown et al. Managing technical debt in software-reliant systems. In *FSE/SDP workshop on Future of software engineering research*, 2010.

[7] Jim Buckley, Tom Mens, Matthias Zenger, Rashid Awais, and Gúnter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 2005.

[8] Swanson E. Burton. The dimensions of maintenance. In *ICSE*, 1976.

[9] Ned Chapin, Joanne E Hale, Khaled Md Khan, Juan F Ramil, and Wui-Gee Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance: Research and Practice*, 2001.

[10] Bee Bee Chua and June Verner. Examining requirements change rework effort: A study. *International Journal of Software Engineering & Applications*, 2010.

[11] Alistair Cockburn. Basic use case template. *Humans and Technology, Technical Report*, 1998.

[12] Henning Femmer, Jakob Mund, and Daniel Méndez Fernández. It's the Activities, Stupid! A New Perspective on RE Quality. In *RET workshop at ICSE*, 2015.

[13] Daniel Méndez Fernández, Birgit Penzenstadler, Marco Kuhrmann, and Manfred Broy. A meta model for artefact-orientation: fundamentals and lessons learned in requirements engineering. In *Model Driven Engineering Languages and Systems*. 2010.

[14] Sanjay Ghosh, Srini Ramaswamy, and Raoul Praful Jetley. Towards requirements change decision support. In *APSEC*, 2013.

[15] S.D.P. Harker, K.D. Eason, and J.E. Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *RE*, 1993.

[16] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Reading, 1999.

[17] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *ICSE*, 2009.

[18] Sharon McGee and Des Greer. A software requirements change source taxonomy. In *ICSEA*, 2009.

[19] Sharon McGee and Des Greer. Sources of software requirements change from the perspectives of development and maintenance. *International Journal on Advances in Software*, 2010.

[20] Sharon McGee and Des Greer. Software requirements change taxonomy: Evaluation by case study. In *RE*, 2011.

[21] Nur Nurmuliani, Didar Zowghi, and Sue Fowell. Analysis of requirements volatility during software development life cycle. In *ASWEC*, 2004.

[22] Nur Nurmuliani, Didar Zowghi, and Susan P. Williams. Requirements volatility and its impact on change effort: Evidence-based research in software development projects. In *ASWEC*, 2006.

[23] P. Runeson and M. Höst. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *EMSE*, 2009.

[24] George E. Stark, Paul Oman, Alan Skillicorn, and Alan Ameele. An examination of the effects of requirements changes on software maintenance releases. *Journal of Software Maintenance: Research and Practice*, 1999.

Publication D: On The Impact of Passive Voice Requirements on Domain Modelling

Authors Henning Femmer, Jan Kučera, and Antonio Vetrò

Venue 8th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2014

Abstract [Context] The requirements specification is a central artefact in the software engineering (SE) process, and its quality (might) influence downstream activities like implementation or testing. One quality defect that is often mentioned in standards is the use of passive voice. However, the consequences of this defect are still unclear. [Goal] We need to understand whether the use of passive voice in requirements has an influence on other activities in SE. In this work we focus on domain modelling. [Method] We designed an experiment, in which we ask students to draw a domain model from a given set of requirements written in active or passive voice. We compared the completeness of the resulting domain model by counting the number of missing actors, domain objects and their associations with respect to a specified solution. Results: While we could not see a difference in the number of missing actors and objects, participants which received passive sentences missed almost twice the associations. [Conclusion] Our experiment indicates that, against common knowledge, actors and objects in a requirement can often be understood from the context. However, the study also shows that passive sentences complicate understanding how certain domain concepts are interconnected.

Extended Summary This paper is summarized in Section 4.2.3.3.

Authors Contributions I co-designed the experiment, co-performed the analysis, and co-reported the results.

On The Impact of Passive Voice Requirements on Domain Modelling

Henning Femmer
Technische Universität
München, Germany
femmer@in.tum.de

Jan Kučera
Technische Universität
München, Germany
kucera@in.tum.de

Antonio Vetrò
Technische Universität
München, Germany
vetro@in.tum.de

ABSTRACT

Context: The requirements specification is a central artefact in the software engineering (SE) process, and its quality (might) influence downstream activities like implementation or testing. One quality defect that is often mentioned in standards is the use of passive voice. However, the consequences of this defect are still unclear. **Goal:** We need to understand whether the use of passive voice in requirements has an influence on other activities in SE. In this work we focus on domain modelling. **Method:** We designed an experiment, in which we ask students to draw a domain model from a given set of requirements written in active or passive voice. We compared the completeness of the resulting domain model by counting the number of missing actors, domain objects and their associations with respect to a specified solution. **Results:** While we could not see a difference in the number of missing actors and objects, participants which received passive sentences missed almost twice the associations. **Conclusion:** Our experiment indicates that, against common knowledge, actors and objects in a requirement can often be understood from the context. However, the study also shows that passive sentences complicate understanding how certain domain concepts are interconnected.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specification

General Terms

Requirements Engineering, Quality Assurance, Natural Language

Keywords

Requirements Engineering, Analytical Quality Assurance, Experimentation

ESEM'14 September 18-19, 2014, Torino, Italy.

1. INTRODUCTION

During software development, the artefacts that are produced by Requirements Engineering (RE), such as use cases [8], are usually the central items for communication of stakeholders' needs. Based on these artefacts, developers write source code, testers design test cases and finally the customer accepts or rejects the product.

This central role of RE artefacts suggests that their constitution, i.e. their quality, is important for the rest of the process. This, together with the fact that producing RE artefacts is rarely the goal of the project, implies that we need to understand this notion of requirements quality from a quality-in-use perspective on the software development lifecycle. Hence, the central questions is: Which property of the RE artefact has what kind of influence on activities performed with the artefact?

Such requirements properties are often proposed by common standards (e.g. as *Requirement Language Criteria* in the most relevant RE standard ISO29148 [7], cf. [14]). One of the properties that is often proposed, is the use of passive voice in requirements. The argumentation is usually that in passive sentences it is unclear who is performing a certain action on the system (e.g. [10]). However, when we look at specifications in practice, we nearly always see violations of this property (for examples see [1, 3, 4, 11]). We argue that this is not only because of the possible lack of distribution of the standard in practice, but more due to the fact that we have not yet understood the real impact of passive sentences in textual RE specifications onto downstream SE activities.

An analysis on where textual requirements are used in SE is performed in [8]. There, one of the first steps is to create a domain model that describes the concepts on which a system is working and their interrelations. This domain model, no matter whether it is build explicitly or implicitly, forms the common understanding of the concepts that are used by the involved stakeholders throughout the further system development process. We focus this work on the impact of passive voice requirements on domain modelling for these two reasons, i.e. 1) proximity of the domain modelling activity to requirements specification and 2) usage of domain models on later phases of software development.

2. STUDY

To understand the impact of passive voice in requirements on domain modelling, we must first define requirements and domain models in our context and then identify the activities that are needed to create a domain model.

2.1 Object of study: Requirements

In the following, we understand a *requirement* as a single, textual sentence expressing a stakeholder's need. This is a common format used in many domains and mirrored in common requirements formats such as *ReqIF*[1] and tools such as *DOORS*[2]. One example of a passive voice is from [4]:

Example 2.1. Dependent on which turn signal is set, the arrow showing in the same direction lights up blinking, as long as the turn signal is set.

A version of this example in active voice would be:

Example 2.2. The system shall light up blinking of the arrow showing in the same direction, dependent on which turn the driver sets the turn signal.

2.2 Object of study: Domain model

As a *domain model* we understand a set of domain concepts plus relations between these concepts. A domain model for this requirement is the one represented in Figure 1.

Figure 1: Domain model for the example

2.3 Activites for domain modelling

In order to evaluate the domain models that experiment participants produce, we have to detail how the domain modeling is performed. For this, we took the Unified Process (UP) [8] as a reference to find the detailed subactivities that are part of domain modelling. The advantage of using the UP is that this process provides a comprehensive list of artefacts and corresponding activities: however, most of these activities are not specific to the UP, as they are performed in many forms of processes – either explicitly or implicitly.

Accordingly, we identified the following three subactivities where the usage of passive sentences in requirement specifications could potentially have an impact:

1. Find Actors
2. Identify Domain Objects
3. Identify Associations and Aggregations

For the third subactivity, we focused only on associations, because we wanted to keep the study design simple. Furthermore, aggregations can be interpreted as a special case of associations.

2.4 Goal and research questions

The goal of this study (formally defined in Table 1) is to understand whether using passive sentences in requirements has a negative impact on domain modelling activities.

From the goal definition we derive our research question:

RQ1 Is the use of passive sentences in requirements harmful for domain modelling?

The question can be broken down in three subquestions:

RQ1.1 Is the use of passive sentences in requirements harmful for finding actors?

RQ1.2 Is the use of passive sentences in requirements harmful for identifying domain objects?

RQ1.3 Is the use of passive sentences in requirements harmful for identifying associations?

[1] `http://www.omg.org/spec/ReqIF`

[2] `http://www-03.ibm.com/software/products/en/ratidoor`

Table 1: Research goal

Characterize the impact of passives sentences in requirements on domain modelling
with respect to the quality of the artefacts produced from the activities *Find Actors*, *Identify Obvious domain objects*, *Identify Associations*
from the point of view of the software developer (or business or requirements analyst)
in the context of an analysis of requirements from real projects by graduate and undergraduate students in Computer Science

2.5 Design, methodology and metrics

We designed an experiment for university students. Each participant received a set of seven requirements: for one group (P) all requirements were in the original, passive form (requirements are passive examples that we found in the specifications [4] and [11]), for the other group (A) the same requirements were transposed into the corresponding active form (see Table 2). The participants were randomly assigned to one of the two groups of requirements (P,A). They had to perform three activities for modelling the domain: identify the actors, domain objects and associations. Each activity that the participant should perform was previously instrumented by reading material and an example of a solution[3]. To verify our thesis, we observed the artefacts produced by those three activities and compared the artefacts produced in the two groups, by counting the number of *missing actors*, number of *missing domain objects* and number of *missing associations* with respect to the master solution.

i) Number of missing actors: We counted all actors that were identified in the master solution and not recognized by the participants. Additional actors were ignored since missing actors is a more serious error than superfluous ones. Also, we carefully looked for synonyms (e.g., if the master solution identified the actor "realtor", we also accepted "real-estate agent" or even "user" (if there is a clear distinction to other actors).

ii) Number of missing domain objects: The number of missing domain objects were counted similarly to the number of missing actors (i.e. synonyms are accepted). We did not count as a mistake when a participant identified an actor in the first activity and did not write down the actor name when required to draw or list the domain objects. We assumed that the subject correctly identified that actor as an object in this case.

iii) Number of missing associations: We evaluated whether the associations connect to correct domain objects. Although we asked the participants to include also the directions and names of the associations, we didn't evaluate them: these descriptions rather served as a point of assurance for the evaluator that the participant understood the task correctly.

With N_A and N_P being the number of missing actors, missing domain objects, and missing associations respectively in active sentences (A) and passive ones (P), our research questions translate in the following pair of null (1)

[3] All experiment data is available for checks and replication at `http://goo.gl/WlTPE5`

Table 2: Requirements Used in Experiment

ID	Passive voice requirement	Translated into active voice
1	The search results shall be returned no later 30 seconds after the user has entered the search criteria.	The system shall be capable of returning the search results latest 30 seconds after the user has entered the search criteria.
2	The CMA report shall be returned no later 60 seconds after the user has entered the CMA report criteria.	The system shall be capable of returning the CMA report latest 60 seconds after the user has entered the CMA report criteria.
3	The realtor shall be notified of new client appointments after automatic synchronization with office system.	The system shall notify the realtor of new client appointments after automatic synchronization with the office system.
4	All transaction details shall be obtained from the Statement Database.	The system shall obtain all transaction details from the Statement Database.
5	All additions of new users shall be recorded on the User Report.	The system shall record all additions of new users on the User Report.
6	The reliability of the indicator lights and the engine control light shall be tested, whenever the instrument cluster is activated.	The system shall test the reliability of the indicator lights and the engine control light, whenever the system activates the instrument cluster.
7	Dependent on which turn signal is set, the arrow showing in the same direction lights up blinking, as long as the turn signal is set.	The system shall light up blinking of the arrow showing in the same direction, dependent on which turn the driver set the turn signal.

and alternative (2) hypotheses (one pair for each of the three activities):

$$H_0 : N_P \leq N_A \quad (1)$$

$$H_A : N_P > N_A \quad (2)$$

The Mann-Whitney test with 95% confidence interval was used to test the three null hypotheses.

2.6 Participants selection

The subjects of the experiment are B.Sc., M.Sc. and Ph.D. students from Technische Universität München (see Table 3). To reduce the threat of participation of students with insufficient knowledge, we removed the 2 participants who achieved less than 70% correct answers in a short test in the field of RE, extracted from [12]. We furthermore excluded one participant who misunderstood the task and one participant who did not finish the experiment.

Table 3: Participants (final number in brackets)

Student's degree	A	P	Sum
B.Sc.	4(2)	1(0)	5(2)
M.Sc.	3(3)	5(5)	8(8)
PhD	3(2)	2(2)	5(4)
unknown	0(0)	1(1)	1(1)
Sum	10(7)	9(8)	19(15)

3. THREATS TO VALIDITY

We report herein the validity issues of our experiment, according to the traditional classification [15]. Regarding internal threats, we recognise the risk that the activities are so simple that no impact could occur in the experiment even though there is some impact in the reality. To reduce this threat we selected seven requirements from real industrial projects. Second, the students in our context had very limited context information. However, this applies for participants with both active and passive voices and thus should not impact the outcome per-se. Nevertheless, it remains open whether the same effect can be observed in practice. We observe a conclusion threat as well: since the activities in the experiment are from an engineering field, there is also a high risk that each participant provides a different solution. We controlled this threat by providing comprehensive instrumentation and defining tolerant criteria for correctness (see Section 2.5).

Finally we report external threats. The first one relates to the generalizability of results to industrial practices, since participants of the experiment were students. The use of students as study subjects has been longly discussed in the software engineering literature and in general considered as suitable under certain conditions, based on generally accepted criteria for validity evaluation of empirical studies [6]. Runeson [13] observed that graduate level students are feasible subjects for revealing improvement trends: since the focus in our study was a comparison and not to find an absolute level of improvements, according to [13] the use of students in our case is suitable. In addition, a study on requirements prioritisation [2] showed that experience and commitment are important factors when using students as study subjects: we balanced this threat by letting the participation of students voluntary and by ensuring an adequate level of experience and skills of participants. Finally, we used requirements from real industrial projects.

4. RESULTS

The box plot in Figure 2 gives an insight on the results and a comparison between active and passive voices requirements. In addition, Table 4 reports the results of the non parametric Mann-Whitney test with 95% confidence interval for the difference between the two groups, and Cliffs' δ as non parametric standardised effect size measure. It was not possible to reject the null hypotheses for finding actors and identifying domain objects. On the contrary, the null hypothesis on identifying associations was rejected in favor of its alternative, i.e. the number of missing associations is higher in requirements with passive sentences (effect estimate: 75% of the time).

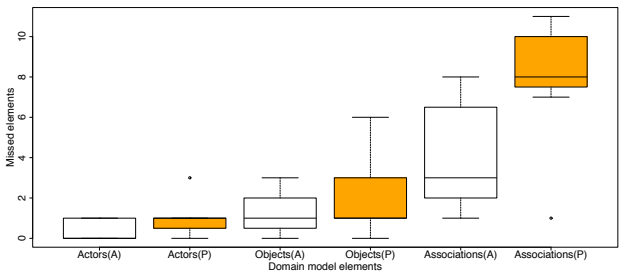


Figure 2: Number of errors for requirements in (A)ctive and (P)assive voice

5. DISCUSSION

First, the participants were able to identify actors in both active and passive voices. However, there was one difference in Requirement 3 (slightly better performance of the passives) and Requirement 7 (heavily worse performance of the passives). For the latter, the participants were unable to

Table 4: Results and descriptive statistics

	Activity	Mean A.	Mean P.	Median A.	Median P.	P value	Conf. Int.	Cliff's δ
missed	actors	0.43	1.00	0	1	0.10	$(0;\infty)$	0.39
	objects	1.29	2.00	1	1	0.25	$(-1;\infty)$	0.25
	assoc.	4.14	7.88	3	8	0.02	$(1;\infty)$	0.75

decide that the "driver" of a car was the main actor. We interpret this as follows: The role of the context is important to find the actor when it is not explicitly present; hence, it may lead to misunderstandings, but in our experiments this was not the case in nearly all examples. While in real situations we assume that more context is present and thus the difference would be even less, we have to analyse the different forms of passives in more depth to understand.

Second, for the identification of domain concepts, the results are not really different between the two groups. One could argue that it is possible to identify all relevant domain objects from the text, regardless of the form.

Last, the identification of associations shows a statistically significant difference. We see three different reasons for this: Either, this difference only shows up as significant because the activity is harder. In fact, the average number of errors is higher. But, this is the same for both active and passive voice. Or, the reason is that requirements in passive voice contain less information. But this would contradict the fact that there is no significant difference between the active and passive during identification of objects. Hence, we argue that, even though the information about the existing concepts is there, the passive complicates understanding how these concepts are linked. An additional observation supports this hypothesis: 3 out of the 9 participants in the group of passive voices (P) were not even able to draw relations, even though they quite correctly identified actors and objects.

6. RELATED WORK

There are various approaches that aim at detecting passive voice in requirements (as one form of ambiguity), with the assumption of bad impacts (e.g. [5] or [9]). Other approaches such as [10] go even further and aim at compensating the missing actors by deducing them from the context.

However, we are not aware of any study that focuses on understanding the impact of this property of requirements onto activities of the software engineering lifecycle.

7. CONCLUSION AND FUTURE WORK

In this paper, we described an experiment conducted to characterise the impact of requirements in passive voice onto domain modelling, as one activity of the SE lifecycle.

The results indicate that while the commonly discussed danger of missing actors did not show to be substantial, there is a statistically significant gap in the understanding of how concepts are related in a sentence in passive voice.

We provided a link to the experiment instrumentation for the sake of exact replication of this study. Additionally, we especially need replications with other requirements to analyse whether there is a difference in outcome when using different forms of passive voice.

On a methodical level, we are working on an evidence-based approach towards understanding quality for requirements engineering in an activity-based manner. This study is one step indicating that experiments can provide such evidence on property-impact-activity relationships.

Acknowledgments

We would like to thank Maximilian Junker, Daniel Méndez Fernández and Andreas Vogelsang for their reviews.

8. REFERENCES

[1] Canal monitoring and control system. Technical report, MoDRE, 2011.

[2] P. Berander. Using students as subjects in requirements prioritization. In *ISESE*, Aug 2004.

[3] Boston Scientific. Pacemaker system specification. Technical report, 2007.

[4] K. Buhr, N. Heumesser, F. Houdek, H. Omasreiter, F. Rothermel, R. Tavakoli, and Z. T. DaimlerChrysler demonstrator: System specification. Technical report, EMPRESS Project, 2003.

[5] G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado, and V. Moreno. A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 2011.

[6] M. Höst, B. Regnell, and C. Wohlin. Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. *Empirical Softw. Eng.*, 2000.

[7] ISO, IEC, and IEEE. 29148:2011 - Systems and software engineering - Requirements engineering. Technical report, 2011.

[8] I. Jacobson, G. Booch, and J. E. Rumbaugh. *The unified software development process*. Addison-Wesley, 1999.

[9] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, 2008.

[10] L. Kof. Treatment of passive voice and conjunctions in use case documents. *NLDB*, 2007.

[11] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, 2012.

[12] R. S. Pressman and D. Ince. *Software engineering: a practitioner's approach*. McGraw-Hill, 1992.

[13] P. Runeson. Using Students as Experiment Subjects - An Analysis on Graduate and Freshmen Student Data. In *EASE*, 2003.

[14] F. Schneider and B. Berenbach. A Literature Survey on International Standards for Systems Requirements Engineering. In *CSER*, 2013.

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

Publication E: Experiences from the Design of an Artifact Model for Distributed Agile Project Management

Authors Henning Femmer, Marco Kuhrmann, Jörg Stimmer, and Jörg Junge

Venue International Conference on Global Software Engineering (ICGSE'14)

Abstract The organization of projects with distributed teams is a demanding task for every project manager. Requirements need to be collected, documented, and discussed, and the resulting tasks must be distributed to the responsible sites. These activities require an efficient and continuous communication. Furthermore, it is necessary to monitor a project and to track its progress from a management perspective.

As a solution, we opt for a monitoring strategy that is based on the project artifacts and corresponding reports. For this, we defined in a previous work a generic artifact model for agile methods to enable seamless communication and data exchange between projects and teams. In this paper, we present a concrete instance aiming at providing the backbone of the information and data exchange subsystem of a SaaS-based collaborative project management and governance software for distributed software development. We present the artifact model, give insights into its development, and discuss its feasibility.

Our findings show that while the previously defined reference model adequately reflects basic concepts and thus allows for coupling distributed projects, we need to refine the artifact model to emphasize project management/governance and its implementation in tools.

Extended Summary This paper is summarized in Section 4.2.1.

Authors Contributions I co-designed and co-executed the study, and analyzed and co-reported the results.

Experiences from the Design of an Artifact Model for Distributed Agile Project Management

Henning Femmer, Marco Kuhrmann
Technische Universität München
Garching, Germany
{femmer,kuhrmann}@in.tum.de

Jörg Stimmer, Jörg Junge
pliXos GmbH
81247 Munich, Germany
{joerg.stimmer,joerg.junge}@plixos.com

***Abstract*—The organization of projects with distributed teams is a demanding task for every project manager. Requirements need to be collected, documented, and discussed, and the resulting tasks must be distributed to the responsible sites. These activities require an efficient and continuous communication. Furthermore, it is necessary to monitor a project and to track its progress from a management perspective. As a solution, we opt for a monitoring strategy that is based on the project artifacts and corresponding reports. For this, we defined in a previous work a generic artifact model for agile methods to enable seamless communication and data exchange between projects and teams. In this paper, we present a concrete instance aiming at providing the backbone of the information and data exchange subsystem of a SaaS-based collaborative project management and governance software for distributed software development. We present the artifact model, give insights into its development, and discuss its feasibility. Our findings show that while the previously defined reference model adequately reflects basic concepts and thus allows for coupling distributed projects, we need to refine the artifact model to emphasize project management/governance and its implementation in tools.**

***Index Terms*—agile methods; artifact model; governance; distributed projects; project management; experience report**

I. Introduction

Managing a software project is a demanding task, as many activities need to be organized, e.g., requirements elicitation, software development, test, or solution delivery. Distributed software development pushes project management to the next level of difficulty. Project management is performed in a setting in which teams are spread across the globe. Hence, project managers have to deal with time zones, asynchronous communication, language and culture barriers [1], heterogeneous software processes, or delayed/inconsistent reports not adequately reflecting the current project progress. In order to successfully manage distributed settings, project managers require sophisticated tool support that allows for a "near real time" project monitoring and tracking, and that allows the distributed team to share information and artifacts.

A centralized solution in which a shared team server provides the team with the necessary infrastructure seems to be the "promised land", as infrastructure (e.g., repository, test, or build server) as well as collaboration options (e.g., built-in real-time communication and collaboration support) [2]–[4] are provided. Consequently, the question for the data to be exchanged and its structure arises.

Problem Statement & Research Objective. In [5], we proposed a generic artifact model to support data exchange among distributed project sites. The proposed model addressed the need to standardize data structures to abstract from local processes. However, the generic artifact model emerges from a systematic literature review and, thus, only reflects state-of-the-art as reported in literature. In the paper at hand, we discuss the feasibility of the proposed model in practice by investigating the following research questions:

RQ	Question and Rationale
RQ1	*Is the generic artifact model sufficiently complete for practical application?* When applying the generic artifact model in practice, we need to analyze, customize, and/or revise the model in order to address specific (technical) needs. Therefore, we investigate how well the generic model reflects real world requirements, especially, in terms of completeness of the model.
RQ2	*Is the generic artifact model sufficiently precise for practical application?* One purpose of an artifact model is to provide a precise description of a data model. Due to the nature of the generic artifact model, we expect refinements, e.g., of attributes. Thus, we investigate the precision of the generic model compared to the refined model.
RQ3	*Is the refined model still compatible with the generic artifact model?* Based on the generic model, we defined a process interface for data exchange. We investigate whether the refined model is compatible with the generic model to evaluate the feasibility of the generic model for establishing a common project and process interface.

Contribution. In this paper, we contribute an artifact model to support the construction of tools for managing distributed projects. For this, we use a previously defined reference artifact model for agile methods and enhance it for the use as a real-world data exchange model for a SaaS-based collaborative project management tool. We provide an analysis to compare the models and discuss our lessons learnt during the collaboration with practitioners. The rest of the paper is organized as follows: In Sect. II, we discuss related work and previously published material. In Sect. III, we introduce the improved artifact model, analyze it in Sect. IV, present our lessons learnt in Sect. V, before summarizing the paper in Sect. VI.

II. Background & Related Work

Agile methods gained much attention over the years and, beyond small-scale and co-located development, are also subject for research in distributed software development, e.g., [6]–[8]. Hersleb and Mockus [9] investigated collaboration in

978-1-4799-4360-9/14 $31.00 © 2014 IEEE
DOI 10.1109/ICGSE.2014.9

distributed projects. As part of collaboration, direct communication is a basic pillar of agile methods. However, direct communication is hard to realize in distributed settings and, thus, often relies on tools [**?**], [10]. In [11], we found artifacts as means to direct communication between projects beneficial. However, regarding agile processes, we also found significant gaps of artifact descriptions. In response, we conducted a systematic literature review to improve the understanding of artifacts in agile methods, and proposed an artifact-based process interface to couple distributed projects [5].

A Generic Artifact Model for Agile Methods. The motivation behind the general artifact model is that in the context of distributed development the data (structures) and information of the local processes usually differ. However, although processes may differ, all information that is exchanged needs some degree of standardization. The basis for defining an artifact model to support distributed project management is a generic artifact model that provides a general notion of agile methods, and how agile methods can be systematized [5]. The artifact model emerges from a comprehensive literature study in which the reported state of the art on the use of artifacts in agile methods was investigated. The complete model ist shown in [5]; in the following, we describe the most important parts.

The heart of the artifact model is the class *Artifact*, representing any piece of information created in a project (realized using the composite pattern, cf. [**?**]). Artifacts have dependencies among each other to model relationships to enable tailoring for creating project-specific artifact models. Also, artifacts have roles assigned that are either responsible for the artifacts' creation, or that contribute to an artifact's creation or modification. All elements of the model are subclasses of *Artifact*, e.g., *BacklogItem*, *Code*, or *Deliverable*. In order to directly support the description of a project-specific approach, the artifact model comprises several refined artifact types addressing typical project disciplines, e.g., project-, task- and time management: *TeamMember*, *ProjectBacklogItem*, or *Task*; development: *SourceCode*, *Ressource*, or *Release*; quality assurance: *TestCase* or *UnitTest*; or requirements engineering: *Feature*, *UserStory*, or *UseCase*. However, we designed the artifact model in a generic fashion that requires a refinement (extension, deletion, modification of classes and attributes) respecting the actual organization and project context.

III. An Artifact Model for Agile PM

In the following sections, we present a context-specific refinement of the previously defined artifact model [5] to support the development of a tool for distributed agile project management and governance. In order to transfer the ideas and concepts into practice, we refined the artifact model and adopted it for being used as a backbone for a real-world application. In this section, we describe the refinement approach, before presenting the refined artifact model.

A. Refinement Approach

The refinement was conducted in a cooperation project in which *pliXos GmbH* and TU München investigated solutions to support distributed projects and their management. The goal was to develop an actionable artifact model for a tool under development at pliXos, called the *Outsourcing Director*. For this, we refined the existing model through the following steps: 1) Suitability analysis of the general (*reference*) artifact model. 2) Analysis of the *Outsourcing Director* use cases: specifically, we looked at (implicit/explicit) artifact models of possible collaboration systems, e.g., Microsoft's Team Foundation Server (TFS) [12]. 3) Analysis and design workshops to enhance the reference model and to develop the target model. 4) Feedback (industry and potential customers for the tool) and refinement cycles.

In summary, the refined artifact model was inductively developed. We refined the model in several workshops taking into account the roadmap of the tool development, respective use cases, and especially comprehensive feedback from clients.

B. Refined Artifact Model

The refined artifact model serves the development of a data format to allow for collaboration and data exchange between project sites from the perspective of project management. We describe the model by its packages, and go into details for selected refinements. However, the focus of this paper is not to explain the entire model but to discuss its refinement and applicability and explain our lessons learnt.

1) Resulting Model: Figure 1[1] shows the refined artifact model resulting from the aforementioned refinement steps. Around the key class *Artifact*, we refined elements from the generic artifact model and introduced new elements, which we organized in packages. The package *Artifacts* comprises the artifacts reflecting the project disciplines *Planning*, *Requirements & Specification*, *Change Management*, *Development*, and *Testing*. In these packages, we locate all artifact types that are subject to controlling and reporting, e.g., *Task*, *User Story*, or *Issue*.

To allow for creating a distributed project structure, we added the packages *GloBuS Bus* and *Service Provider Project Configuration*. In the first package, we locate those artifact types necessary to build the common structure between parties. In the GloBuS context, a project has a *Client*, and a project is carried out by at least one *ServiceProvider*. Furthermore, in a project, a number of *GloBuSReleases* (deliverables) are shipped to the client. While the *GloBuS Bus* package represents the overall project, the provider configuration package represents the structure of a sub-project conducted by a service provider. The root element is the class *Project*, which relates the project team (class *TeamMember*) with artifacts being produced by the team. Furthermore, a project produces several deliverables, which are (collections of) artifacts themselves, and that become part of an integrated *GloBuSRelease*. That is, according to the artifact-based design approach, the provider configuration package serves as a proxy for the fine-grained artifact model. All considered artifact types are children of the

[1]This is a simplified version without attributes and relations. The full version can be found here: http://www4.in.tum.de/~kuhrmann/sonst/globus.zip

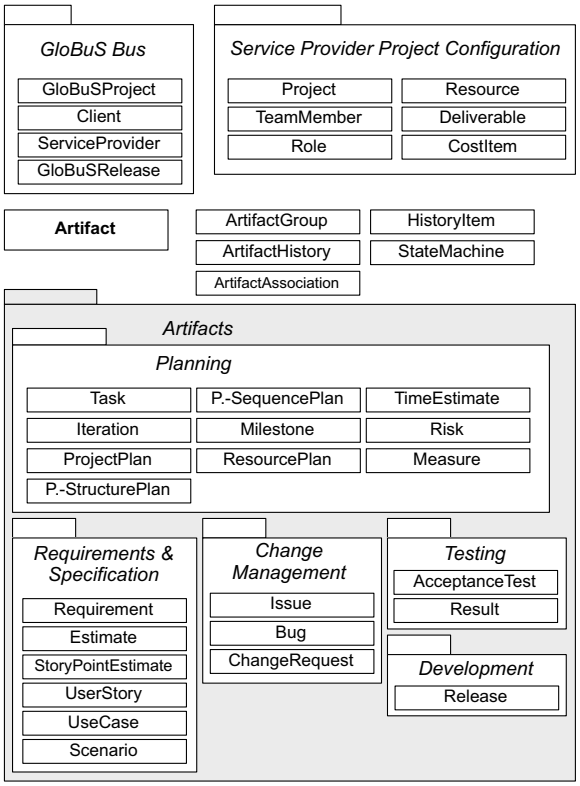


Fig. 1. The refined artifact model for the GloBuS project (simplified).

class *Artifact* and thus allow for polymorphic data structures to be exchanged among (sub-)projects. To support the management of a distributed project, the refined model comprises all artifact types considered necessary to organize and distribute work, and to create reporting lines to track project progress.

2) Examples for Refinements: To use the reference artifact model [5] in practice, we had to refine it. The following two examples provide some insights into the construction procedure.

A major refinement considered essential was the notion of finances. To enable project managers to control a project, effort needs to be estimated, and effort needs to be converted into costs to allow, e.g., for creating offers, or bidding on a publicly announced project. For this, the attribute *estimate* (original model *BacklogItem*) was extracted and explicitly modeled as a class *Estimate* for which the refinements *TimeEstimate* and *StoryPointEstimate* were introduced. Furthermore, a class *CostItem* was defined and linked to resources (internal costs) and roles (external costs). Every Task has resources and several estimates assigned, and, thus, allows for estimating effort, time, or costs. Including internal and external costs allows for internally calculating costs for personnel or consuming resources, and, at the same time, for billing the clients.

As second example for a substantial refinement, we discuss the enriched attribute lists, e.g., for the artifact type *Issue*. The initial study, on which the reference model is based, revealed no fine-grained attribute structure. To make *Issue* artifacts tangible, we relied on previous research in the context of process enactment and adopted work item structures from Microsoft's TFS process templates for the intended artifact model. This approach was then also applied to the remaining artifact types to provide an initial but proven set of attributes.

IV. Analyzing the Artifact Model

In this section, we analyze the refined artifact model to answer the research questions. For this, we briefly show the study design in Sect. IV-A and present the results in Sect. IV-B.

A. Study Design

The overall goal is to provide a refined artifact model aiding the development of project management tools. For this, the general artifact model for agile methods was taken as reference with the purpose to analyze and compare the refined model in a practical context. To investigate the objective, we defined three research questions (see Table I): *RQ1* focuses on the amount of refinement necessary to get from the reference model to the applicable refined model on a coarse-grained level. *RQ2* analyzes the details that we needed to specify to achieve applicability, and *RQ3* analyzes whether the reference model still sufficiently serves as the least common denominator for different artifact models.

To answer the research questions, we conducted a comparative analysis of the reference model and the refined artifact model. In the following, we describe in more detail how we addressed the research questions: For *RQ1*, we walked through each class in both the reference as well as the refined model, and decided which element exists in either both or only a single artifact model. For the discussion, we afterwards qualitatively analyzed intersections as well as gaps between the models, by walking through the elements and reflecting on the rationales behind their presence or absence. For *RQ2*, we looked at the attributes and references of the model and marked those classes that were refined during the process. Again, the results were qualitatively interpreted for the discussion. For *RQ3*, we specifically selected the classes of the reference model that were considered part of the proposed interface [5], which shall define the least common denominator regarding data structures to be exchanged among project sites, e.g., artifacts or planning information. For each class defined in the proposed interface, we decided whether the class was present in the refined model, whether a subclass or renamed class of the original class was present in the refined model, or whether the class was not part of the refined model at all. Afterwards, reasons were again qualitatively analyzed.

Author 1 and Author 2 performed the data collection for these tasks in critical discussion by element-wise comparing artifact model printouts and storing the results in a spreadsheet[2]. The results were validated together with Author 3 and Author 4.

[2]Full dataset: http://www4.in.tum.de/~kuhrmann/sonst/globus.zip

B. Results

The presentation of results is structured according to the research questions.

RQ1: Figure 2 shows the relation between the two models. Apparently, there is a gap between the reference model and the refined model: 28 of the 41 classes (ca. 68%) are not included in the refined model and 25 of the 38 classes (ca. 66%) of the refined model are not present in the reference model. This leads to 13 classes that are present in both artifact models.

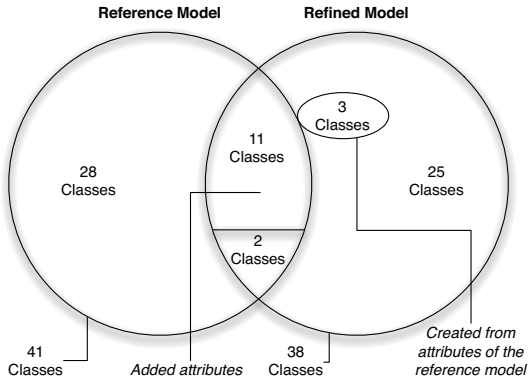


Fig. 2. Comparison of the reference and refined artifact model

Interpretation—When looking into the details, large parts of the original model were removed, whereas other parts, e.g., the *GloBuS Bus*, are completely new (Sect. III-B). We were expecting that some parts must be added because of the context. However, we did not expect the large number of classes that did not appear in the final model. For instance, the packages *GloBuS Bus* and *Service Provider Project Configuration* were completely new. The reason: the generic artifact model does not provide any means to model project configurations, which is, however, necessary when developing project management support. On the other hand, the code-related artifacts from the reference model were removed, as particular code artifacts, e.g., classes or resource file, were considered irrelevant in the actual context. From this, we argue that the reference model, which reflects the reported state of the art of the use of artifacts in agile methods, is yet incomplete from the perspective of project management, as it does not address the overall project organization.

RQ2: As Figure 2 shows, 13 classes are present in both the reference as well as the refined artifact model. Of these 13 classes, 11 classes needed further refinement of the attributes (in summary: 45 attributes were refined and/or added to these classes). Extracting an attribute of a class and creating a separate class with further attributes lead to 3 more refinements. For instance, in the generic artifact model, in the class *Artifact*, information regarding artifact status and history were comprised as attributes. In dialog with industry, it was suggested to make this (essential) information more explicit, e.g., to better support history analyses, or flexible state machine configuration.

Interpretation—This analysis mirrored our perception of the reference model. Even though on a coarse-grained level we felt that the general information is present, we expected that all reused classes must be refined to fit to our concrete purpose and setting. From this, we argue that the general direction of the generic reference artifact model is correct, however, that it needs refinement to be used in practice.

RQ3: The artifact-based process interface as part of the reference model [5] consists of 10 classes. In this research question, we analyzed in how far the refined model can serve to the interface. Figure 3 shows the three different levels of support: directly supported classes are marked with green color, indirectly supported classes were branded with a yellow color (their proxy classes in light-green hatched), and non-supported classes were marked red.

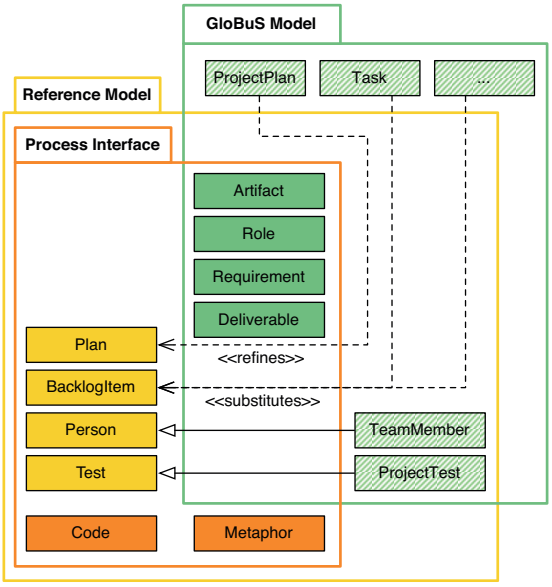


Fig. 3. Supported classes of the interface of the reference model

The figure shows that four classes of the interface (*Artifact*, *Role*, *Requirement* and *Deliverable*) are also present in the refined model and are thus directly supported. Four further classes (*Plan*, *BacklogItem*, *Person*, and *Test*) are not directly part of the refined model; however, classes that are defined as subclasses in the reference model or contain at least the information of the interface (*refines, substitutes* in the figure) were present in the refined model (indirectly supported). Thus, we can say that the information required for the interface is also present in the refined model. For example, even though we did not model a *Person* as class in the refined model, we find the subtype *TeamMember* that contains at least all the information of the class *Person*. Two classes (*Code*, *Metaphor*) are present in the interface, but not in the refined model.

Interpretation—The directly supported artifacts are central to any project management approach, and also, in contrary to the indirectly supported classes, have a very well-defined meaning and purpose. The indirectly supported classes are

mostly very abstract (e.g., *Person*, *Plan*, or *Test*) and thus need more specific details in order to be of value. Lastly, we have to investigate the non-supported classes: First, code was not in the focus of the GloBuS endeavor and, thus, it did not surprise us that there was a gap between the reference and the refined artifact model. For the *Metaphor*, experience has always told us that this is a concept that has been discussed widely in the theory of the XP community, but did not make it into practice. Or, as [13] puts it: "The 'metaphor' is the practice of agile processes most ignored by practitioners". This study once again supports that claim.

V. Lessons Learnt

In our study, we discussed the feasibility of a previous reference artifact model for practical use. In detail, we analyzed how we need to transform the reference model to satisfy the requirements of its deployment in practice. Comparing both models—the reference model and the refined model—we saw that the reference model needed extensive refinement. We found that especially the details regarding the attributes needed substantial revision. However, although several adjustments were necessary to enable the model for practical use, the resulting refined model is still "compatible" with the reference model, and, thus, the intention to define a generic interface to exchange project data is still met with the refined model.

Moreover, our study revealed further interesting insights: the study indicates—again—that the concept of a *metaphor* is widely discussed in theory but apparently not relevant in practice (the respective artifact was removed from the model). Another striking and interesting insight of the results are the predominant concepts of traditional project management, which shape the once very agile artifact model. After extensive discussion with practitioners, we conclude that this is due to the fact that if an agile process is transferred into a global scale, more and more information for the business context is needed—here the practice often relies on the long experience of traditional project management techniques. The experts using these techniques told us that that traditional key-performance indicators are still necessary for controlling regardless of the process used. Moreover, during the design workshops we realized that the more "formalization" (to support tool development) is necessary, the less agility remains in the model. From this, we argue that we still miss adequate techniques to precisely model agile projects today.

We deliberately started to create an artifact model for a particular context in order to extend this model iteratively driven by demand. However, we consider the presented artifact model a first step towards a deeper understanding of structures in agile projects, and for exchanging project information. We are very interested in the feedback of future users of the artifact model, and we cordially invite researchers and practitioners to share their experiences.

VI. Conclusion

In this paper, we described the refinement of an existing reference artifact model for its application in distributed agile project management. We furthermore comparatively analyzed the difference between the generic reference model and the concrete, practically usable refinement for the GloBuS project.

Our study shows that large parts of the artifact model must be refined, either by creating new classes or by adding attributes to existing classes. However, the study also shows that even though refinements are necessary, the original model provides enough material to start with. The original intention (provide common ground) is still valid, and the study shows a compatibility of both models. Our study also raises questions on the dissemination of the concept of a *metaphor* in practice.

Particularly interesting is the result that the artifact model that we created and that started out as an *agile* artifact model had to be extended more and more with *traditional* project management artifacts. This raises a question that must be analyzed in detail in future investigations: *Is it true that even for agile project management, the more we think on a global scale, the more the artifact model resembles models of old-school project management?*

Acknowledgments

The project *GloBuS* was supported by the Bavarian State Ministry for Economic Affairs, Infrastructure, Transport and Technology (StMWIVT) under IUK-1110-0002//IUK391/001.

References

[1] D. Šmite, N. B. Moe, and R. Torkar, "Pitfalls in Remote Team Coordination: Lessons Learned from a Case Study," in *Product-Focused Software Proces (PROFES)*. Springer, 2008.

[2] P. Tell and M. Babar, "Activity Theory applied to Global Software Engineering: Theoretical Foundations and Implications for Tool Builders," in *International Conference on Global Software Engineering (ICGSE)*, 2012, pp. 21–30.

[3] F. Lanubile, T. Mallardo, and F. Calefato, "Tool support for geographically dispersed inspection teams," *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 217–231, 2003.

[4] F. Calefato and F. Lanubile, "Socialcde: A social awareness tool for global software teams," in *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2013, pp. 587–590.

[5] M. Kuhrmann, D. M. Fernández, and M. Gröber, "Towards artifact models as process interfaces in distributed software projects," in *International Conference on Global Software Engineering (ICGSE)*, 2013.

[6] H. Holz and F. Maurer, "Knowledge Management Support for Distributed Agile Software Processes," in *Advances in Learning Software Organizations*. Springer-Verlag, 2003.

[7] M. Paasivaara and C. Lassenius, "Collaboration practices in global inter-organizational software development projects," *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 183–199, 2004.

[8] H.-C. Estler, M. Nordio, C. Furia, B. Meyer, and J. Schneider, "Agile vs. structured distributed software development: A case study," in *ICGSE*, 2012.

[9] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, 2003.

[10] M. R. Thissen, J. M. Page, M. C. Bharathi, and T. L. Austin, "Communication tools for distributed software development teams," in *the 2007 ACM SIGMIS CPR conference*. ACM Press, 2007, pp. 28–35.

[11] M. Kuhrmann, C. Lange, and A. Schnackenburg, "A survey on the application of the v-modell xt in german government agencies," in *Proceedings of the 18th Conference on European System & Software Process Improvement and Innovation (EuroSPI)*, 2011.

[12] M. Kuhrmann and G. Kalus, "Werkzeugspezifisches Tailoring für das V-Modell XT," Technische Universität München, Research Report (in German) TUM-I0804, 2008.

[13] J. Tomayko and J. Herbsleb, "How Useful Is the Metaphor Component of Agile Methods? A Preliminary Study," CMU, Tech. Rep., 2003.

Publication F: Rapid Requirements Checks with Requirements Smells: Two Case Studies

Authors Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer

Venue 1st International Workshop on Rapid Continuous Software Engineering (RCoSE) at the 36th International Conference on Software Engineering (ICSE), 2014

Abstract Bad requirements quality can have expensive consequences during the software development lifecycle. Especially, if iterations are long and feedback comes late – the faster a problem is found, the cheaper it is to fix.

We propose to detect issues in requirements based on requirements (bad) smells by applying a light-weight static requirements analysis. This light-weight technique allows for instant checks as soon as a requirement is written down. In this paper, we derive a set of smells, including automatic smell detection, from the natural language criteria of the ISO/IEC/IEEE 29148 standard.

We evaluated the approach with 336 requirements and 53 use cases from 9 specifications that were written by the car manufacturer Daimler AG and the chemical business company Wacker Chemie AG, and discussed the results with their requirements and domain experts.

While not all problems can be detected, the case study shows that lightweight smell analysis can uncover many practically relevant requirements defects. Based on these results and the discussion with our industry partners, we conclude that requirements smells can serve as an efficient supplement to traditional reviews or team discussions, in order to create fast feedback on requirements quality.

Extended Summary This paper is summarized in Sections 4.3 and 4.4.

Authors Contributions I designed approach and case study, executed the interviews, analyzed and reported on the results.

https://doi.org/10.1145/2593812.2593817

Rapid Requirements Checks with Requirements Smells: Two Case Studies

Henning Femmer
Technische Universität München, Germany
femmer@in.tum.de

Daniel Méndez Fernández
Technische Universität München, Germany
mendezfe@in.tum.de

Elmar Juergens
CQSE GmbH, Germany
juergens@cqse.eu

Michael Klose
Wacker Chemie AG, Germany
michael.klose@wacker.com

Ilona Zimmer
MBtech Group GmbH & Co. KGaA, Germany
ilona.zimmer@mbtech-group.com

Jörg Zimmer
Daimler AG, Germany
joerg.zimmer@daimler.com

ABSTRACT

Bad requirements quality can have expensive consequences during the software development lifecycle. Especially, if iterations are long and feedback comes late – the faster a problem is found, the cheaper it is to fix.

We propose to detect issues in requirements based on requirements (bad) smells by applying a light-weight static requirements analysis. This light-weight technique allows for instant checks as soon as a requirement is written down. In this paper, we derive a set of smells, including automatic smell detection, from the natural language criteria of the ISO/IEC/IEEE 29148 standard.

We evaluated the approach with 336 requirements and 53 use cases from 9 specifications that were written by the car manufacturer Daimler AG and the chemical business company Wacker Chemie AG, and discussed the results with their requirements and domain experts.

While not all problems can be detected, the case study shows that lightweight smell analysis can uncover many practically relevant requirements defects. Based on these results and the discussion with our industry partners, we conclude that requirements smells can serve as an efficient supplement to traditional reviews or team discussions, in order to create fast feedback on requirements quality.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specification

General Terms

Requirements Engineering, Quality Assurance, Natural Language

RCoSE'14, June 3, 2014, Hyderabad, India
ACM 978-1-4503-2856-2/14/06
http://dx.doi.org/10.1145/2593812.2593817

Keywords

Requirements Engineering, Analytical Quality Assurance, Requirements Smells

1. INTRODUCTION

Issues in requirements, such as ambiguities or incomplete requirements specifications, can lead to time and cost overrun in the project [20]. Generally speaking, a problem that is found late in the project is more expensive than if it was found early [6]. Therefore, we need fast feedback cycles that enable to react early to pitfalls during requirements engineering.

Some of these issues require specific domain knowledge to be uncovered. For example, it is very difficult to detect with automatic approaches whether a requirements specification is lacking necessary features.

However, other issues can be detected more easily: If a specification states that a sensor should work with *sufficient accuracy*, without detailing what sufficient means in the context, the specification is incomplete. The same holds for other pitfalls such as loopholes: Phrasing that a certain property of the software under development should be fulfilled *as far as possible* can cause misinterpretations and difficult consequences during the acceptance phase of a product.

Consequently, an approach that gives requirements engineers and project participants fast feedback on possible issues in the specification could provide valuable feedback. However, since requirements in industry are nearly exclusively written in natural language [21] and natural language has no formal semantics, these issues are hard to detect. To face the challenge of fast feedback and the imperfect knowledge of a specification's semantics, we created an approach that is based on what we call *requirements (bad) smells*, which are concrete symptoms for a requirement artefact's quality defect.

In this paper, we settle on the ISO/IEC/IEEE 29148:2011 standard [15] (in the following: *ISO 29148*) as a definition for requirements quality. The standard supplies a list of so-called *Requirements Language Criteria*, such as loopholes, ambiguous adverbs or comparative and negative statements. Based on this standard, we present a set of 8 smells that

indicate potential issues in requirements specifications according to the standard and implement an automatic smell detection for their discovery.

In two case studies, we applied the smell detection to 336 requirements and 53 comprehensive use cases from 9 specifications that were created in 2 different companies. Based on the results, we analyse with experts from the respective companies whether requirements smell analysis can be a beneficial approach for detecting defects in requirements specifications.

2. RELATED WORK

Various authors have worked on quality assurance of software requirements. Some focus on the classification of quality into characteristics [6], others develop comprehensive checklist, e.g. [19], [3], [2] or constructive approaches, e.g. [7], to name only a few.

Some researchers focussed on automatic detection of specific defects in requirements specifications. This includes detection of cloning in RE artefacts [16], detection of similar requirements [9], ambiguity [12], or detection of missing information and passive sentences [18].

Some groups have developed tools that focus on a broader understanding of requirements quality, instead of just a single aspect, e.g. the ARM tool [28] that is based on the IEEE 830 standard [14] that aims at developing metrics for requirements quality instead of giving feedback to developers. Consequently, only quantitative evaluation is performed.

Also the QuaRS tool [5, 8] analyses natural language requirements. However, their approach is based on a proprietary quality model and we could not find a discussion of the results with practitioners.

Circe [11] is able to detect more violations of quality characteristics in a more exact way by building logical models of the requirements specifications. However, their approach assumes that the specifications is written in certain patterns. This is often not the case in industry.

Research Gaps: We identified gaps from three sides: evaluation, quality definition and technique.

A major drawback that we see with the existing approaches, are the evaluations. Only few of the works apply their ideas on real industry specifications. Those who do apply their approach on real industry specifications, only give quantitative summaries, explaining which finding was detected and how often. Some authors also give examples of findings, but we could not find a detailed evaluation of how the findings relate to acknowledged requirement defects, i.e. together with the people who are supposed to use the approach. In our opinion, especially in the tacit domain of natural language, we must understand the impact of a finding in order to justify its detection.

Second, the existing approaches are based on proprietary definitions of quality, based on experience or on what can be measured. We have not seen an approach based on the novel ISO 29148 standard. Also, we have not seen an explicit understanding of how the produced findings relate to quality or quality defects. Our approach of smells covers this aspect systematically.

Lastly, from the technical side, different rules from the new quality standard required specific solutions: We have not seen the use of morphological analysis in requirements engineering quality assurance.

3. REQUIREMENTS SMELLS

In this section, we first introduce a short terminology on requirements smells, we then describe which smells we created, and finally explain how we detect the smells.

3.1 Requirements Smell Terminology

One concept for lightweight quality analysis are smells, which are proposed in the work by Fowler and Beck [10] to answer the question: At which point is the quality of code so low that we need to change it? According to the authors, the answer cannot be objectively measured, but we can only look for certain symptoms. This idea has also been transferred to (Unit) Test Smells [27] and finally to Natural Language Test Smells for user acceptance tests [13]. We apply the concept of smells to requirements.

Accordingly, we define *requirements quality* in terms of fitness-for-purpose, which implies that *bad quality* is a general property of a requirements artefact that has negative effects on activities in the software lifecycle.

Furthermore, a *quality defect* is a concrete instance or manifestation of bad quality in the artefact. This enables us to define a *requirements (bad) smell* as a concrete symptom for a requirement artefact's quality defect. In contrast to requirements defects, a requirements smell only shows a concrete indication for bad quality. Additionally, whether a smell turns into a problem is very *context*-specific. Lastly, we define a *finding* as instances of a smell, which might or might not be a defect.

3.2 Requirements Smell Design

We develop requirements smells based on an existing definition of quality. For this paper, we took the ISO 29148 requirements engineering standard [15] as a baseline. The reasons for this standard were two-fold:

First, the ISO 29148 standard has been created to harmonise a set of existing standards, including the well-known IEEE 830:1998 [14] standard. It differentiates between quality characteristics for a set of requirements, such as completeness or consistency, and quality characteristics for individual requirements, such as unambiguity, singularity etc. The standard furthermore describes the usage of requirements in different project phases and describes exemplary contents and structure for requirements specifications.
Therefore, we argue that this standard is based on a broad agreement and acceptance. Recent literature studies come to the same conclusion [24].

Second, the standard provides readers with a list of so-called *requirements language criteria*, which should help to choose proper language for requirements specifications. The authors of the standard argue that violating the criteria results

> in requirements that are often difficult or even impossible to verify or may allow for multiple interpretations. [15, p.12]

In detail, the requirements language criteria consist of the following elements: [15]

Ambiguous Adverbs and Adjectives refer to adverbs and adjectives that are unspecific.
Examples: *almost always*, *significant*, *minimal*.

Vague Pronouns are unclear relations of a pronoun.
Example: *The system must have a keypad and a keyboard.* It *should use the German layout.*

Table 1: All implemented Smells

Smell Name	Implementation
Ambiguous Adverbs and Adjectives Smell	Dictionaries
Vague Pronouns Smell	POS tagging: Substituting pronouns
Subjective Language Smell	Dictionaries
Comparative Phrases Smell	Morph. Analysis: Adjectives and adverbs in comparative form POS tagging: Conjunctions of comparison
Superlatives Smell	Morph. Analysis: Adjectives and adverbs in superlative form
Negative statements	Dictionaries
Non-verifiable Terms Smell	Dictionaries
Loopholes Smell	Dictionaries
Incomplete references	Not implemented

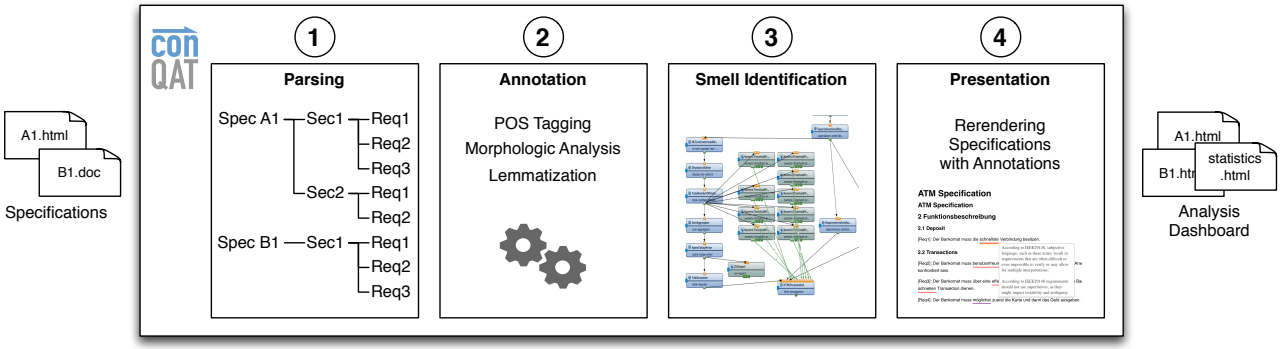


Figure 1: The Overall Smell Detection Process

Subjective Language refer to words of which the semantics is not objective.
Examples: *user friendly*, *easy to use*, *cost effective*

Comparative Phrases are used in requirements that express a relation of the system to specific other systems.
Examples: *better than*, *higher quality*

Superlatives are used in requirements that express a relation of the system to all other systems.
Examples: *best performance*, *lowest response time*.

Negative Statements are "statements of system capability not to be provided"[15]. Some argue that negative statements can lead to underspecification.
Example: *The system must not accept VISA credit cards.*
For this example, a more complete specification describes how the system reacts on the unaccepted input.

Open-ended, Non-verifiable Terms are hard to verify as they offer a choice of possibilities.
Examples: *provide support*, *but not limited to*, *as a minimum*

Loopholes enable stakeholders to ignore certain parts of the specification.
Examples: *if possible*, *as appropriate*, *as applicable*

Incomplete References are references that a reader cannot follow (e.g. no location provided).
Example: *[1] "Unknown white paper". Peter Miller.*

In the following we use all of these characteristics except for *incomplete references* as there were no explicit references in our specifications. All remaining features were considered as smells for bad quality of requirements specifications. At this point we assume that these criteria apply for all specifications; however, we will discuss the appropriateness of the given list based on concrete experience from the case studies in Section 5.2.

3.3 Requirements Smell Detection

The requirements smell detection, as presented in this paper, serves the automatic identification of requirements smells to support further manual quality assurance tasks (and potential corrections of requirements smells). In the following, we introduce the process for the automatic part of the approach, i.e. the detection of requirements smells.

The process consists of four steps (see Fig. 1): Parsing the specifications into single requirements, annotating the requirements with meta-information, detecting the requirements smells and finally creating a human-readable presentation of the findings, which are displayed integrated into the specification.

For the annotation and smell detection phase we employ various techniques, including techniques from Natural Language Processing (NLP)[17]. The smell detection is based on three different techniques. Tbl. 1 gives an overview of the techniques used for each individual smell.

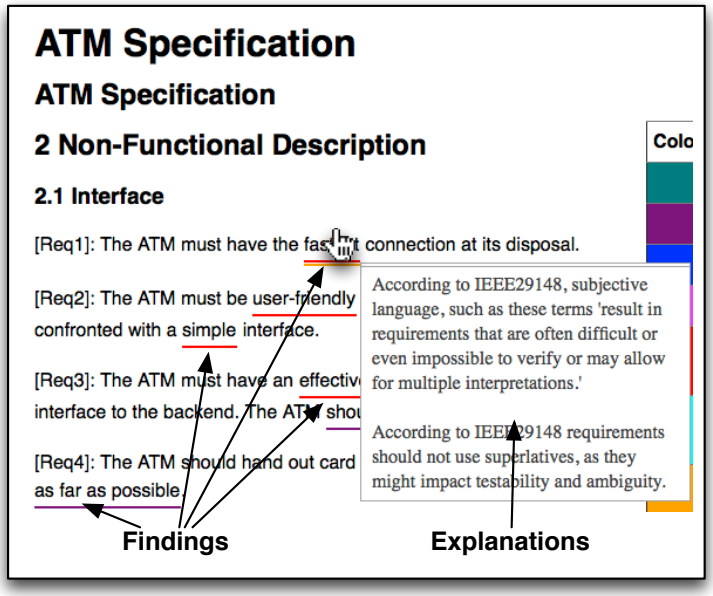


Figure 2: A Sample Output from the Smell Detection Tool for a Dummy Specification

POS Tagging: For two smells, we use a technique called part-of-speech (POS) tagging. Given a sentence in natural language text, it determines the role and function of each single word in the sentence. The output is usually a so-called *tag* for each word, e.g. whether a word is an adjective, a particle or a possessive pronoun. We used the Stanford NLP library [26] and the RFTagger [23] for this. Both are statistical, probabilistic taggers that train models similar to Hidden Markov Models, based on existing databases of tagged texts. A detailed introduction into the technical details of POS tagging is beyond the scope of this paper, but can be found, for example, in [17]. We use POS tagging to determine so-called substituting pronouns. These are pronouns that do not repeat the original noun and, thus, need a human's interpretation of its dependency.

Morphological Analysis: Based on POS tagging, we perform a more detailed analysis of text and determine its inflection. This contains, among others to determine a verb's tense or an adjective's comparison. We use this technique to analyse if adjectives or adverbs are used in their comparative or superlative form.

Dictionaries: For the remaining five smells we use dictionaries, based on the proposals of the standard [15], and on our experience in the case studies. We furthermore apply lemmatisation for these words, which is a normalisation technique that reproduces the original form of a word. In other words, if a lemmatiser is applied to the words *were*, *is* or *are*, the lemmatiser will for all three return the word *be*. Lemmatisation is in its purpose very similar to stemming (e.g. the famous Porter Algorithm [22]), yet not based on heuristics, but on the POS tag as well as the word's morphological form.

The whole approach is implemented on top of the software quality analysis toolkit ConQAT[1]. ConQAT offers a platform for detailed data analysis, which we extended with NLP features. We furthermore developed a presentation that allows to read the finding in its context. In this presentation, the complete specification is displayed, and findings are annotated in a spelling-correction style. This follows the idea of smells as only indications that must be evaluated holistically in its context. Lastly, the system gives detailed information, when a user hovers a finding (see Fig. 2).

4. EVALUATION

We evaluated the approach in two case studies with 9 specifications from industry. In the following, we report on these studies.

4.1 Research Questions

For this first evaluation, we had three research questions in mind:

RQ1: Can we find defects with smell detection? First, we want to discuss the potential of the approach of requirements smells and lightweight smell analysis. To this end, we ask whether the approach produces results that pinpoint to defects of the requirements specification

RQ2: How many findings are present in the requirements specifications? Second, besides the potential of the approach, we also want to analyse the outcomes from a requirements engineering perspective. For this, we wanted to understand the distribution of findings across domains, specifications and the different smells.

[1] `http://www.conqat.org`

RQ3: Would requirements engineers use a requirements smell tool? Last, we wanted to have an opinion of both experts on whether or not a requirements smells approach would be useful for them.

4.2 Study Design

The study consisted of an automatic smell detection (see Fig. 1) and a manual evaluation phase (see Fig. 3):

In the smell detection phase, the specifications are first parsed from their original format (`.html` and `.doc`) into a machine readable format and parsed into individual requirements or use cases. Afterwards, each sentence and each word in each requirement is annotated with the information necessary for smell analysis, which is the POS information, morphologic information and the lemmatised word (see Sec. 3.3). Next, all smells detection algorithms are performed for each requirement, producing a set of findings for each requirement. These are subsequently presented in so-called *Analysis Dashboards*, which are human-readable presentations, including statistics and annotated views on the specification.

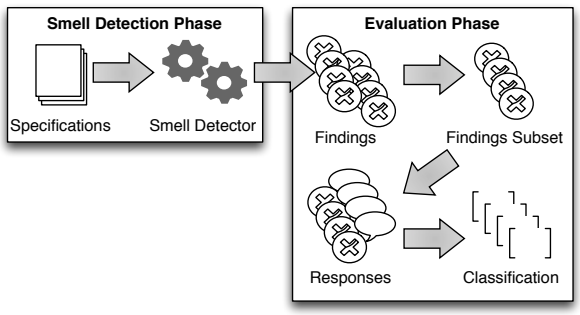


Figure 3: The Data Collection Process

In the evaluation phase, we first had to chose a subset of findings from the whole set of findings produced by the smell detection, as many findings are similar and the time of experts is rare. For the selection, we tried to filter false positives[2] and find samples from all smells.

For this paper, we were interested in the results of a lightweight static analysis of requirements. In order to receive unbiased and open opinions, we asked the participating experts of their opinion on these samples as an open question instead of a closed evaluations (e.g. ratings on a Likert scale).

Afterwards, two authors classified the qualitative answers independently using open coding (as known from the e.g. Grounded Theory approach [1]). Afterwards, we compared the codes as a validity procedure and in case of inconsistencies, resolved them.

Lastly, we asked the participating experts whether requirements smells tool could support their work.

4.3 Case & Subject Description

The case selection was driven by opportunity. We were approached by companies that wanted to understand whether there are issues in their requirements. All specifications were written in German language.

The two cases come from different domains and, thus, we have different representations of the requirements. Both companies are very successful and mature in their software development and are investing into good requirements engineering. In the following, we describe the chosen cases in detail.

Daimler AG: Daimler AG is one of the key players in the automotive industry with several hundred thousand employees and over 100 billion US dollar revenue.

At Daimler AG, we analysed six different specifications that were written by various authors. The specifications describe functionality in various domains of engine control as well as driving information. In this case, requirements are written down in the form of sentences, identified by an ID. The authors are domain experts who are coached on writing requirements.

The specifications A1-A6 that we analysed consist of 323 requirements (see Table 2). All of the specifications of Daimler AG analysed in our study were created by domain experts in a pilot phase after a change in requirements engineering at Daimler AG. We reviewed 22 findings with an external coach who works as a consultant for requirements engineering tightly collaborating with the group for many years.

Wacker Chemie AG: In the second case, we analysed specifications of business information systems from Wacker Chemie AG (or short: Wacker). Wacker is a globally active chemical company headquartered in Munich, Germany, with about 16,000 employees and a revenue of 4.63 billion Euros (2012). The systems that we analysed fulfil company-internal purposes, such as systems for access to Wacker buildings or support systems for document management.

We analysed three Wacker specifications that were written by five different authors. At Wacker Chemie AG, functional requirements are written as use cases (including fields for *Name*, *Description*, *Role*, and *Precondition*), whereas non-functional requirements are described in simple sentences. The specifications consisted of 53 use cases and 13 unstructured requirements (see Table 2). For the reviews of the findings, we selected 18 findings and discussed them with the Chief Software Architect, who also has several years of experience in quality assurance.

4.4 Results

We base our answer to RQ1 on the produced presentations of our tool and the responses to the findings by the experts, the answer to RQ2 on the total number of findings, and the answer to RQ3 on the written statements by the experts.

RQ1: Can we find defects with lightweight smell analysis?

We wanted to understand the potential of the approach in terms of ability to detect requirements defects. In the following, we first provide some examples before summarising the results from our analysis.

Examples: The automatic smell analysis produced 356 findings over all 9 specifications (see Tbl. 5), of which we selected 40 findings ($\sim$11%) to discuss with the analysts in depth (see Fig. 3). An exemplary finding of each smell is shown in Tbl. 3. To demonstrate the types of issues that can be found, we will discuss three issues in depth here and go over a summary of the remainder afterwards.

[2]Obviously, this prevents us from analysing the precision of our smell detection.

Table 2: Study Objects

Specification	Topic	Size (Words)	Size (Sentences)	# Requirements	# Use Cases
A1	Adaptive valve control	2098	121	91	
A2	Exhaust control	2540	125	72	
A3	Driving information	215	13	12	
A4	Engine startup control	1118	76	44	
A5	Engine control	579	49	49	
A6	Powertrain communication	1248	66	55	
SUM		7798	450	323	
B1	Management of access control	2337	172	9	18
B2	Event notification	1162	103	3	19
B3	Document management	490	26	1	16
SUM		3989	301	13	53

Table 3: Exemplary Findings; shortened and translated from German by the Authors

Smell Name	Exemplary Finding
Ambiguous Adverbs and Adjectives Smell	If the (...) quality is **too low**, a fault must be written to the error memory.
Vague Pronouns Smell	The software must implement services for applications, **which** must communicate with controller applications deployed on other controllers. [Note: The translation is less ambiguous than the original finding in German, as the reflexive pronoun in English identifies its relation more clearly. The original requirement stated: *Die Software muss Dienste für Anwendungen implementieren, welche über ein Steuergerät hinaus mit anderen Steuergeräte-Anwendungen kommunizieren müssen.*]
Subjective Language Smell	The architecture as well as the programming must ensure a **simple** and **efficient** maintainability.
Comparative Phrases Smell	The display (...) contains the fields A, B and C, as well as **more exact** build infos.
Superlatives Smell	The system must provide the signal in the **highest** resolution that is desired by the signal customer.
Negative statements	One's own user **cannot** be deleted.
Non-verifiable Terms Smell	The system may only be activated, if all required sensors (...) work with **sufficient** measurement accuracy.
Loopholes Smell	**As far as possible**, inputs are checked for plausibility.

Subjective Language Smell In specification B2, a specification of a business information system, we found the requirement that the software *must ensure a simple and efficient maintainability.* This description of a non-functional requirement contains the classical error of violating verifiability. It is very hard to use this requirement properly in other activities of the software development lifecycle, e.g. engineers will find it hard to develop code against this requirement and testers will not be able to decide whether the resulting software fulfils the requirement during acceptance testing. The expert classified two instances of this finding as a major defect.

Vague Pronouns Smell The second example, found in specification A6, exemplarily shows the difficulties that come with complicated grammatical structures. In the example given in Tbl. 3, it remains unclear whether the *software*, the *services* or the *applications* should communicate with other applications. It is sometimes possible to deduce the reference from the context of the requirement, e.g. in this case we could take the word *other applications* as a hint that the word *which* refers to the word *application*. However, we still argue that the requirement contains potential for misunderstandings for all kinds of roles that are in contact with the specification.

Loopholes Smell The third example is taken from specification B1, a requirement artefact that describes the internal software system that manages the guests who access Wacker's properties, including chemical plants. The example contains the ambiguous phrase that a certain requirement should be fulfilled *as far as possible*. This is obviously problematic as, e.g. a developer might have a different opinion on the possibilities than the tester. Accordingly, testing will be performed subjectively.

After showing 40 findings to experts, we classified the qualitative responses to summarise the results. Tbl. 4 shows a summary of the answers, with responses where the experts would take action listed at the top and rather rejecting responses at the bottom part of the table.

Table 4: Qualitative Summary Smell Findings (Open Coding)

Classification (Code)	Occurrence	Explanation
Potential problem	8	This finding revealed a potential problem.
Needs review	6	This requirement needs a review.
Implicit knowledge	4	There is some implicit knowledge, which should be written down.
Missing reference	2	There should be a reference at this point.
Major defect	2	This is a big issue that must be addressed.
Refinement expected	6	While this is not an issue here, it must be further explained and refined at a different point.
No need for high quality	2	This could be problematic, but this part of the specification is not so important (e.g. information only, see Sec. 5)
Domain specialists knowledge	4	This finding seems problematic, but is clear to a domain expert.
No problem	4	This is not a problem here.
Finding wrong	1	The smell detection did not work properly.
Unsure whether a negative is a problem	1	It is unclear whether and why this formulation should be a problem (see Sec. 5)

Responses: We can see that there are many requirements in which experts directly proposed actions on this specification. Two times, which are two findings similar to the `Subjective Language Smell` example in Tbl. 3, the expert concluded that this is a major defect. The findings revealed especially issues with unstated information, i.e. *implicit knowledge*, and *missing references*. Six times, the expert explained that the finding probably pinpoints to a defect and that he would suggest a further review.

On the other side, experts classified that for some of the findings they would not take any further action. The most frequent cause was related to the subject under analysis: 8 times the experts told us that we were looking at the wrong spot, either because they said that this part is not really relevant in the specification[3] or because they stated that the requirements should be refined in a different artefact. These findings imply that a detailed understanding of purpose of the requirements is necessary to detect issues where they really matter. Another very common reason was that *domain specialists knowledge* is the reason for the finding, but there it was not seen necessary to make this knowledge explicit.

To summarise, we can see that the smell approach is able to detect requirements defects, as exemplified and validated with the experts.

RQ2: How many findings are present in the requirements specifications?

We wanted to understand in how far findings of the different smells are present in the specifications. Therefore, we analyse the distribution of findings across three dimensions: How are findings distributed across specifications, domains, and requirements smells? Tbl. 5 shows the total number of findings for all natural language criteria of the standard.

We see that nearly *all specifications* are subject to smells. The distribution varies between 0.3 and 0.62 findings per sentence, with specification B3 as an outlier, which we will discuss in the next paragraph. Obviously, the number of findings increases with the size of the specifications.

Furthermore, it is interesting to see that the total number of findings (see Tbl. 5) are quite similar in *both domains* (0.42 smells per sentence for the A1-A6 and 0.55 smells per sentence for B1-B3). One discrepancy that we looked at in more detail are the number of `loophole` findings. The reason for this was an extensive use of the German verb *soll*, which translates to *should* and is thus non-binding in contracts (in contrary to *shall*; cf. [4] or [15]). Hence, we see this certain error multiple times, especially in specification B3. Requirements authors at Daimler AG, in comparison, are taught to use the standard modalities where appropriate. This explains the discrepancy between the specifications. This is especially reflected in the variable `Smells per Sentence` in Tbl. 5.

The *distribution between the smells* varies strongly. Striking are the number of `negative statements` findings and `vague pronouns` findings. A selection of `negative statements` findings that we presented to our industry partners has lead to discussions with both experts on which we will report in detail in Sec. 5. For the `vague pronouns`, the reason lies in the implementation: As explained in Sec. 3.3, the smell detector suggests all substituting pronouns as findings. However, it turns out in the study that this is an overapproximation. Even though some of the sentences are indeed hard to understand (e.g. the example from Tbl. 3), very often it was very clear which word was substituted by the pronoun. For example, one automotive requirement from specification A4 constrained *The gear lever must be in Position P or N. This is not the case for (...).* In this case, even though the pronoun *this* is substituting, the reference is nevertheless quite clear from the context. Future work could include deeper linguistic dependency analysis of sentences, e.g. following the work of Smith [25].

RQ3: Would requirements engineers use a requirements smell tool?

After the analysis and interviews, we asked the expert of both Wacker and Daimler AG if they can comment whether

[3]Some parts of the specification were only considered to be further information and thus should not need to be of high quality.

Table 5: Quantitative Summary of Smell Findings

Specification	All Smells	Smells per Sentence	Negative Statements Smell	Superlative Requirements Smell	Comparative Requirements Smell	Subjective Language Smell	Loophole Smell	Non-verifiable Term Smell	Vague Pronouns Smell	Ambiguous Adverbs and Adjectives Smell
A1	45	0.37	11	7	7	4	2	0	13	1
A2	57	0.46	14	1	5	6	4	2	24	1
A3	8	0.62	2	0	0	0	0	0	6	0
A4	29	0.38	8	0	1	3	1	1	15	0
A5	20	0.41	5	0	0	0	0	1	14	0
A6	32	0.48	13	0	7	0	0	4	8	0
Sum	191	0.42	53	8	20	13	7	8	80	2
B1	100	0.58	20	6	7	5	18	1	43	0
B2	31	0.30	3	0	9	2	2	0	15	0
B3	34	1.31	0	1	1	0	21	0	10	1
Sum	165	0.55	23	7	17	7	41	1	68	1

or not they think the method is a helpful support. Their answers were:

Expert 1: I think that smells can help to analyse a specification. To use this correctly, the following aspects should be considered:

First, the people who need to write the specification, received training which gives the required performance criteria. Second, abstraction level's must be taken into account during smell detection process, since at higher abstractions level's different criteria can not be met (e.g., vague pronouns or subjective language).

Expert 2: The method of requirements smells is a valuable extension in the area of requirements engineering and gives helpful input concerning the quality of specified requirements in early development phases.

I like to compare requirements smells to the "check spelling aid" known e.g. from Microsoft Word, so for me requirements smells are intuitive and lightweight and should be used and integrated within requirements engineering and quality assurance processes.

Even though this is just anecdotal and, thus, subjective evidence, it forms a first external impression which encourages us to invest more effort into the development of requirements smells and analyse the approach in more depth.

4.5 Threats to Validity

We made four choices that could have had an impact the validity of the results.

First, the classification for RQ1 was performed by the authors. To address this threat, we performed triangulation of the classification between the first and the second author. For this, the second author of the paper conducted an independent classification of the responses to the 40 selected findings without being directly involved in the discussions with the industry participants. Out of 40 classifications, the recoding resulted in 6 out of 40 inconsistent classifications, which were directly resolved in discussions, and 18 out of 40 classifications that were chosen on a more coarse grained level; for instance, the second author selected the code "OK" for responses that indicated to findings being not classified as problems while the first author could reveal more fine-grained codes such as "Unsure whether a negative is a problem". Our interpretation of the independent classification result is, thus, that the results of the coding are sufficiently reliable.

Second, we selected the study objects by opportunity. These were the requirements for which we could get feedback from people with knowledge about the systems. To the best of our knowledge there are no benchmark requirements sets with proper information about its quality. However, we analysed requirements of 9 different systems, both from systems engineering and software engineering of traditional business information systems.

Third, we selected the set of findings that were discussed with industry experts not at random, but at our choice. This was done purposely in order to understand if the approach is generally able to find defects. Due to this decision, the number of issues presented in the results are not necessarily representative for the whole set of findings, i.e. no conclusions can be drawn about the quality of the smell detection approach in terms of precision or recall. All conclusions drawn in this paper respect this assumption.

Last, we selected the expert reviewers by availability. Even though they were familiar with the specifications, in future it would be best to ask not coaches nor architects, but the team members who use the requirements, e.g. testers or developers.

5. DISCUSSION

The study brought up several further questions that have strong implications on future research. Therefore, we discuss several of these aspects in more depth.

5.1 Smells for Rapid Requirements Analysis

In this paper, we analysed the usage of requirements smells in the context of rapid feedback for requirements analysis. In our context, we were able to show that it is possible to find relevant defects with automatic smell detection. The (automatic) smell detection took 59 seconds for A1 to A6 and 24 seconds for B1 to B3 in total. We consider this duration to be an effort reasonable to most types of projects.

However, it is important to note that a smell approach cannot substitute manual reviews or inspections. A requirements smell detection can only pinpoint to possible defects or common pitfalls. As a matter of fact, considering the low effort necessary to conduct an automatic smell detection, we believe that such a lightweight approach would greatly add to human inspections. Those manual approaches, in turn, can find deep flaws in requirements specifications. This includes violations of correctness as well as violations of completeness, such as missing functionality.

For this reason, we argue that requirements smells, just as in code smells, can serve as a very valuable input for inspections or reviews.

5.2 ISO 29148 Language Criteria as Smells

Industry experts did not agree on all natural language criteria that were proposed by ISO 29148.

This was especially the case for the `Negative Statements Smell`. The question whether or not findings of this smell lead to a problem was strongly discussed: On the one hand, the standard argues that negative phrases and statements should be avoided as a type of "unbounded or ambiguous terms". One could argue that formulating requirements in negative statements can lead to incompleteness. For the example given in Tbl. 3, the requirements specification lacks the information on how the system should react in case the user tries to delete his own dataset, or how else the system should prevent this to happen.

There are, on the other hand, non-functional requirements that are very hard to formulate in positive statements, e.g. requirements describing the prevention of system access. In any way, we need to find ways to understand and prove the impact of findings in a less argumentative (as proposed by the standard) and more empirically sound manner.

Another smell that produced interesting results was the `Superlative Requirements Smell`. The reasoning for this smell is that a requirements that is stated as, e.g. *highest resolution of a signal* or *fastest response of a sensor*, is inherently difficult to verify. However, it again depends on the context of the requirement. Taking the first example, if all possible resolutions are clear (i.e. if the set of possibilities is finite) the requirement is indeed verifiable. A similar argumentation holds for the `Comparative Requirements Smell`. We can see here, that the Requirements Smells and Requirements Smell Detection must be improved by adding a context to the smell definition. This improves the understanding on when a finding of a smell turns into a defect. It remains open, however, how to include this knowledge into a smell detector.

5.3 Implementation of Smell Detection

Besides understanding how to define the smells and understand which are the best smells to pinpoint to problematic spots in requirements artefacts, we can furthermore discuss how to detect the smells most appropriately after their definition. In this paper we basically made use of three techniques: Dictionaries, POS tagging and morphological analysis.

Some implementations depend on solely dictionaries (as others have done with similar problems before [3]). To our experience, this remains the most vague way of detecting certain smells as the dictionaries can only detect a finite set of words. While this is a perfect solution for smells, where there are only finite forms of this smell (also called *closed classes* in natural language processing [17]; e.g. there is only a small set of words that express negation, like "not", "neither", "no", etc.), it is inherently imprecise when it comes to open classes, e.g. `Non-verifiable terms`. Dictionaries furthermore inherently struggle when it comes to domain-specific language. So far, the only chance we see in this case is to taylor each smell based on feedback from the respective domain.

Other implementations are based on POS tagging and morphological analysis. Since the specifications were written in German and morphological analysis libraries are sparse for the German language, we had to combine various libraries, such as the Stanford NLP [26] with German NLP libraries [23]. We could identify some false positives that arise from the imprecision that comes with these libraries. This is especially the case when it comes to domain-specific terms and proper nouns.

Some implementations could be further refined, if more information from word dependencies would be used (see Sec. 4.4). For example, for the `Vague Pronouns Smell` we could evaluate the linguistic structure even further and try to detect whether there are multiple nouns that this pronoun could refer to.

5.4 Subject of Analysis

In this study we treated all parts of requirements specifications equally and assumed that each requirement had to be of highest quality, e.g. unambiguous, exact and verifiable. While this is true for many cases, experts opposed this assumption in some cases.

In an interview one of the experts told us that a certain finding was an issue on requirements level, but that in this spot it was ok, because this particular requirement was intended to be on a more abstract, goal-like level. This indicates that it is important to understand and taylor the smell detection to the abstraction level and granularity of the requirements (cf. [19]).

The same also holds for information texts and introductions. It is still completely unclear to which extent the quality of these parts of the specifications matter.

6. CONCLUSION

In this study, we proposed a light-weight approach to detect requirements smells. This approach is based on the natural language criteria of ISO 29148 and serves to rapidly detect requirements that violate certain RE principles. We furthermore developed an implementation that is able to detect these violations using part-of-speech (POS) tagging, morphological analysis and dictionaries.

We applied the approach in two case studies to 336 requirements and 53 use cases taken from 9 specifications that were created in 2 different companies. We discussed the results with industry experts and concluded that the approach is suitable to detect relevant defects in requirements. It cannot find all possible defects but experts judged it as a valuable input for requirements reviews. Furthermore, we saw that violations of the natural language criteria are present across domains and various specifications; however, the study also shows that we need further analyses to understand the impact of these violations.

Future work includes understanding negative statements in requirements, enhancement of the smell detection via dependency analysis and development of new requirements smells through interviews with testers and developers. We also want to understand the scalability of the approach: We are currently working on the analysis of a large business requirements specification of an industry partner.

Acknowledgments

We want to thank Daimler AG, especially Heike Frank, and Wacker Chemie AG for their support during the case studies, as well as Sebastian Eder, Maximilian Junker, Benedikt Hauptmann and the anonymous reviewers for their helpful and encouraging reviews.

7. REFERENCES

[1] S. Adolph, W. Hall, and P. Kruchten. Using Grounded Theory to study the Experience of Software Development. *Empirical Software Engineering*, 16(4), 2011.

[2] B. Anda and D. I. K. Sjøberg. Towards an inspection technique for use case models. In *Software Engineering and Knowledge Engineering (SEKE)*, 2002.

[3] D. Berry, A. Bucchiarone, and S. Gnesi. A new quality model for natural language requirements specifications. In *Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2006.

[4] S. Bradner. Key words for use in rfcs to indicate requirement levels, 1997. RFC 2119.

[5] A. Bucchiarone, S. Gnesi, and P. Pierini. Quality Analysis of NL Requirements : An Industrial Case Study. In *Requirements Engineering*, 2005.

[6] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos. Identifying and measuring quality in a software requirements specification. In *Software Metrics Symposium*, 1993.

[7] C. Denger, D. M. Berry, and E. Kamsties. Higher quality requirements specifications through natural language patterns. In *Software Science, Technology, and Engineering*, 2003.

[8] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The Linguistic Approach to the Natural Language Requirements Quality: Benefit of the use of an Automatic Tool. In *NASA Goddard Software Engineering Workshop*, 2001.

[9] D. Falessi, I. C. Society, and G. Cantone. Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques. *Software Engineering*, 39(1), 2013.

[10] M. Fowler and K. Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[11] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. *Software: Practice and Experience*, 32(2):113–133, Feb. 2002.

[12] B. Gleich, O. Creighton, and L. Kof. Ambiguity detection: Towards a tool explaining ambiguity sources. *Requirements Engineering*, 2010.

[13] B. Hauptmann, M. Junker, S. Eder, L. Heinemann, R. Vaas, and P. Braun. Hunting for smells in natural language tests. In *International Conference on Software Engineering (ICSE)*, 2013.

[14] IEEE Computer Society. IEEE Recommended Practice for Software Requirements Specifications. Technical report, 1998.

[15] ISO, IEC, and IEEE. ISO/IEC/IEEE 29148:2011. Technical report, ISO IEEE IEC, 2011.

[16] E. Juergens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schaetz, S. Wagner, C. Domann, and J. Streit. Can Clone Detection Support Quality Assessments of Requirements Specifications? In *International Conference on Software Engineering (ICSE)*, 2010.

[17] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Pearson Education, 2014.

[18] L. Kof. Treatment of Passive Voice and Conjunctions in Use Case Documents. *Natural Language Processing and Information Systems*, 2007.

[19] A. V. Lamsweerde. *Requirements Engineering*. John Wiley & Sons, 2009.

[20] D. Méndez Fernández and S. Wagner. Naming the Pain in Requirements Engineering: Design of a Global Family of Surveys and First Results from Germany. In *Evaluation and Assessment in Software Engineering (EASE)*, 2013.

[21] L. Mich, F. Mariangela, and N. I. Pierluigi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1), 2004.

[22] M. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 1980.

[23] H. Schmid and F. Laws. Estimation of conditional probabilities with decision trees and an application to fine-grained POS tagging. In *Conference on Computational Linguistics*, 2008.

[24] F. Schneider and B. Berenbach. A Literature Survey on International Standards for Systems Requirements Engineering. In *Conference on Systems Engineering Research*, 2013.

[25] N. A. Smith. *Linguistic structure prediction*. Morgan & Claypool Publishers, 2011.

[26] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2003.

[27] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok. *Refactoring test code*. CWI, 2001.

[28] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. Automated analysis of requirement specifications. In *International Conference on Software Engineering (ICSE)*, 1997.

Publication G: Rapid Quality Assurance with Requirements Smells

Authors Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, Sebastian Eder

Venue Journal of Systems and Software

Abstract Bad requirements quality can cause expensive consequences during the software development lifecycle, especially if iterations are long and feedback comes late.

We aim at a light-weight static requirements analysis approach that allows for rapid checks immediately when requirements are written down.

We transfer the concept of *code smells* to Requirements Engineering as *Requirements Smells*. To evaluate the benefits and limitations, we define Requirements Smells, realize our concepts for a smell detection in a prototype called *Smella* and apply Smella in a series of cases provided by three industrial and a university context.

The automatic detection yields an average precision of 59% at an average recall of 82% with high variation. The evaluation in practical environments indicates benefits such as an increase of the awareness of quality defects. Yet, some smells were not clearly distinguishable.

Lightweight smell detection can uncover many practically relevant requirements defects in a reasonably precise way. Although some smells need to be defined more clearly, smell detection provides a helpful means to support quality assurance in Requirements Engineering, for instance, as a supplement to reviews.

Extended Summary This paper is summarized in Sections 4.3 and 4.4.

Authors Contributions I designed the prototype, co-designed the case study, co-executed the interviews, co-analyzed and co-reported the results.

The Journal of Systems and Software 123 (2017) 190–213

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Rapid quality assurance with Requirements Smells

Henning Femmer[a,*], Daniel Méndez Fernández[a], Stefan Wagner[b], Sebastian Eder[a]

[a] Software & Systems Engineering, Technische Universität München, Germany
[b] Institute of Software Technology, University of Stuttgart, Germany

ARTICLE INFO

Article history:
Received 5 January 2015
Revised 19 February 2016
Accepted 28 February 2016
Available online 4 March 2016

Keywords:
Requirements engineering
Automatic defect detection
Requirements Smells

ABSTRACT

Bad requirements quality can cause expensive consequences during the software development lifecycle, especially if iterations are long and feedback comes late. We aim at a light-weight static requirements analysis approach that allows for rapid checks immediately when requirements are written down. We transfer the concept of *code smells* to requirements engineering as *Requirements Smells*. To evaluate the benefits and limitations, we define Requirements Smells, realize our concepts for a smell detection in a prototype called *Smella* and apply Smella in a series of cases provided by three industrial and a university context. The automatic detection yields an average precision of 59% at an average recall of 82% with high variation. The evaluation in practical environments indicates benefits such as an increase of the awareness of quality defects. Yet, some smells were not clearly distinguishable. Lightweight smell detection can uncover many practically relevant requirements defects in a reasonably precise way. Although some smells need to be defined more clearly, smell detection provides a helpful means to support quality assurance in requirements engineering, for instance, as a supplement to reviews.

1. Introduction

Defects in requirements, such as ambiguities or incomplete requirements, can lead to time and cost overruns in a project (Méndez Fernández and Wagner, 2015). Some of the issues require specific domain knowledge to be uncovered. For example, it is very difficult to decide whether a requirements artifact is complete without domain knowledge. Other issues, however, can be detected more easily: If a requirement states that a sensor should work with *sufficient accuracy* without detailing what *sufficient* means in that context, the requirement is vague and consequently not testable. The same holds for other pitfalls such as loopholes: Phrasing that a certain property of the software under development should be fulfilled *as far as possible* leaves room for subjective (mis-)interpretation and, thus, can have severe consequences during the acceptance phase of a product (Femmer et al., 2014b; ISO, IEC and IEEE, 2011).

To detect such quality defects, quality assurance processes often rely on reviews. Reviews of requirements artifacts, however, need to involve all relevant stakeholders (Salger, 2013), who must manually read and understand each requirements artifact. Moreover, they are difficult to perform. They require a high domain knowledge and expertise from the reviewers (Salger, 2013) and the quality of their outcome depends on the quality of the reviewer (Zelkowitz et al., 1983). On top of all this, reviewers could be distracted by superficial quality defects such as the aforementioned vague formulations or loopholes. We therefore argue that reviews are time-consuming and costly.

Therefore, quality assurance processes would benefit from faster feedback cycles in requirements engineering (RE), which support requirements engineers and project participants in immediately discovering certain types of pitfalls in requirements artifacts. Such feedback cycles could enable a lightweight quality assurance, e.g., as a complement to reviews.

Since requirements in industry are nearly exclusively written in natural language (Mich et al., 2004) and natural language has no formal semantics, quality defects in requirements artifacts are hard to detect automatically. To face this challenge of fast feedback and the imperfect knowledge of a requirement's semantics, we created an approach that is based on what we call *Requirements (Bad) Smells*. These are concrete symptoms for a requirement artifact's quality defect for which we enable rapid feedback through automatic smell detection.

In this paper, we contribute an analysis of whether and to what extent Requirements Smell analysis can support quality assurance in RE. To this end, we

* Corresponding author. Tel.: +49 8928917080.
E-mail addresses: femmer@in.tum.de (H. Femmer), mendezfe@in.tum.de (D. Méndez Fernández), Stefan.Wagner@informatik.uni-stuttgart.de (S. Wagner), eders@in.tum.de (S. Eder).

http://dx.doi.org/10.1016/j.jss.2016.02.047
0164-1212/

1. define the notion of *Requirements Smells* and integrate the Requirements Smells[1] concept into an analysis approach to complement (constructive and analytical) quality assurance in RE,
2. present a prototypical realization of our smell detection approach, which we call *Smella*, and
3. conduct an empirical investigation of our approach to better understand the usefulness of a Requirements Smell analysis in quality assurance.

Our empirical evaluation involves three industrial contexts: The companies Daimler AG as a representative for the automotive sector, Wacker Chemie AG as a representative for the chemical sector, and TechDivison GmbH as an agile-specialized company. We complement the industrial contexts with an academic one, where we apply Smella to 51 requirements artifacts created by students. With our evaluations, we aim at discovering the accuracy of our smell analysis taking both a technical and a practical perspective that determines the context-specific relevance of the detected smells. We further analyze which requirements quality defects can be detected with smells, and we conclude with a discussion of how smell detection could help in the (industrial) quality assurance (QA) process.

1.1. Previously published material

This article extends our previously published workshop paper (Femmer et al., 2014b) in the following aspects: We provide a richer discussion on the notion of Requirements Smell and give a precise definition. We introduce our (extended) tool-supported realization of our smell analysis approach and outline its integration into the QA process. We extend our first two case studies with another industrial one as well as with an investigation in an academic context to expand our initial empirical investigations by

1. investigating the accuracy of our smell detection including precision, recall, and relevance from a practical perspective,
2. analyzing which quality defects can be detected with smells and
3. gathering practitioner's feedback on how they would integrate smell detection in their QA process considering both formal and agile process environments.

1.2. Outline

The remainder of this paper is structured as follows. In Section 2, we describe previous work in the area. In Section 3, we define the concept of Requirements Smells and describe how we derived a set of Requirements Smells from ISO 29148. We introduce the tool realization in Section 4 and discuss the integration of smell detection in context of quality assurance in Section 5. In Section 6, we report on the empirical study that we set up to evaluate our approach, before concluding our paper in Section 7.

2. Related work

In the following, we discuss work relating to the concept of natural language processing and smells in general, followed by quality assurance in RE, before critically discussing currently open research gaps.

[1] In context of our studies, we use the ISO/IEC/IEEE 29148:2011 standard (ISO, IEC and IEEE, 2011) (in the following: *ISO 29148*) as basis for defining requirements quality. The standard supplies a list of so-called *Requirements Language Criteria*, such as loopholes or ambiguous adverbs, which we use to define eight smells (see also the smell definition in Section 3.2).

2.1. The notion of smells in software engineering

The concept of code smells was, to the best of our knowledge, first proposed by Fowler and Beck (1999) to answer the question at which point the quality of code is so low that it must be refactored. According to Fowler and Beck, the answer cannot be objectively measured, but we can look for certain concrete, visible symptoms, such as duplicated code (Fowler and Beck, 1999) as an indicator for bad maintainability (Juergens et al., 2009). This concept of smells, as well as the list that Fowler and Beck proposed, led to a large field of research. Zhang et al. (2011) provide an in-depth analysis of the state of the art in code smells. The metaphor of smells as concrete symptoms has since then been transferred to quality of other artifacts including (unit) test smells (van Deursen et al., 2001) and smells for system tests in natural language (Hauptmann et al., 2013). Ciemniewska et al. (2007), further characterize different defects of use cases through the term use case smell. In our work, we extend the notion of smells to the broader context of requirements engineering and introduce a concrete definition for the term *Requirements Smell*.

2.2. Quality assurance of software requirements

The concept of Requirements Smells is located in the context of RE quality assurance (QA), which is performed either manually or automatically.

2.2.1. Manual QA

Various authors have worked on QA for software requirements by applying manual techniques. Some put their focus on the classification of quality into characteristics (Davis et al., 1993), others develop comprehensive checklists (Anda and Sjøberg, 2002; Berry et al., 2006; Lamsweerde, 2009; Kamsties and Peach, 2000; Kamsties et al., 2001). Regarding QA, some develop constructive QA approaches, such as creating new RE languages, e.g. Denger et al. (2003), to prevent issues up front, others develop approaches to make analytic QA, such as reviews, more effective (Shull et al., 2000). In a recent empirical study on analytical QA, Parachuri et al. (2014) manually investigate the presence of defects in use cases. To sum it up, these works on manual QA provide analytical and constructive methods, as well as (varying) lists for defects. They strengthen our confidence that today's requirements artifacts are vulnerable to quality defects.

2.2.2. Automatic QA

Various publications discuss the automatic detection of quality violations in RE. We summarize existing approaches and tools, their publications, and empirical evaluations in Table 2. We also created an in-depth analysis of in total 27 related publications evaluating which quality defects or smells the approaches opt for in their described detection. In the following, we will first explain two related areas (automatic QA for redundancy and for controlled languages), before discussing automatic QA for ambiguity in general. For ambiguity, we first describe those approaches that conducted empirical evaluations of precision or recall of quality defects related, but not identical to, the ones of ISO 29148. Afterwards, we focus on publications that mention the same criteria as in the ISO 29148 (see Table 1 for this list and their respective empirical evaluations) and discuss the chosen approaches and results. We publish the complete list of each quality defect that is detected by each of the 27 papers, as well as the precision and recall (where provided), online as supplementary material (Femmer et al., 2015).

2.2.3. Automatic QA for redundancy

One specific area of QA is avoiding redundancy and cloning. Whereas Juergens et al. (2010) use ConQAT to search for syntactic

Table 1
Related work on criteria of ISO-29148 standard, detailed supplementary material can be found online (Femmer et al., 2015).

	ARM	QuARS				RQA	SREE	Smella	RETA
	(Wilson et al., 1997)	(Fabbrini et al., 2001b)	(Fabbrini et al., 2001a)	(Fantechi et al., 2003)	(Bucchiarone et al., 2005)	(Génova et al., 2011)	(Tjong and Berry, 2013)	(Femmer et al., 2014b)	(Arora et al., 2015)
Ambiguous Adv. & Adj.								E/Q	
Comparatives								E/Q	
Loopholes (or Options)	Q	E/Q	E/Q	E	E		Q*/P*	E/Q	
Negative Terms						O		E/Q	
Non-Verifiable Terms								E/Q	
Pronouns						O	Q*/P*	E/Q	O
Subjectivity		E/Q	E/Q	E	E			E/Q	
Superlatives								E/Q	

O = No empirical analysis, E = Examples from Case, Q = Quantification, P = Precision analyzed, R = Recall analyzed, * = Aggregated over multiple smells.

Table 2
Related approaches and tools, and their evaluation, detailed supplementary material can be found online (Femmer et al., 2015).

Tool/Approach	Purpose (unless ambiguity det.)	Publications	Evaluation	Precision	Recall
ConQAT	Redundancy	(Juergens et al., 2010)	E/Q/P	0.27–1	–
(Falessi)	Redundancy	(Falessi et al., 2013)	Q/P/R	up to 96	up to 96
ReqAlign	Redundancy	(Rago et al., 2014)	Q/P/R	0.63	0.86
RETA	Structured Language Rules	(Arora et al., 2015)	E/Q/P/R	0.85–0.94	0.91–1
AQUSA	User Story Rules	(Lucassen et al., 2015)	E/Q/P	0.63–1	–
CIRCE	Structured Language Rules	(Gervasi and Nuseibeh, 2002; Ambriola and Gervasi, 2006)	E	–	–
(Ciemniewska)		(Ciemniewska et al., 2007)	E	–	–
(Kof)		(Kof, 2007a)	E/Q	–	–
(Kiyavitskaya)		(Kiyavitskaya et al., 2008)	E/Q	–	–
RESI		(Körner and Brumm, 2009a, 2009b, 2009c)	E/Q	–	–
HeRA		(Knauss and Flohr, 2007; Knauss et al., 2009)	E	–	–
Alpino		(De Bruijn and Dekkers, 2010)	E/Q	–	–
(Chantree)		(Chantree et al., 2006)	E/P/R	0.6–1	0.02–0.58
Gleich		(Gleich et al., 2010)	E/Q*/P*/R*	0.34–0.97	0.53–0.86
(Krisch)		(Krisch and Houdek, 2015)	E/Q/P	0.12	–
ARM	RE Artifact Metrics	(Wilson et al., 1997)	Q	–	–
QuARS / SyTwo		(Fabbrini et al., 2001b, 2001a; Fantechi et al., 2003; Bucchiarone et al., 2005)	E/Q	–	–
RQA		(Génova et al., 2011)	O	–	–
SREE		(Tjong and Berry, 2013)	Q*/P*	0.66–0.68*	–
Smella		(Femmer et al., 2014b)	E/Q	–	–

O = No empirical analysis, E = Examples from Case, Q = Quantification, P = Precision analyzed, R = Recall analyzed, * = Aggregated over multiple smells.

identity resulting from a copy-and-paste reuse, Falessi et al. (2013) aim at detecting similar content, therefore using methods from information retrieval (such as Latent Semantic Analysis (Lucia et al., 2007)). Rago et al. (2014) extend this work specifically for use cases. Their tool, ReqAlign, classifies each step with a semantic abstraction of the step. These publications analyze the performance of their approaches, and depending on the artifact and methods achieve precision and recall close to 1 (see Table 2).

2.2.4. Automatic QA for controlled languages

Another specific area is the application of controlled language and the QA of controlled language. RETA (Arora et al., 2015) specifically analyzes requirements that are written via certain requirements patterns (such as with the EARS template (Mavin et al., 2009)). Their goal is to detect both conformance to the template but also some of the ambiguities as defined by Berry et al. (2003). The authors report on a case study where they look at the template conformance in depth, indicating that template conformance can be classified with various NLP suites to a high accuracy (Precision $>$ 0.85, Recall $>$ 0.9), both with and without glossaries. However, the performance of ambiguity detection (such as the detection of pronouns) is not further discussed in the publication. Similarly, AQUSA (Lucassen et al., 2015) analyzes requirements written in user story format (cf. Cohn (2004) for a detailed introduction into user stories), and detects various defects, such as missing rationales, where they achieve a precision of 0.63–1. Circe (Ambriola and Gervasi, 2006; Gervasi and Nuseibeh, 2002) is a further tool that assumes that requirements are written in such requirements patterns and detects violations of context- and domain-specific quality characteristics by building logical models. The authors report on six exemplary findings, which were detected in a NASA case study. However, despite their value to automatic QA, such approaches require very specific requirements structure.

2.2.5. Automatic QA for ambiguity in general

The remaining approaches listed in Table 2 aim at detecting ambiguities in unconstraint natural language. Since the quality defects detected by the approaches by Ciemniewska et al. (2007), Kof (2007b), HeRA by Knauss and Flohr (2007), Knauss et al. (2009), Kiyavitskaya et al. (2008), RESI by Körner and Brumm (2009a), Körner and Brumm (2009b), Körner and Brumm (2009c), and Alpino by De Bruijn and Dekkers (2010) are not the ones discussed in ISO 29148 and since we could not find an evaluation of precision and recall of these approaches, we omit discussing these

approaches in-depth here. An analysis of what these approaches focus on in detail as well as their evaluation can be found in short in Table 2 and in full length in our supplementary material online (Femmer et al., 2015). In the following, we first report on those publications that focus on criteria different from ISO 29148, but which report precision or recall. Afterward, we describe publications that aim at detecting quality violations of ISO 29148 (see Table 1).

First, Chantree et al. (2006) target the specific grammatical issue of coordination ambiguity (detecting problems of ambiguous references between parts of a sentence), mostly through statistical methods, such as occurrence and co-occurrence of words. In a case study, they report on a precision of their approach mostly between 54% and 75%. even though they do not explicitly differentiate between the detected ambiguities and the concept of pronouns. Second, Gleich et al. (2010) base their approach on the ambiguity handbook, as defined by Berry et al. (2003), as well as company-specific guidelines. They compare their dictionary- and POS-based approach against a gold standard which they created by letting people highlight ambiguities in requirements sentences. The gold standard deviates substantially, however, from what is considered high quality in their guidelines. Therefore, they create an additional gold standard, mostly based on the guideline rules. Consequently, their precision[2] varies between 34% for the pure experts opinion, and 97% for a more guideline-based gold standard. Third, Krisch and Houdek (2015), focus on the detection of passive voice and so-called weak words. They present their dictionary- and POS-based approach to practitioners and find many false positives, similar to our RQ 3. In average, a precision of 12% is reported for the weak words detection. These approaches focus on very related, but not identical quality violations or smells.

2.2.6. Automatic QA for ISO 29148 criteria

Lastly, we specifically focus on those approaches that report to detect the criteria from the ISO 29148 standard. Table 1 provides an overview of these works and their respective evaluations.

The ARM tool (Wilson et al., 1997) defines quality in terms of the (now superseeded) IEEE 830 standard (IEEE Computer Society, 1998) and proposes generic metrics, instead of giving feedback directly to requirements engineers. The metrics are calculated through counting how often a set of pre-defined terms (per metric) occurs in a document, including a metric of what we call Loopholes. Even though they report on a case study with 46 specifications from NASA, only a quantitative overview is reported.[3] The QuARS tool (Fabbrini et al., 2001a, 2001b) is based on the author's experience. Bucchiarone et al. (2005) describe the use of QuARS in a case study with Siemens and show some exemplary findings. SyTwo (Fantechi et al., 2003) adopts the quality model of QuARS and applies it to use cases. Loopholes and Subjectivity are parts of the QuARS quality model. Also RQA is built on a different, proprietary quality model, as described by Génova et al. (2011), which includes negative terms as well as pronouns as quality defects. These works also built upon extending natural language with NLP annotations, such as POS tags and searching through dictionaries for certain problematic phrases. However, we could not find a detailed empirical investigation of these tools, e.g. with regards to precision and recall. SREE is an approach by Tjong and Berry (2013), which aims at detection of ambiguities with a recall of 100%. Therefore, they completely avoid all NLP approaches (since they come with imprecision), and build large dictionaries of words. The tool includes detection of loopholes, as well as pronouns; however, they report only on an aggregated precision for all the different types of ambiguities (66–68%) from two case studies. In our previous paper (Femmer et al., 2014b), we searched for violations of ISO 29148, yet we provided only a quantitative analysis, as well as qualitative examples. As mentioned before, RETA also issues warnings for pronouns, however, the evaluation in their paper (Arora et al., 2015) focusses on template conformance.

2.3. Discussion

Previous work has led to many valuable contributions to our field. To explore open research gaps, we now critically reflect on previous contributions from an evaluation, a quality definition and a technical perspective.

First, one gap in existing automatic QA approaches is the lack of empirical evidence, especially under realistic conditions. Only few of the introduced contributions were evaluated using industrial requirements artifacts. Those who do apply their approach on such artifacts focus on quantitative summaries explaining which finding was detected and how often it was detected. Some authors also give examples of findings, but only few works analyze this aspect in depth with precision and recall, especially in the fuzzy domain of ambiguity (see Table 2). When looking at the characteristics that are described in ISO 29148, we have not seen a quantitative analysis of precision and recall. Furthermore, reported evidence does not include qualitative feedback from engineers who are supposed to use the approach, which could reveal many insights that cannot be captured by numbers alone. However, we postulate that the accuracy of quality violations very much depends on the respective context. This is especially true for the fuzzy domain of natural language where it is important to understand the (context-specific) impact of a finding to rate its detection for appropriateness and eventually justify resolving the issue.

Second, the existing approaches are based on proprietary definitions of quality, based on experience or, sometimes, simply on what can be directly measured. The ARM tool (Wilson et al., 1997) is loosely based on the IEEE 830 (IEEE Computer Society, 1998) standard. However, as the recent literature survey by Schneider and Berenbach (2013) states: *"the ISO/IEC/IEEE 29148:2011 is actually the standard that every requirements engineer should be familiar with"*. We are not aware of an approach that evaluates the current ISO 29148 standard (ISO, IEC and IEEE, 2011) in this respect. As Table 1 shows, for most language quality defects of ISO 29148, there has not yet been a tool to detect these quality defects. To all our knowledge, for neither of these factors, there is an differentiated empirical analysis of precision and recall. Yet, many other quality models (most notably from the ambiguity handbook by Berry et al., 2003) and quality violations could lead to Requirements Smells, as far as they comply with the definition given in the next section.

Finally, taking a more technical perspective, our Requirements Smell detection approach does not fundamentally differ from existing approaches. Similar to previous works, we apply existing NLP techniques, such as lemmatization and POS tagging, as well as dictionaries. For the rules of the ISO 29148 standard, no parsing or ontologies (as used in other approaches) were required. However, to detect superlatives and comparatives in German, we added a morphological analysis, which have not yet seen in related work.

In summary, in our contribution, we extend the current state of reported evidence on automatic QA for requirements artifacts via systematic studies in terms of distribution, precision, recall, and relevance, as well as by means of a systematic evaluation with practitioners under realistic conditions. We perform this on both existing, as well as new quality defects taken from the ISO 29148. Therefore, we extend our previously published first empirical steps

[2] Gleich et al. calculate their metrics based on the combination of all ambiguities; unfortunately, they do not differentiate, e.g. by the type of ambiguity. Also, to our knowledge, the gold standard does not differentiate between the types. This prevents a direct comparison to their work.

[3] See also our RQ 1 in Section 6.

(Femmer et al., 2014b) to close these gaps by thorough empirical evaluation.

3. Requirements Smells

We first introduce the terminology on Requirements Smells as used in this paper. In a second step, we define those smells we derived from ISO 29148 and which we use in our studies, before describing the tool realization in the next section.

3.1. Requirements Smell terminology

Code smells are supposed to be an imprecise indication for bad code quality (Fowler and Beck, 1999). We apply this concept of smells to requirements and define it as follows: A Requirements Smell is an indicator of a quality violation, which may lead to a defect, with a concrete location and a concrete detection mechanism. In detail, we consider a smell as having the following characteristics:

1. A Requirements Smell is an *indicator* for a quality violation of a requirements artifact. For this definition, we understand requirements quality in terms of quality-in-use, meaning that bad requirements artifact quality is defined by its (potential) negative effects on activities in the software lifecycle that rely on these requirements artifacts (see also Femmer et al., 2015).
2. A Requirements Smell does *not necessarily lead to a defect* and, thus, has to be judged by the context (supported e.g. by (counter-/)indications). Whether a Requirements Smell finding is or is not a problem in a certain context must be individually decided for that context and is subject to reviews and other follow-up quality assurance activities.
3. A Requirements Smell has a *concrete location* in an entity of the requirements artifact itself, e.g. a word or a sequence. Requirements Smells always provide a pointer to a certain location that QA must inspect. In this regard, it differs from general quality characteristics, e.g. completeness, that only provide abstract criteria.
4. A Requirements Smell has a *concrete detection mechanism*. Due to its concrete nature, Requirements Smells offer techniques for detection of the smells. These techniques can, of course, be more or less accurate.

Furthermore, we define a *quality defect* as a concrete instance or manifestation of a quality violation in the artifact, in contrast to a *finding* which is an instance of a smell. However, like a smell indicates for a quality violation, the finding indicates for a defect. Fig. 1 visualizes the relation of these terms.

In the following, we will focus on natural language Requirements Smells, since requirements are mostly written in natural language (Mich et al., 2004). Furthermore, the real benefits of smell detection in practice should come with automation. Therefore, the remainder of the paper discusses only Requirements Smells where the detection mechanism can be executed automatically (i.e. it requires no manual creation of intermediate or supporting artifacts).

3.2. Requirements Smells based on ISO 29148

We develop a set of Requirements Smells based on an existing definition of quality. For the investigations in scope of this paper, we take the ISO 29148 requirements engineering standard (ISO, IEC and IEEE, 2011) as a baseline. The reasons for this are two-fold.

First, the ISO 29148 standard was created to harmonize a set of existing standards, including the IEEE 830:1998 (IEEE Computer Society, 1998) standard. It differentiates between quality characteristics for a set of requirements, such as completeness or consistency,

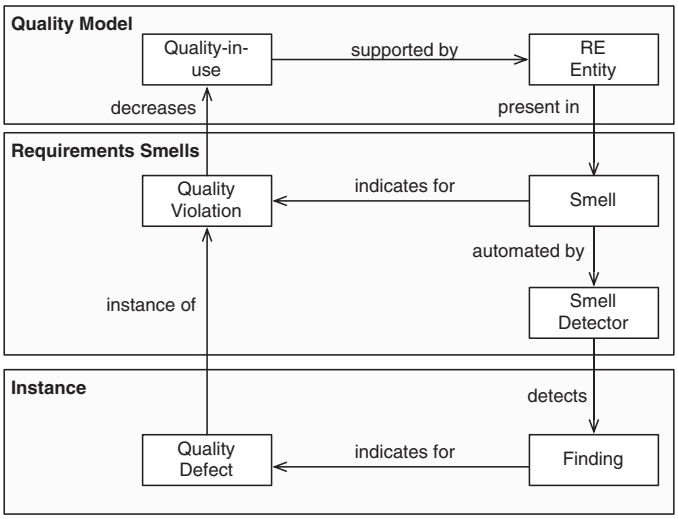


Fig. 1. Terminology of Requirements Smells (simplified).

and quality characteristics for individual requirements, such as unambiguity and singularity. The standard furthermore describes the usage of requirements in different project phases and gives exemplary contents and structure for requirements artifacts. Therefore, we argue that this standard is based on a broad agreement and acceptance. Recent literature studies come to the same conclusion (Schneider and Berenbach, 2013).

Second, the standard provides readers with a list of so-called *requirements language criteria* which support the choice of proper language for requirements artifacts. The authors of the standard argue that violating the criteria results *"in requirements that are often difficult or even impossible to verify or may allow for multiple interpretations"* (ISO, IEC and IEEE, 2011, p.12). For defining our smells, which we describe next, we refer to this section of the standard and use all the defined requirements language criteria. We employ those criteria as a starting point and define the smells by adding the affected entities (e.g. a word) and an explanation. Here, we do not discuss the impact smells have on the quality-in-use. Essentially, smells hinder the understandability of requirements and consequently their subsequent handling and their verification (for a richer discussion, see also previous work in Femmer et al. (2015)).

Our current understanding is based on the examples given by the standard. A subset of the language criteria, namely `Subjective Language`, `Ambiguous Adverbs and Adjectives` and `Non-verifiable Terms`, as defined in ISO, IEC and IEEE (2011), are strongly related, essentially since subjective language is a special type of ambiguity, which may lead to issues during verification. Since the intention of this work is to start with the standard as a definition of quality, in the following, we will remain with the provided definition based on the language criteria and leave the development of a precise and complete set of Requirements Smells to future work. In detail, we use the requirements language criteria to derive the smells summarized next.

Smell Name:	`Subjective Language`
Entity:	Word
Explanation:	Subjective Language refers to words of which the semantics is not objectively defined, such as *user friendly, easy to use, cost effective.*
Example:	The architecture as well as the programming must ensure a **simple** and **efficient** maintainability.

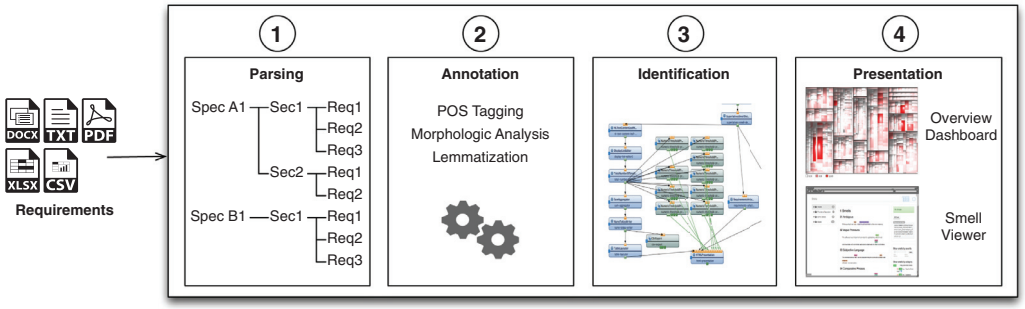


Fig. 2. The overall smell detection process.

Smell Name:	`Ambiguous Adverbs and Adjectives`
Entity:	Adverb, Adjective
Explanation:	Ambiguous Adverbs and Adjectives refer to certain adverbs and adjectives that are unspecific by nature, such as *almost always, significant* and *minimal*.
Example:	If the (…) quality is **too low**, a fault must be written to the error memory.

Smell Name:	`Loopholes`
Entity:	Word
Explanation:	Loopholes refer to phrases that express that the following requirement must be fulfilled only to a certain, imprecisely defined extent.
Example:	**As far as possible**, inputs are checked for plausibility.

Smell Name:	`Open-ended, Non-verifiable Terms`
Entity:	Word
Explanation:	Open-ended, non-verifiable terms are hard to verify as they offer a choice of possibilities, e.g. for the developers.
Example:	The system may only be activated, if all required sensors (…) work with **sufficient** measurement accuracy.

Smell Name:	`Superlatives`
Entity:	Adverb, Adjective
Explanation:	Superlatives refer to requirements that express a relation of the system to all other systems.
Example:	The system must provide the signal in the **highest** resolution that is desired by the signal customer.

Smell Name:	`Comparatives`
Entity:	Adverb, Adjective
Explanation:	Comparatives are used in requirements that express a relation of the system to specific other systems or previous situations.
Example:	The display (…) contains the fields A, B, and C, as well as **more exact** build infos.

Smell Name:	`Negative Statements`
Entity:	Word
Explanation:	Negative Statements are "statements of system capability not to be provided"(ISO, IEC and IEEE, 2011). Some argue that negative statements can lead to underspecification, such as lack of explaining the system's reaction on such a case.
Example:	The system must **not** sign off users due to timeouts.

Smell Name:	`Vague Pronouns`
Entity:	Pronoun
Explanation:	Vague Pronouns are unclear relations of a pronoun.
Example:	The software must implement services for applications, **which** must communicate with controller applications deployed on other controllers.

Smell Name:	`Incomplete References`
Entity:	Text reference
Explanation:	Incomplete References are references that a reader cannot follow (e.g. no location provided).
Example:	*[1] "Unknown white paper". Peter Miller.*

4. Smella: a prototype for Requirements Smell detection

Requirements Smell detection, as presented in this paper, serves to support manual quality assurance tasks (see also the next section). The smell detection is implemented on top of the software quality analysis toolkit ConQAT,[4] a platform for source code analysis, which we extended with the required NLP features. In the following, we introduce the process for the automatic part of the approach, i.e. the detection and reporting of Requirements Smells. To the best of our knowledge, there is no tool, other than the ones mentioned in related work, that detect and present these smells in natural language requirements documents.

The process takes requirements artifacts in various formats (MS Word, MS Excel, PDF, plain text, comma-separated values) and consists of four steps (see also Fig. 2):

1. *Requirements parsing* of the requirements artifacts into single items (e.g. sections or rows), resulting in plain texts, one for each item
2. *Language annotation* of the requirements with meta-information
3. *Findings identification* in the requirements, based on the language annotations
4. *Presentation* of a human-readable visualization of the findings as well as a summary of the results

The techniques behind these steps are explained in the following section.

4.1. Requirements parsing

Our current tool is able to process several file formats: MS Word, MS Excel, PDF, plain text and comma-separated values (CSV). Depending on the format, the files are parsed in different ways. Plain text and PDF are taken as is and parsed file by file. Microsoft Word files are grouped by their sections. For Microsoft Excel and CSV files, we define those columns that represent the IDs or names (if there are any), and those columns should be used as text input to detect smells.

If a file is written in a known template, such as a common template for use cases, we can make use of this template to understand structural defects, such as lacking content items in a template. In the remainder of this paper, however, we focus on the natural language Requirements Smells as provided by the ISO standard.

4.2. Language annotation

For the annotation and smell detection steps, we employ three techniques from Natural Language Processing (NLP) (Jurafsky and Martin, 2014). Table 3 additionally shows which of the techniques we use for which smell.

[4] http://www.conqat.org .

Table 3
Detection techniques for smells.

Smell name	Detection mechanism
Subjective Language	Dictionary
Ambiguous Adverbs and Adjectives	Dictionary
Loopholes	Dictionary
Open-ended, non-verifiable terms	Dictionary
Superlatives	Morphological analysis or POS tagging
Comparatives	Morphological analysis or POS tagging
Negative Statements	POS tagging and dictionary
Vague Pronouns	POS tagging: Substituting pronouns.
Incomplete References	Not in scope of this study

POS tagging: For two smells, we use part-of-speech (POS) tagging. Given a sentence in natural language, it determines the role and function of each single word in the sentence. The output is a so-called *tag* for each word indicating, for instance, whether a word is an adjective, a particle, or a possessive pronoun. We used the Stanford NLP library (Toutanova et al., 2003) and the RFTagger (Schmid and Laws, 2008) for this. Both are statistical, probabilistic taggers that train models similar to Hidden Markov Models based on existing databases of tagged texts. A detailed introduction into the technical details of POS tagging is beyond the scope of this paper but can be found, for example, in Jurafsky and Martin (2014). We use POS tagging to determine so-called substituting pronouns. These are pronouns that do not repeat the original noun and, thus, need a human's interpretation of its dependency.

Morphological analysis: Based on POS tagging, we perform a more detailed analysis of text and determine a word's inflection. This includes, inter alia, determining a verb's tense or an adjective's comparison. We use this technique to analyze if adjectives or adverbs are used in their comparative or superlative form.

Dictionaries & lemmatization: For the remaining five smells, we use dictionaries based on the proposals of the standard (ISO, IEC and IEEE, 2011) and on our experiences from first experiments in a previous work (Femmer et al., 2014b). We furthermore apply lemmatization for these words, which is a normalization technique that reproduces the original form of a word. In other words, if a lemmatizer is applied to the words *were*, *is* or *are*, the lemmatizer will return for all three the word *be*. Lemmatization is in its purpose very similar to stemming (see, e.g. the Porter Algorithm (Porter, 1980)), yet not based on heuristics but on the POS tag as well as the word's morphological form. For Requirements Smells, the difference is significant: For example, the words *use* and *useful* stem to the same word origin (*use*), but to different lemmas (i.e. meanings; *use* and *useful*). Whereas the lemma *use* is mostly clear to all stakeholders, the lemma *useful* is easily misinterpreted.

4.3. Findings identification

Based on the aforementioned information, we identify findings. This step actually finds the parts of an artifact that exhibit bad smells. Dependent on the actual smell, we use different techniques, as shown in Table 3. If the smell relates to a grammatical aspect, we search through the information from POS tagging and morphological analyses. For example, for the Superlatives Smell, we report a finding if an adjective is, according to morphologic analysis, inflected in its superlative form. If the smell does not relate to grammatical aspects but rather the semantics of the requirements, we identify the smell by matching the lemma of a word against a set of words from pre-defined dictionaries. Since the requirements under analysis in our cases did not contain references, incomplete references are not part of our tool at present.

4.4. Findings presentation

We implemented the presentation of findings in a prototype, which we call *Smella* (Smell Analysis). Smella is a web-based tool that enables viewing, reviewing and blacklisting findings as well as a hotspot analysis at an artifact level. In the Smella presentation, we display the complete requirements artifact and annotate findings in a spell checker style. This follows the idea of smells as only indications that must be evaluated in their context. Lastly, the system gives detailed information when a user hovers a finding (see Fig. 3). In the following, we shortly describe the features of Smella

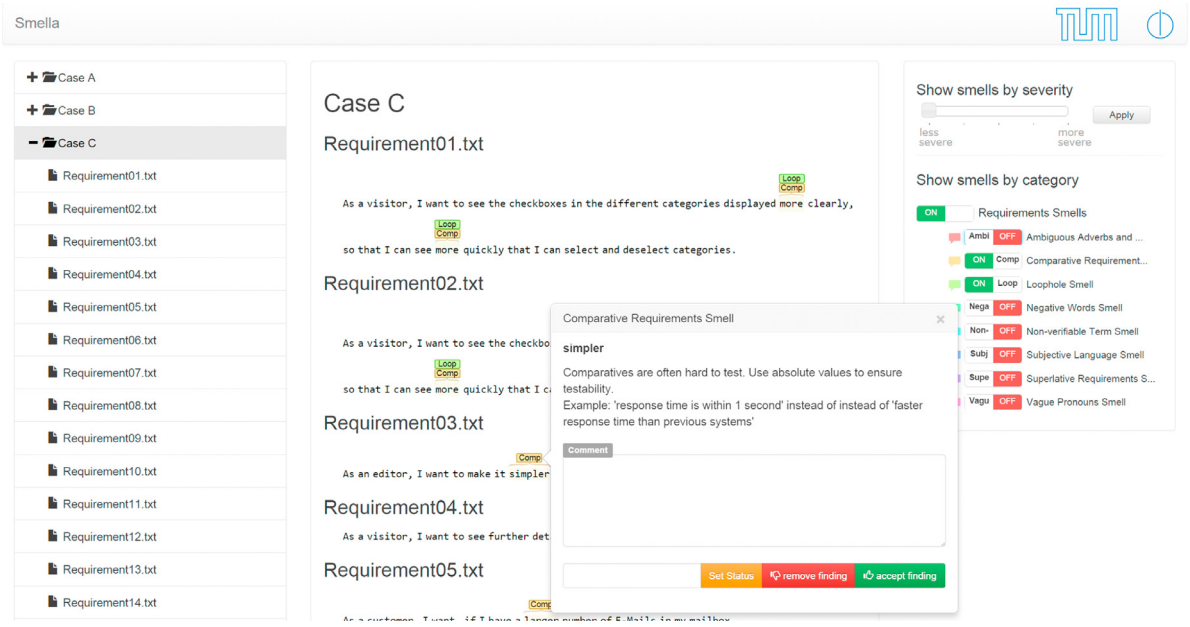


Fig. 3. A sample output from the smell detection tool (detailed artifact view) with some smells disabled and some findings blacklisted.

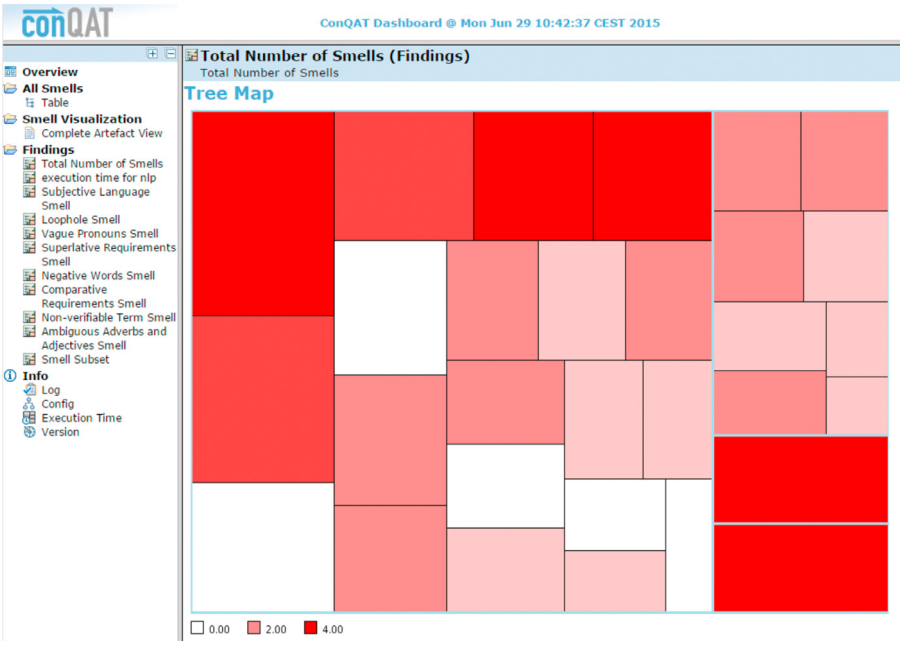


Fig. 4. A sample output from the smell detection tool (hotspot analysis view).

in detail to provide the reader with a rough understanding of the prototype.

View findings: At the level of a single artifact, we present the text of the artifact and its structure. We mark all findings in the text. With a click on the markers, more information about the finding is displayed. The tool provides an explanation of the rationale behind this smell and possible improvements for the finding depending on the smell (every smell has a message for improvements).

Review findings: We allow the user to write a review and to set a status for each finding, both supporting feedback mechanisms within and between project teams. A user has the possibility to accept or reject a finding but also to set a custom state, for example *under review*. Accepting a finding means the finding needs to be addressed. If a finding is rejected, the finding does not need to be addressed. The semantics of the custom status is open to the reviewer.

Blacklist findings: Smells are only indicators for issues. Therefore, users can reject findings. If a finding is rejected by the user, the finding is removed from the visualization and will not be presented to the user anymore.

Disable smells: Often, users are interested in only a subset of smells or even just one smell. Therefore, we allow the user to hide all findings of particular smells and to select the smells she wants to display in the artifact view.

Analyze hotspots: In this view, we present all artifacts in a colored treemap (see Fig. 4). Every box in the treemap is one artifact, with the color of the box indicating the number of findings: the more red an artifacts is, the more findings it contains (the more it "smells" bad). The artifacts are grouped by their folder structure. The tool provides a summarized treemap for all smells as well as a separate treemap for all individual smells. With these treemaps, users can identify artifacts or groups of artifacts exhibiting a high number of findings – for one single smell but also for all smells together. This feature supports the identification of candidates for in-depth reviews.

5. Requirements Smell detection in the process of quality assurance

The Requirements Smell detection approach described in previous sections serves the primary purpose of supporting quality assurance in RE. The detection process itself is, however, not restricted to particular quality assurance tasks, nor does it depend on a particular (software) process model as we will show in Section 6. Hence, a smell detection, similar to the notion of quality itself, always depends on the views in a socio-economic context. Thus, how to integrate smell detection into quality assurance needs to be answered according to the particularities of that context. In the following, we therefore briefly outline the role smell detection can generally take in the process of quality assurance. More concrete proposals on how to integrate it into specific contexts are given in our case studies in Section 6.

We postulate the applicability of the Requirements Smell detection in the process of both constructive and analytical quality assurance (see Fig. 5). From the perspective of a constructive quality assurance, authors can use the smell detection to increase their awareness of potential smells in their requirements artifacts and to remove smells before releasing an artifact for, e.g., an inspection. External reviewers in turn, can then use the smell detection to prepare analytical, potentially cost-intensive, quality assurance tasks, such as a Fagan inspection (Fagan, 2002). Such an inspection involves several reviewers and would benefit from making potential smells visible in advance. Iterative inspection approaches are also known as phased inspections, as defined by Knight and Myers (1993).

We furthermore believe that one major advantage is that the scope of our smell detection is not to enforce resolving a potential smell but to increase the awareness of the like and to make transparent later reasoning why certain decisions have been taken. Please note that two different roles (e.g. requirements engineer and QA engineer) can take two different viewpoints on the same smell, respectively its criticality and whether it should be resolved or not. In addition, a finding could be unambiguous to the author, but unclear to the target group of readers (represented by the reviewers). Therefore, one contribution of our tool-supported smell detection

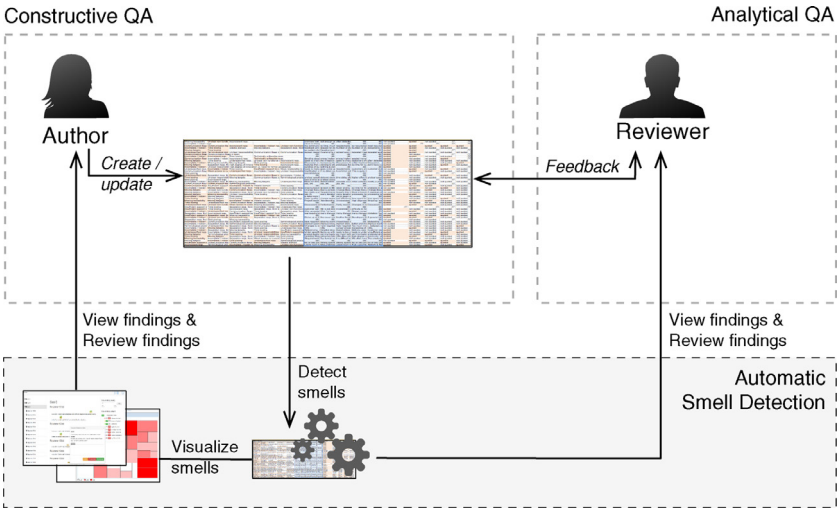


Fig. 5. A suggestion for applying Requirements Smell detection in QA.

is also to actively foster the communication between reviewers and authors and to enable continuous feedback between both roles. For this reason, we enable stakeholders in Smella to comment on detected smells and make explicit whether they need to be resolved or whether and why they have been accepted or rejected.

6. Evaluation

For a better, empirical understanding of smells in requirements artifacts, we conducted an exploratory multi-case study with both industrial and academic cases. We particularly rely on case study research over other techniques, such as controlled experiments, because we want to evaluate our approach in practical settings under realistic conditions. For the design and reporting of the case study, we largely follow the guidelines of Runeson and Höst (2008).

6.1. Case study design

Our overall research objective is as follows:

Research objective: Analyze whether automatic analysis of Requirements Smells helps in requirements artifact quality assurance.

To reach this aim, we formulate four research questions (RQ). In the following, we introduce those research questions, the procedures for the case and subjects selection, the data collection and analysis, and the validity procedures.

6.1.1. Research questions

RQ 1: How many smells are present in requirements artifacts? To see if the automatic detection of smells in requirements artifacts could help in QA, we first need to verify that Requirements Smells exist in the real world. The answer to this question fosters the understanding how widespread the smells under analysis are in industrial and academic requirements artifacts.

RQ 2: How many of these smells are relevant? Not only the number of detected smells is important. If many of the detected smells are false positives and not relevant for the requirements engineers and developers, it would hinder QA more than it would help. As relevancy is a rather broad concept, we break down RQ 2 into two sub-questions.

RQ 2.1: How accurate is the smell detection? The first sub-question looks at the more technical view on relevance. We want to find false positives and false negatives to determine the precision and recall of the analysis in terms of correct detection of the defined smell.

RQ 2.2: Which of these smells are practically relevant in which context? This second sub-question is concerned with practical relevance. We investigate whether practitioners would react and change the requirement when confronted with the findings.

RQ 3: Which requirements quality defects can be detected with smells? After we understood how relevant the analyzed Requirements Smells are, we want to understand their relation to existing quality defects in requirements artifacts. Hence, we need to check whether, and if so, which defects in requirements artifacts correspond to smells, as we understand smell findings as indicators for defects.

RQ 4: How could smells help in the QA process? Finally, we collect general feedback from practitioners whether (and how) smell detection could be a useful addition to QA for requirements artifacts and whether as well as how they would integrate the smell detection into their QA process.

6.1.2. Case and subjects selection

Our case and subject selection is opportunistic but in a way that maximizes variation and, hence, evaluates the smell detection in very different contexts. This is particularly important for investigating requirements artifacts under realistic conditions, also due to the large variation in how these artifacts manifest themselves in practice. A prerequisite for our selection is the access to the necessary data. To get a reasonable quantitative analysis of the number of smells (RQ 1) and qualitative analysis of the relation of smells and defects (RQ 3), we complement our three industrial cases with a case in an academic setting. There, various student teams are asked to provide software with a certain set of (identical) functionality for a customer as part of a practical course. This is also a realistic setting but provides us with a higher number of specifications and reviews than in the industrial cases.

We will refer to the subjects of the industrial cases as *practitioners* and we will call the latter subjects *students*.

6.1.3. Data collection procedure

We used a 6-step procedure to collect the data necessary for answering the research questions.

1. *Collect requirements artifact(s) for each case.* We retrieved the requirements artifacts to be analyzed in each case. For one case, the requirements were stored in Microsoft Word Documents. For the other cases, this involved extracting the requirements

from other systems, either a proprietary requirements management tool (resulting in a list of html files), or the online task management system JIRA, which led to a set of comma-separated values files. For the student projects, the students handed in their final artifacts either as a single PDF or as a PDF with the general artifact and another PDF with the use cases. Where authors explicitly structured requirements in numbered requirements, user stories or use cases, we counted these artifacts.

2. *Run the smell detection via Smella.* We applied our detection tool as introduced in Section 4.4 on the given requirements artifacts, which generated a list of smells per artifact.
3. *Classify false positives.* For all cases in which we wanted to present our results to practitioners, we reviewed each detected finding. In pairs of researchers, we classified the findings as either true or false positive. We classified a finding as false positive if the finding was not an instance of the smell, e.g. because the results of the linguistic analysis was incorrect.[5] For artifacts containing more than 10 findings of a smell, we only inspected a set of 10 random findings (of that smell) per artifact. The same holds for Case D, where we inspected 10 random findings of each category for the whole case.
4. *Inspect documents for false negatives.* To calculate the recall of the smell detection, for each case we randomly selected one artifact that a pair of researchers inspected for false negatives. To ease the manual inspection, we grouped the smells `Subjective Language`, `Ambiguous Adverbs and Adjectives`, `Loopholes`, `Non-verifiable Terms` (as `Ambiguity-related smells`). We classified whether a finding is a true or false negative based on the same conditions as in the previous step.
   One common cause for false negatives for dictionary-based smells can be that an ambiguous phrase is not part of the dictionary. Since we developed the dictionaries based on existing dictionaries, such as the standard, these dictionaries are not yet complete and must be further developed. However, since this is an issue that is not a problem of the smell detection approach in general, but rather a configuration task, we did not take these findings into consideration for the recall.
5. *Get rating by practitioners.* We selected a subset of the true positive findings so that we cover all smells with a minimum of two findings per smell as far as the artifacts allowed. When we found repeating or similar findings, e.g. multiple similar sentences with the same smell, we also included one of these findings into the set.
   We presented this subset to the practitioners and interviewed them, finding by finding, through three closed questions (see also Table 9): Q1: Would you consider this smell as relevant? Q2: Have you been aware of this finding before? Q3: Would you resolve the finding? Of these, the former two must be answered with *yes* or *no*. For the last question, we also needed to take the criticality into account. Therefore, in case practitioners answered that they would resolve a finding, we also asked whether they would resolve it immediately, in a short time (i.e. within this project iteration) or in a long time (e.g. if it happens again). In addition to these three questions, we took notes of qualitative feedback, such as discussions.
6. *Interview practitioners.* In addition to the ratings, we performed open interviews with practitioners about their experience with the smell detection and how they might include it in their quality assurance process. We took notes of the answers.
7. *Get review results from students.* Lastly, the students performed reviews of the artifacts of other student teams. They documented and classified found problems according to a checklist (see Table A.11) without awareness of the smell findings in their artifacts. We then collected the review reports from the students.

[5] For example, if the linguistic analysis incorrectly classified the word *provider* in the sentence *"As a provider, I want […]"* as a comparative adjective.

6.1.4. Analysis procedure

We structure our analysis procedure into seven steps. Each step leads to the results necessary for answering one of our research questions.

1. *Calculate ratios of findings per artifact*. To understand whether smells are a common issue in requirements artifacts, we compared the quantitative summaries of smells in the various artifacts and domains. To enable a comparison between different types of requirement artifacts, we used the number of words in each artifact as a measure of size. Hence, we finally reported the ratio of findings per 1000 words for each smell and all smells in total. This provided answers for RQ 1.
2. *Calculate ratios of findings for parts of user stories.* In one case, we had a common structure of the requirements, because they were formulated as user stories. To get a deeper insight into the distribution of smells and findings, we calculated the ratios of findings per 1000 words for each part. We divided the user stories into the parts *role* ("As a … ") *feature* ("I want to … ") and *reason* ("so that … ") using regular expressions. We counted the words and findings in each part. This provided further insights into the answer for RQ 1.
3. *Calculate ratios of false positives.* After a rough overview obtained under the umbrella of RQ 1 describing the number of findings for each smell of the varying artifacts, we wanted to better understand the smell's relevance. The first step was to calculate the ratios of false positive as we classified them in Step 3 of the data collection. We reported false positive rates overall and for each smell. This provides the first part of the answer to RQ 2.1.
4. *Calculate ratios of false negatives.* The precision of a smell detection is tightly coupled with the recall. Therefore, we calculated the ratio of detected smell findings to all existing findings, according to our manual inspection, as described in Step 4 of the data collection procedure. This provides the second part of the answer to RQ 2.1.
5. *Calculate ratio of irrelevant smells.* We were not only interested in errors in the linguistic analysis but also in how relevant the correct analyses were for the practitioners. Hence, we calculated and reported the ratios of findings considered irrelevant by the practitioners. This answers RQ 2.2.
6. *Compare defects from reviews with findings.* From the students, we received review reports for each artifact. As the effort to check them all would have been overwhelming, we took a random sample of 20% of the artifacts. For each of the defects detected in the review, we checked if there is a corresponding finding from a smell. This answers RQ 3.
7. *Interpret interview notes.* To answer finally RQ 4, we analyze the interview transcripts and code the answers given by the interviewees manually.

6.1.5. Validity procedure

First, we used peer debriefing in the sense that all data collection and analyses were done by at least two researchers. Analysis results were also checked by all researchers. This researcher triangulation especially increases the internal validity. Furthermore, we kept an audit trail in a Subversion system to capture all changes to documents and analyses.

Table 4
Study objects

Artifact	Topic	Size in words	# Requirements	# Use cases	# User stories
A1	Adaptive valve control	1896	91		
A2	Exhaust control	2244	72		
A3	Driving information	199	12		
A4	Engine startup control	975	44		
A5	Engine control	524	49		
A6	Powertrain communication	1100	55		
Sum Daimler		6938	323		
B1	Management of access control	2093	9	18	
B2	Event notification	1015	3	19	
B3	Document management	458	1	16	
Sum Wacker		3566	13	53	
C1	Webshop for fashion articles	5226			168
C2	CMS in transportation domain	2742			123
C3	CRM system	6863			230
C4	Webshop for hardware articles	13,124			561
Sum TechDivision		27,955			1082
Avg Stuttgart		4470		18.9	
Sum Stuttgart		227,973		966	
Sum over all		266,432	336	53	1082

Second, we performed all the classifications of findings into true and false positives in pairs. This already helped to avoid misclassifications. To further check our classifications, we afterwards did an independent re-classification of randomly selected 10% of the findings and calculated the inter-rater agreement. We discussed to clarify which findings we consider false positives and repeated the classifications until we reached an acceptable agreement. The same procedure held for the inspection of artifacts to detect false negatives, which we also conducted in pairs. Furthermore, we also independently re-classified one of the artifacts to understand the inter-rater agreement on the false negatives. Overall, our analysis for false positives and relevance of the findings is also a validity procedure in the sense that we check in RQ 2 the results from RQ 1.

Third, we discussed with the practitioners what relevance of smells means in the context of the study to avoid misinterpretations. Furthermore, we gave the students review guidelines to give them an indication what quality defects in requirements artifacts might be. Both serve in particular as mitigation to threats to the internal and the construct validity.

Fourth, we performed the analysis of the correspondence between smells and defects with a pair of researchers. This pair derived a classification of the found and not found defects. Both other researchers reviewed the classification, and we improved it iteratively until we reached a joint agreement.

Fifth, we performed member checking by showing our transcriptions and interpretations for RQ 4 to the interviewed practitioners and incorporating feedback.

Finally, to support the external validity of the results of our study, we aimed at selecting cases with maximum variation in their domains, sizes, and how they document requirements.

6.2. Results

In the following, we report on the results of our case studies. We first describe the cases and subjects under analysis, before we answer the research questions. We end by evaluating the validity of the cases.

6.2.1. Case and subjects description

The first three cases contain requirements produced in different industrial contexts: embedded systems in the automotive industry, business information systems for the chemical domain and agile development of web-based systems. While the first two represent rather classical approaches to Requirements Engineering, the third case applies the concept of user stories, as it is popular in agile software development. The fourth case is in an academic background and employs both use cases and textual requirements. Regarding subject selection, for each industrial case we selected practitioners involved in the company, domain and specification. We executed the findings rating (Step 5) and the interviews regarding the QA process (Step 6) with the same experts, so that their answer in Step 6 is based on their experience with practical, real examples. In the following, we describe the cases, as well as the experts or students for each case. Table 4 provides a quantitative overview of the cases.

Case A: Daimler AG. Daimler AG is a multinational automotive corporation headquartered in Stuttgart, Germany. At Daimler, we analyzed six different requirements artifacts (A1–A6) which were written by various authors. The requirements artifacts describe functionality in different domains of engine control as well as driving information. In this case, requirements are written down in the form of sentences, identified by an ID. The authors are domain experts who are coached on writing requirements.

The requirements artifacts A1–A6 consist of 323 requirements in total (see Table 4). All of the artifacts of Daimler analyzed in our study were created by domain experts in a pilot phase after a change in the requirements engineering process as part of a software process improvement endeavor. For RQ 2.2., we reviewed 22 findings with an external coach who works as a consultant for requirements engineering and has tightly collaborated with the group for many years.

Case B: Wacker Chemie AG. In the second case, we analyzed requirements artifacts of business information systems from Wacker Chemie AG. Wacker is a globally active company working in the chemical sector and headquartered in Munich, Germany. The systems that we analyzed fulfill company-internal purposes, such as systems for access to Wacker buildings or support systems for document management.

We analyzed three Wacker requirements artifacts that were written by five different authors. At Wacker, functional requirements are written as use cases (including fields for *Name, Description, Role* and *Precondition*) whereas non-functional requirements are described in simple sentences. The artifacts consisted of 53 use cases and 13 numbered requirements (see Table 4). For the reviews of the findings in RQ 2.2, we selected 18 findings and discussed

Table 5
Study objects usage in research questions.

Case	RQ 1: distribution	RQ 2.1: precision	RQ 2.1: recall	RQ 2.2: relevance	RQ 3: defect types	RQ 4: QA process
A: Daimler	✓	✓	✓			✓
B: Wacker	✓	✓	✓			✓
C: TechDivision	✓	✓	✓	✓		✓
D: Univ. of Stuttgart	✓	✓	✓		✓	

them with the Chief Software Architect, who also has several years of experience in quality assurance.

Case C: TechDivision. For the third case, we analyzed the requirements of the agile software engineering company TechDivision GmbH. TechDivision has around 70 employees, working in 3 locations in Germany. They focus mainly on web development, i.e. creating product portals and e-commerce solutions for a variety of companies, as well as web consulting, especially focusing on search engine optimizations. Many of their products involve customization of Magento[6] or Typo3[7] frameworks.

In their projects, TechDivision follows an agile software development process using either Scrum (Schwaber and Sutherland, 2011) or Kanban (Anderson, 2010) methodologies. For their requirements, TechDivision applies user stories (Cohn, 2004), which they write and manage in Atlassian JIRA[8]. User stories at TechDivison follow the common Connextra format: *As a [Role], I want [Feature], so that [Reason]*. We will also follow this terminology here.

The systems under analysis consist of two online shopping portals, a customer-relationship system and a content-management system, all of which we cannot name for non-disclosure-agreement reasons. In total, we analyzed over 1000 user stories containing roughly 28,000 words. For RQ 2.2, we met with an experienced Scrum Master and a long-term developer, who have worked on several projects for TechDivision.

Case D: University of Stuttgart. The requirements of Case D were created by 52 groups of three 2nd-year students each during a compulsory practical course in the software engineering program at the University of Stuttgart. We removed one artifact, because it was incorrectly encoded, thus resulting in 51 requirements artifacts for this analysis.

The resulting requirements artifacts differ vastly in style; hence, we were unable to count them in terms of requirements, but instead only counted the structured use cases as provided by the authors, and quantified the artifacts by word size. The average size of a requirements artifact was 4471 words (min: 1425, max: 8807, see Fig. 6) and contained 19 use cases (min: 6, max: 39), thus creating a set of artifacts of nearly a quarter of a million words, including more than 950 use cases.

For practical reasons, we could not evaluate each research question in each case: For example, RQ 3 depends on the existence of reviews with documented results, which is often not existent in practice. Furthermore, depending the answers of RQ 4 on the potentially less experienced students from Case D would introduce a threat to the validity of our evaluation. Table 5 shows the mapping between research questions and study objects. The interviews for RQ 2.2 and RQ 4 lasted 60 min for each Case A and B and 120 min for Case C.

[6] http://www.magento.com .
[7] http://www.typo3.org .
[8] https://atlassian.com/software/jira .

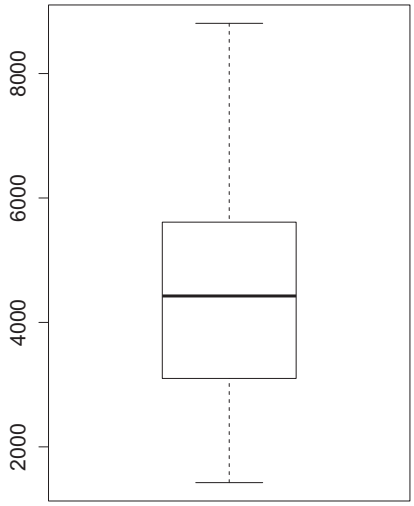


Fig. 6. Variation of size of requirements artifacts in Case D in words.

6.2.2. RQ 1: How many Requirements Smells are present in the artifacts?

Under this research question, we quantify the number of findings that appear in requirements. Table 6 shows the number of findings for each case, each requirements artifact and each smell and also puts these numbers in relation to the size of the artifact. We analyzed requirements of the size of more than 250k words, on which the smell detection produced in total more than 11k findings, thus revealing roughly 44 findings per thousand words.

Table 6 shows that all requirements artifacts contain findings of Requirements Smells. They vary from 5 findings for the smallest[9] case (A3) up to 572 for the largest case (C4). The number of findings strongly correlates with the size of the artifact (see Fig. 7, Spearman correlation of 0.9). Hence, in the remainder, we normalize the number of findings by the size of the artifact.

The artifacts of Daimler have an average of 26 findings per thousand words, in contrast to 41 for both Wacker and TechDivision and 43 for the artifacts produced by the students. Best to analyze the variance within a requirements artifact seems Case D, in which multiple teams had a similar background and project size. Fig. 8 shows the variance between the artifacts of Case D with an average of 44 findings, a minimum of 26 findings (D11) and a maximum of 75 findings (D32) per 1000 words.

When inspecting the different Requirements Smells, we can see that the most common smells are `vague pronouns` with 25 findings per 1000 words, followed by the `negative words` smell with 6 findings and the `loophole` smell with 4 findings. The least often smells are `non-verifiable terms` with 1 finding per 1000 words, and `ambiguous adverbs and adjectives` with 0.25 findings per 1000 words. In fact, the most common smell, `vague pronouns`, appears 100 times more often than the `ambiguous adverbs and adjectives`. To analyze

[9] In terms of total number of words.

Table 6
Quantitative summary of smell findings.

Case	Number of words	All Smells		Subjective Language Smell		Loophole Smell		Vague Pronouns Smell		Superlatives Smell		Negative Words Smell		Comparatives Smell		Non-verifiables Smell		Ambiguous A & A Smell	
		Abs	Rel	Abs	Rel	Abs	Rel	Abs	Rel	Abs	Rel	Abs	Rel	Abs	Rel	Abs	Rel	Abs	Rel
A1	1896	45	23.7	4	2.11	2	1.05	13	6.86	7	3.69	11	5	7	3.69	0	0	1	0.53
A2	2244	52	23.2	6	2.67	3	1.34	20	8.91	1	0.45	14	6.24	5	2.23	2	0.89	1	0.45
A3	199	5	25.1	0	0	0	0	3	15.08	0	0	2	10.05	0	0	0	0	0	0
A4	975	29	29.7	3	3.08	1	1.03	15	15.38	0	0	8	8.21	1	1.03	1	1.03	0	0
A5	524	20	38.2	0	0	0	0	14	26.72	0	0	5	9.54	0	0	1	1.91	0	0
A6	1100	32	29.1	0	0	0	0	8	7.27	0	0	13	11.82	7	6.36	4	3.64	0	0
Sum Daimler	6938	183	26.4	13	1.87	6	0.86	73	10.52	8	1.15	53	7.64	20	2.88	8	1.15	2	0.29
B1	2093	90	43	5	2.39	11	5.26	40	19.11	6	2.87	20	9.56	7	3.34	1	0.48	0	0
B2	1015	28	27.6	2	1.97	1	0.99	13	12.81	0	0	3	2.96	9	8.87	0	0	0	0
B3	458	31	67.7	0	0	19	41.48	9	19.65	1	2.18	0	0	1	2.18	0	0	1	2.18
Sum Wacker	3566	149	41.8	7	1.96	31	8.69	62	17.39	7	1.96	23	6.45	17	4.77	1	0.28	1	0.28
C1	5226	229	43.8	48	9.18	5	0.96	104	19	3	0.57	29	5.55	36	6.89	1	0.19	3	0.57
C2	2742	120	43.8	11	4.01	7	2.55	62	22.61	3	1.09	13	4.74	24	8.75	0	0	0	0
C3	6863	233	34	30	4.37	14	2.04	105	15	6	0.87	31	4.52	45	6.56	1	0.15	1	0.15
C4	13,124	572	43.6	35	2.67	16	1.22	339	25.83	11	0.84	101	7	49	3.73	9	0.69	12	0.914
Sum TechDivision	27,955	1154	41.3	124	4.44	42	1	610	21.82	23	0.82	174	6.22	154	5.51	11	0.39	16	0.57
Mean Stuttgart	4470	198.5	44.4	6.45	1.44	19.65	4	117.37	26.26	5.12	1.14	27.59	6.17	16.63	3.72	4.71	1.05	0.96	0.21
Sum Stuttgart	227,973	10,122	44.4	329	1.44	1002	4	5986	26.26	261	1.14	1407	6.17	848	3.72	240	1.05	49	0.21
Over all	266,432	11,608	43.6	473	1.78	1081	4.06	6731	25.26	299	1.12	1657	6.22	1039	3	260	0.98	68	0.26

the variance in depth, we again take the students' artifacts for reference. Fig. 9 shows the relative number of findings across the projects.

Interpretation. We interpret the quantitative overview along three variables: projects, contexts and the different Requirements Smells.

Projects When comparing at project level, we see that Cases A1–A6 (with outlier A5) and C1–C4 (with outlier C3) show quite similar numbers. In contrast B1 to B3 vary between 28 and 68 findings per 1000 words. When looking into the most extreme outliers B3 and D32, we see a systematic error that creates a large number of findings: Both projects repeatedly explain what the system *should*[10] do instead of what it *must* do. 16 of 19 `loopholes` findings in B3 and 29 of 37 `loophole` findings in D32 root from this problem. This can lead to difficult issues in contracting as requirements that are phrased with a *should* are commonly understood as optional (see e.g. RFC2119 (Bradner, 1997) for a detailed explanation).
Hence, we could see a surprising consistency in two of three industrial case studies. The Wacker data varies, so does the students case. In both cases, the negative extremes point at issues that potentially have expensive consequences.

Context The four cases differ strongly in their context: They write down requirements in different forms, vary in their software development methodology and also produce software for different domains. When comparing the findings at the domain level, we see that Daimler artifacts with an average of 26 findings per thousand words contain less findings than both Wacker and TechDivision with 41 findings and the artifacts produced by the students with 43 findings.
Our partners reported that there have been trainings for the authors of the cases A1–A6 recently, which could explain the difference. Another reason could be the strong focus that the automotive domain puts on requirements and requirements quality in contrast to the other domains. Lastly, also the strict process in this domain could be a reason for this striking difference of the Daimler requirements. Unsurprisingly, the students' requirements form the lower end of the scale, yet not by much.

Requirements Smells When comparing the eight smells, we see a strong variance between the number of findings, both in absolute as well as relative values. A qualitative inspection indicates reasons for the most occurring smells. First, the smell detection for `vague pronouns` finds all substituting pronouns in the requirements. Especially in German, in many sentences the reference of the pronoun can sometimes be derived from gender and grammatical case of the word, thus correctly detecting pronouns, but not *vague* pronouns. RQ 2.1 quantifies this issue. Second, the most common indication for `loophole` findings is the aforementioned use of the word *should*. We discuss this case in-depth with practitioners in RQ 2.2. Third, we will also inspect reasons for the high number of negative words findings in RQ 2.1 and RQ 2.2.

Answer to RQ 1. The number of findings in requirements artifacts strongly correlates with the size of the artifact. There are roughly 44 findings per 1000 words and some contexts show a striking similarity in the number of findings for their artifacts. In our cases, the automotive requirements had a lower number of findings whereas student artifacts contained a higher number of

[10] *Soll* is a German modal verb that is less strict than an English *must*.

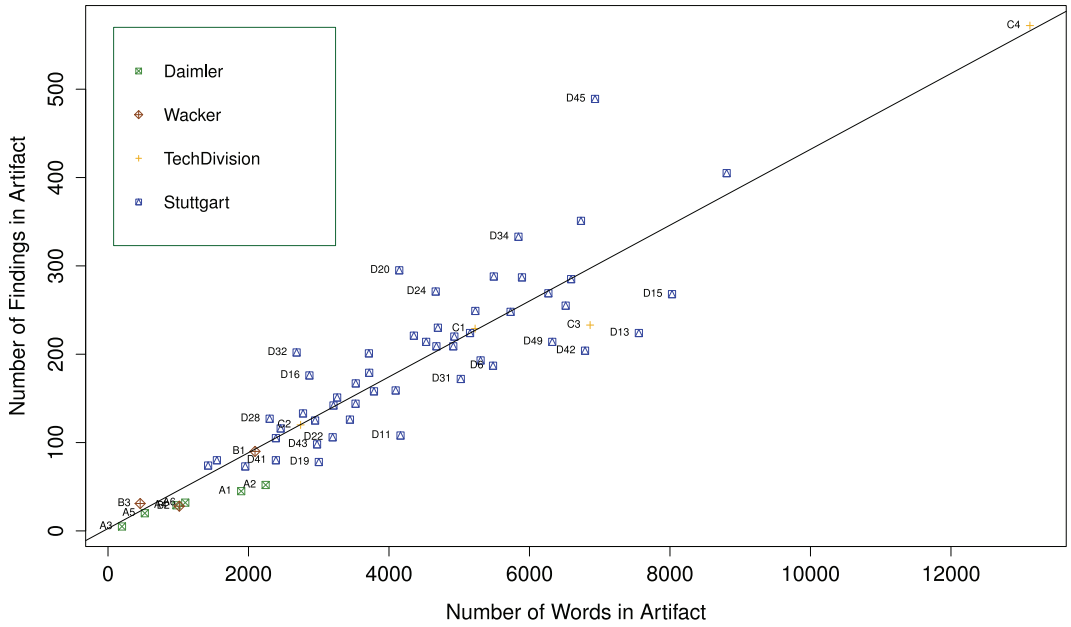


Fig. 7. Number of findings strongly correlates with size of artifact (for readability reasons, for the Stuttgart cases (blue) only IDs of less correlating artifacts are displayed). (For interpretation of the references to color in this figure legend, the reader is referred to either the symbols shown in the legend or to the web version of this article.)

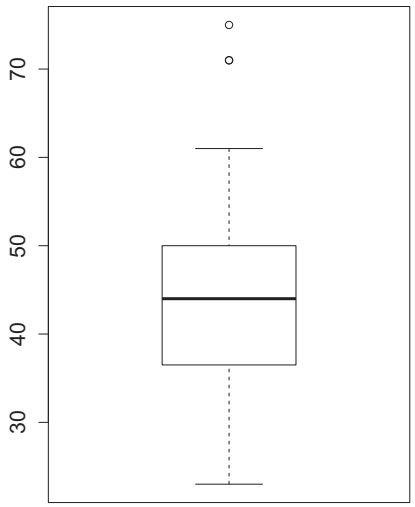


Fig. 8. Number of findings per 1000 words in Case D.

findings relative to the size of the artifacts. The most common findings are for the smells `loopholes` and `vague pronouns`.

6.2.3. RQ 2.1: How accurate is the smell detection?

To understand the capabilities of the smell detection, we need to understand precision as metric indicating how many of the detected findings are correct, as well as recall as a metric indicating how many of the correct findings are detected.

Precision. To understand to which extent the numbers of findings for certain smells in RQ 1 are caused by the detection mechanism, we inspected a random sample of 616 findings by taking equivalent sets of findings from each project and manually classifying whether the finding fulfills the smell definition. We could not inspect the same number of findings of each smell for each project, because some projects only had few or even no findings of a certain smell (see number of findings per project in Table 6).

Table 7
Precision of smell detection.

Smell	Findings inspected	Findings accepted	Findings rejected	Precision
Subjective Language Smell	69	66	3	0.96
Ambiguous Adverbs and Adjectives Smell	21	17	4	0.81
Loophole Smell	60	43	17	0.72
Non-verifiable Term Smell	23	16	7	0.70
Superlative Requirements Smell	39	19	20	0.49
Comparative Requirements Smell	88	42	46	0.48
Negative Words Smell	129	42	87	0.33
Vague Pronouns Smell	187	48	139	0.26
Average	77	36.6	40.4	0.59
Overall	616	293	323	0.48

Table 7 and Fig. 10 show the summary of this analysis: The precision of the detection of the `subjective language` smell revealed only three false positives in total, thus leading to a precision of 0.96. `Non-verifiable words`, `loophole`, and `ambiguous adverbs and adjectives` smells range between 0.70 and 0.81, hence leading to roughly one mistake in four suggestions. `Comparative` and `superlative` smells range around 0.5 which would mean that every second finding is correct. At the rear end of the list are the `negative words` and `vague pronouns` smells with one correct finding in three to four suggestions. Across all smells, the precision is between 0.48 (over all inspections) and 0.59, if we take the varying number of inspected findings between the smells into account. To understand these numbers, we qualitatively inspected the false positive classifications, revealing the following main reasons for false positives:

Grammatical errors in real world language. The first issue that creates false positives is the fact that our study analyzes real world language. Some of the requirements, especially in Case C, contained a number of grammatical flaws as well as dialectal phrases, which lead to wrong results in

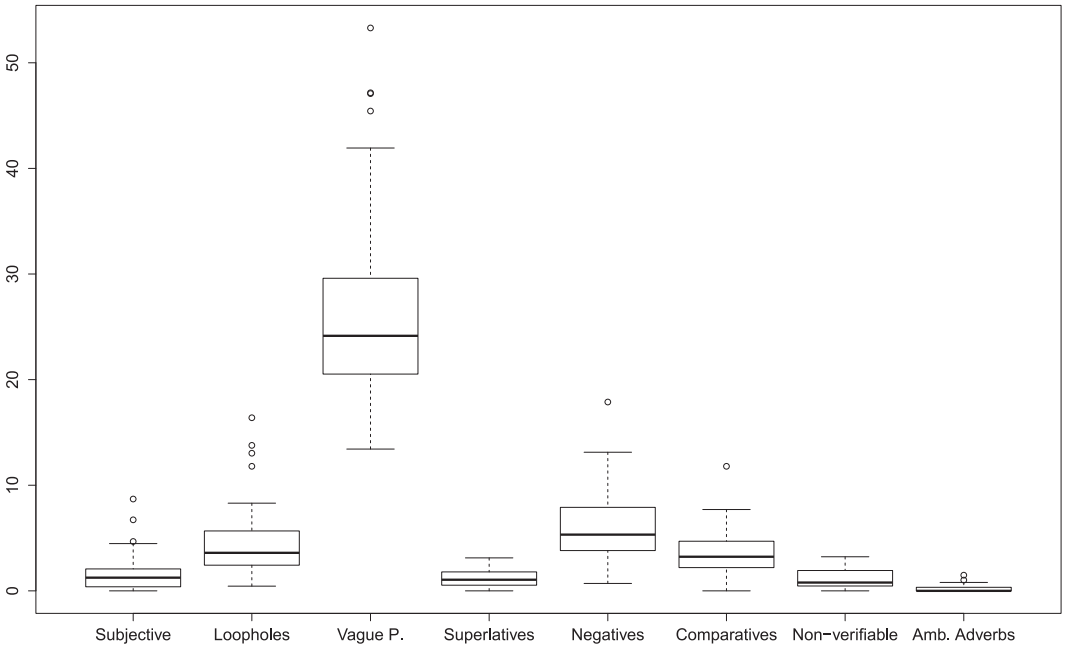


Fig. 9. Variation of smells per 1000 words in Case D.

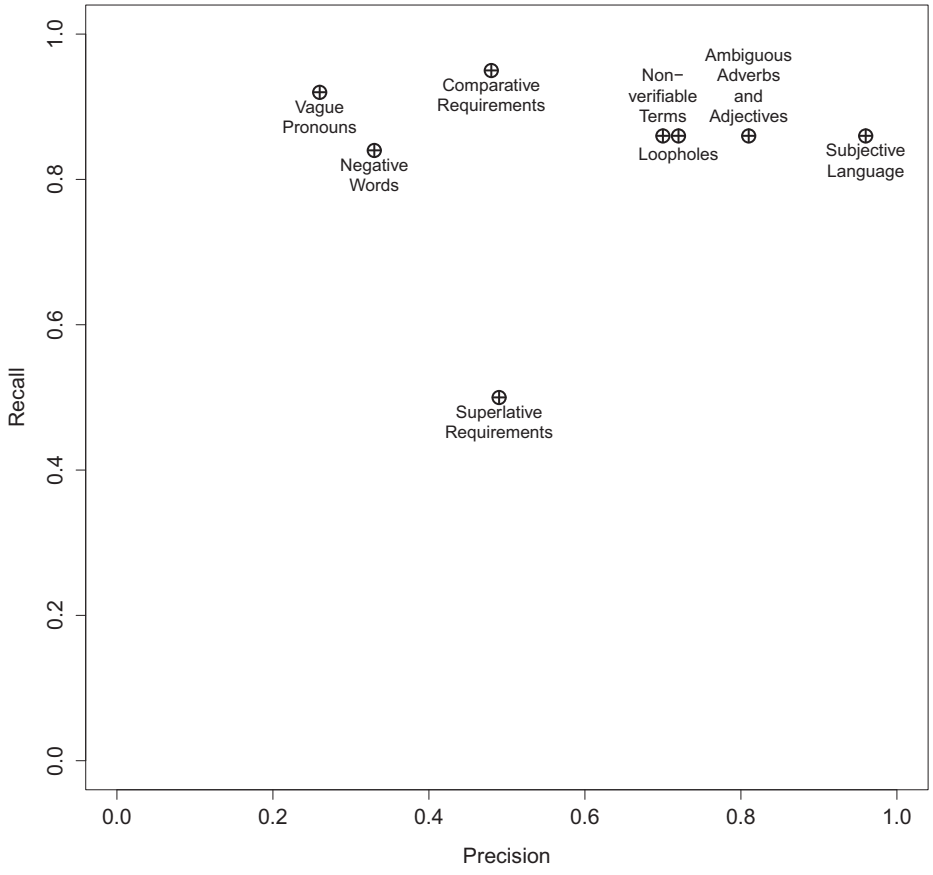


Fig. 10. Precision and recall of the discussed smell detection approaches.

the automatic morphologic analysis and automatic POS tagging and consequently also to false positives during smell detection.

Vague pronouns. The smell detection for `vague pronouns` showed the lowest precision. In the detection of this smell, we look for substituting pronouns, which are pronouns where the noun is not repeated after the pronoun[11], of which we characterize only every fourth finding as a defect. The reason behind this poor performance, besides a

[11] For example,*The father of these*. vs *The father of these kids.*

Table 8
Recall of smell detection within sample of 4 artifacts (16,271 words).

Smell	Findings in artifacts	Findings identified correctly	Recall
Ambiguity-related Smells	74	64	0.86
Superlative Requirements Smell	4	2	0.50
Comparative Requirements Smell	21	20	0.95
Negative Words Smell	64	54	0.84
Vague Pronouns Smell	37	34	0.92
Average	40	34.8	0.82
Overall	200	174	0.87

number of false positives due to the poor grammar mentioned before, is the comparably large number of grammatical exponents of the German language. In addition to number and three grammatical genders, the German language also has four grammatical cases. Therefore, in various instances of substituting pronouns, there is only one grammatical possibility of what the pronoun could refer to.

Findings in conditions. A third reason for false positives is that the smell detection, so far, takes very little context into account. For example, the `comparatives` smell aims at detecting requirements that define properties of the system relative to other systems or circumstances.[12] When searching for grammatical comparatives in requirements, roughly 48% of the cases are of the aforementioned kind. In roughly the same number of cases, however, the comparative describes a condition. For example, if the requirement states that *if the system takes more than 1 second to respond […]*, the comparison is not against another system or circumstance but against absolute numbers. Therefore, in this case, the comparative does not indicate a problem (one could even argue that this is an indicator for *good* quality).

A similar problem holds for the `negative phrases` smell: The smell detection aims at revealing statements of what the system should not do. Often, however, the negative is mentioned in conditions. For example, if the requirements express what to do *if the user input is not zero […]*, the negation relates to a condition and not to a property of the system.

Recall. When analyzing the accuracy of an automatic detection, we must look not only at precision, but also at recall, i.e. the ratio of all detected findings to all defects of a certain type in an artifact. To this end, we inspected one artifact of each case, in total a set of roughly 16,200 words, and manually identified the findings in each artifact. Due to the problems of distinguishing the various ambiguity-related smells, we analyzed the recall of these four smells as if it was one smell, without further differentiation (see Section 6.1.3).

The manual inspection revealed 200 findings in this artifact sample and an average recall of 0.82. Table 8 and Fig. 10 show the summary of the results: The comparison shows a recall between 0.84 and 0.95 for four of the five investigated smells. The highest recall was achieved by the `Comparative Requirements Smell`, with 0.95, which means that the smell detection missed one in 20 findings. The fifth smell, with the lowest recall, is `Superlative Requirements Smell` with a recall of 0.5. However, this smell is one of the rarest of the smells, as one can also see in the results to RQ 1. Therefore our analysis of the recall of this smell is based on few data points. Hence, we suggest to take the recall of this smell with care, and suggest that future studies should investigate this issue in more depth.

A further analysis of the false negatives shows that the smell detection missed findings because of imprecisions in the NLP libraries (i.e. Stanford NLP (Toutanova et al., 2003) for Lemmatization and POS Tagging and RFTagger (Schmid and Laws, 2008) for morphologic analysis). For the dictionary-based smells, the lemmatization did not correctly deduce the correct lemma, e.g. it did not understand that a certain word was a plural of a lemma. If only the lemmatized version of the word, i.e. the singular form, is in the dictionary, then the smell detector does not correctly identify the smell. In the false negative cases for the `Comparative` and `Superlative Requirements Smell`, RFTagger did not correctly classify the inflection.

Interpretation. The study revealed that the precision strongly varies between the different smells. Qualitative analysis provided further insights described next.

We can now explain the high number of findings for vague pronouns in RQ 1. If we assume that a quarter of the findings are correct, the number of findings in this category is closer to the remaining smells. Also, we could see that while there are certain reasons of impreciseness that root from the study objects themselves and are, thus, unavoidable, there is plenty of space for optimization. First, existing techniques from NLP could be applied to improve certain smells, such as the `vague pronouns`. Second, from the examples we have seen, we would argue that the application of heuristics could heavily improve the precision of existing smell detection techniques. For example, if we exploit the information available from POS tagging, we can find out whether a comparison refers to a number or numerical expression.

Regarding recall, our analysis shows only a slight variance between the smells, with the only outlier being the `Superlative Requirements Smell`; however, since this is a very rare smell, this recall is based on only few data points, therefore, we must consider this result with care. When inspecting the reasons for false negatives, we found that optimizations could be made through the lemmatizer. Future research in this direction should compare whether the accuracy of lemmatizers as reported in the field of computational linguistics also holds for requirements engineering artifacts. Furthermore, we analyzed requirements in German language where lemmatization is a more difficult problem than in English, since the language makes stronger use of inflections (e.g. with cases or gender). Hence, smell detectors based on lemmatization for the English language might work better than the results indicate in our analysis.

In general, the precision and recall are therefore comparable to other approaches with related purposes (see Section 2). However, is it sufficient for an application of Requirements Smells in practice?

First, when looking at precision, we must take into account that the current state of practice consists still of manual work and that the cost for running an automatic analysis is virtually zero. Nevertheless, checking a false positive finding takes effort which an inspector could rather spend in reading the document in more detail. However, as we see a high variation in the precision over different smells, we need to discuss these separately. Several of the smells have a precision of 0.7 and higher which is considered acceptable in static code analysis (Bessey et al., 2010). For other Requirements Smells, the precision is below 0.5. This means that every other finding will be a false positive. This can be critical in the effort spent in vain and annoy a user of the smell detection. Yet, we follow Menzies et al. (2007) that a low precision can be still useful "When there is little or no cost in checking false alarms." In

[12] As discussed in Section 3.2, the problem of comparatives in requirements is validation: How can we understand whether a system fulfills a requirements if that requirement is stated in a relative instead of an absolute way? What if the system in comparison changes its properties, would this render the requirement suddenly unfulfilled?

Table 9
Exemplary findings; shortened and translated by the authors, findings in bold.

ID	Finding	Relevant?	Aware?	Resolve?
1	As a visitor, I want to see the checkboxes in the different categories displayed **more clearly**, so that I can see more quickly that I can select and deselect categories.	Yes	Yes	Yes in short term
2	As a visitor, I want to see the checkboxes in the different categories displayed more clearly, so that I can see **more quickly** that I can select and deselect categories.	No	No	No
3	As an editor, I want to make it **simpler** to differentiate between …	No	No	No
4	As a visitor, I want to see **further** details, **e.g.** (…) , so that …	Yes	Yes	Yes immediately
5	As a customer, I want, if I have a **larger** number of E-Mails in my mailbox, …	Yes	Yes	Yes immediately
6	As an editor, I want to make it simpler to differentiate between A and B, therefore, A **should** be labeled as …	Yes	No	Yes in long term
7	As a provider, I want that, **as far as possible**, all fields, are mapped between System A and System B.	Yes	Yes	Yes immediately
8	As a provider I want the news section to be implemented with an effort **as low as possible**.	Yes	Yes	No
9	As a visitor, I do **not** want to see category X, so that I am **not** confronted with the issue.	No	No	No
10	As a visitor of the webpage, I want for **not** selected categories, the displayed hearts of the score (search results list) to be displayed in such a color, that the score display is **not** changed and always only hearts of relevant categories are displayed in color.	Yes	Yes	Yes immediately
11	As an employee, I want that an article, if **no** price is imported, despite the label 'available' to be **not** displayed in SYSTEM X, so that the article automatically resumes when a price is imported.	Yes	Yes	Yes immediately
12	As a user, I want to have the possibility to use custom values for **minimum** and **maximum**, so that (…).	No	No	No
13	As a visitor, I want to have a possibility to browse through previous and next products, so that I can **quickly** and **easily** look at multiple product without having to go back to the overview page.	No	No	No
14	As a visitor, I want to navigate to **meaningfully** structured categories via the menus.	Yes	Yes	Yes immediately
15	As a visitor, I want to **quickly** open the pictures of the website, so that unnecessary waiting is avoided.	Yes	Yes	Yes immediately
16	As a visitor of the website, I want a **nicely** designed search-suggest-box when I enter a text and wait.	Yes	Yes	Yes immediately
17	As a buyer, I want to select from a set of shopping providers (…), so that I can select the **best** suited shopping provider.	No	No	No
18	As an editor, I want to have multiple entry points for linking categories, so that the visitor can (…) get an overview of selected brands and categories and **their** filters.	Yes	No	No
19	As [OTHER SYSTEM], I want that an order of the status 'Order income' transitions into status 'wait for transmission into [SYSTEM]', so that I do not see the order when indexing open orders and so I do not process the order multiple times (and that **one** can see the status of the order in the backend properly).	No	No	No
20	As an editor, I want to know a good way how to transfer news content from [SYSTEM] to [SYSTEM] to be able to efficiently migrate **everything** at once.	Yes	Yes	No

our experience, the cost of checking a finding is often just a few seconds.

Second, when looking at recall, most of the smell detections reach a recall of more than 80%. Various publications, most prominently Kiyavitskaya et al. (2008) and Berry et al. (2012), argue that a recall close to 100% is a basic requirement for any tool for automatic QA in RE. The core argument is that with a lower recall, reviewers stop checking these aspects and consequently miss defects, and that reviewers need to check the complete artifact anyway. However, if taking the example of spell checkers and grammar checks, these are still used on a daily basis, although they are far away from 100% recall. Therefore, one could consequently also argue that the precision is more important than the recall.

In any case, whether the reported precision and recall are sufficient in industry needs further research in the future. As mentioned above, it mainly depends on two factors: the required investment versus the gained benefit (similar to the concept of technical debt). For the required investment, we argue that, based on our experience of analyzing the various cases presented here, one can quickly iterate through the detected findings with low investment. To further support this discussion, the following research question analyzes the aspect of the benefits to practitioners in more detail.

Answer to RQ 2.1. As shown in Tables 7 and 8, and as shown in Fig. 10, the precision is on average around 59%, with an average recall of 82%, but both vary between smells. We consider this reasonable for a task that is usually performed manually. However, this also depends on the relevance of findings to practitioners, which we analyze in RQ 2.2. The study also reveals improvements for future work through the application of deeper NLP.

6.2.4. RQ 2.2: Which of these smells are practically relevant in which context?

To understand whether the Requirements Smells help detecting relevant problems, we first performed a pre-study, in which we confronted practitioners of Daimler and Wacker with findings. The pre-study, which we reported in Femmer et al. (2014b), aimed at receiving qualitative and tacit feedback. It showed that Requirements Smells can in fact indicate relevant defects.

In contrast, in this study we analyze relevance in specific categories by interviewing practitioners at TechDivision on their opinion on the findings in terms of relevance, awareness, and whether these practitioners would resolve the suggested finding.

Quantitative observations. Table 9 reports the 20 findings that we discussed with TechDivision. In summary, we can see that they considered 65% of the findings as relevant for their context. Furthermore, they have not been aware of 45% of the findings. Lastly, they would act on 50% of the presented findings and on 40% even immediately.

Qualitative observations (true positives). The findings that the tool produces mostly constituted forms of underspecification. For example, in Finding #1 (see Table 9): *"As a searcher, I want to see the checkboxes in the different categories displayed**more clearly**, so that … "* (for similar examples, see Findings 3, 4, 14, 16, and 20). In this case, as in many of the other examples, the practitioners stated that no developer could implement this story properly. They also recalled various discussions in estimation meetings on what was to be done to complete these types of stories[13].

[13] Note that discussions can have different objectives, i.e. *what* is to be implemented and *how*. For these, *how* to implement a story is the team's task and thus

Table 10
Findings in different parts of user stories (T = total, Ro = role, F = feature, Re = reason).

Case	#Stories	w/o reason	Size in words				Findings absolute				Findings per 1000 words			
			Total	Role	Feature	Reason	T	Ro	F	Re	T	Ro	F	Re
C1	168	23	5226	801	2375	2050	229	1	83	145	44	1	35	71
C2	123	45	2742	260	1552	930	120	0	62	58	44	0	40	62
C3	230	19	6863	824	3090	2949	233	5	81	147	34	6	26	50
C4	561	203	13,124	1188	8223	3713	572	0	307	265	44	0	37	71
Sum	1082	290	27,955	3073	**15,240**	**9642**	1154	6	**533**	**615**	41	2	**35**	**64**

In the previous research questions, we have seen that Requirements Smells are able to detect `loopholes` in requirements, such as the usage of the word *should*. To understand the relevance of this finding in the context of an agile company, we also discussed the `loophole` in Finding #6. When we pointed out the finding, they responded that they considered expressing what the system *should* do in user stories problematic. They considered this defect a low risk, as the developers understood (*"If you are told that you should take out the trash, you understand that it is an imperative."*) and their user stories did never turn out to be of legal relevance. They concluded that they want to avoid this, but it has no immediate urgency in a project situation.

ISO 29148 discusses the use of `negative statements` (*"capabilities not to be provided"*). In a previous study (Femmer et al., 2014b) practitioners expressed their reluctance of this criterion. In contrast, in this study, practitioners said they would act upon 2 out of 3 of the negative statements (Findings #9–11) that we presented to them as they revealed unclear requirements. In one case they even remembered that this led to discussions about the implementation during the sprint. Table 9 shows many more, similar examples.

Qualitative observations (false positives). Also interesting are those cases that practitioners considered not relevant in their context or where practitioners said they would not act upon. Summarized, the reasons were the following:

- **Domain and context knowledge:** Some stories that were unclear to outsiders were understandable for someone knowing the system under consideration. For example, in user story #18 it was unclear to the first and second author what *their* refers to. It was clear, however, to both practitioners with knowledge about the system.
- **Process requirement:** In Finding #8, the smell reveals another conspicuous finding: The developer should put as *low effort as possible* into the implementation of this story. In the discussion, the reason for this was that the customer did not want to pay much for this implementation. Thus the story should only be fulfilled if it was possible to be fulfilled cheaply. While the practitioners told us they would not change anything about this story, they agreed that the smell pointed out something that violates common user story practice.
- **Finding in reason part:** In four cases, the practitioners agreed to the finding but considered it irrelevant as the finding was inside the *reason* part of the user story. This is due to this part of the user story only serving as additional information. This reason part is not used in testing nor is the information directly relevant for implementation. The main purpose is to understand the business value and to indicate the major goal to the team, similar to goals and goal modeling in traditional requirements engineering (Lamsweerde, 2009).

Answer to RQ 2.2. In summary, the practitioners expressed that 65% of the discussed findings were relevant, as they lead to lengthy discussions and unnecessary iterations in estimation. They also saw the problem of legal binding, but in contrast to the practitioners of Cases A and B, they considered these findings less relevant. Due to these results, they expressed their strong interest in exploring smell detection for projects; we will explain the results of this discussion in RQ 4.

Further observations of quality defects in different parts of a user story

We considered especially the last explanation for rejecting findings (finding in reason part of a user story) particularly interesting. We had noticed that the reason part was often written in a rather imprecise way. To be able to quantify this aspect, we automatically split user stories according to the language patterns and quantified the distribution of words as well as findings over the different parts of user stories.

Table 10 shows the results of this analysis. The number of words is roughly distributed as follows: 11% of the words of a user story describe the role, 55% of the words describe the feature and 34% describe the reason. Of the 1082 user stories, 290 had no reason part at all. Due to this uneven distribution, similar as in the previous analyses, we normalize the number of findings by the number of words in each part resulting in the *number of findings per 1000 words*.

Only 1% of the findings are located in the role part. In fact, when we inspected these findings, they were false positives due to the grammatical problems described in the previous section. The absence of findings in this section is expected, as this part of the user story only names the role and does not offer many chances for smells as described in Section 3.2. For the remainder, 46% of the findings are located in the feature and 53% are located in the reason part. In relation to its size, the difference is striking: With 64 findings per 1000 words, the reason has nearly double the number of findings of the feature part and nearly 70% more findings than the average requirement, as analyzed in Section 6.2.2.

In summary, the reason part of user stories is particularly prone to smells, but the qualitative analysis in RQ 2.2 reveals that practitioners consider findings in this section to be less relevant. This investigation could support further application of Requirements Smells in practice by helping to prioritize smells according to their location.

6.2.5. RQ 3: Which requirements quality defects can be detected with smells?

For 44 of the 51 requirements artifacts the students provided technical reviews. We qualitatively analyzed the results of 10 randomly selected reviews (around 20%). The inspected reviews were conducted by 5–7 reviewers (mean: 5.6), took 90 min and resulted in 18–69 defects (mean: 38.1). We iterated through the 381 defects documented in the reviews and evaluated whether the smell

discussions can help finding the best way. In contrast, *what* the product owner wants is outside of the team's scope and therefore should not be a matter of discussion.

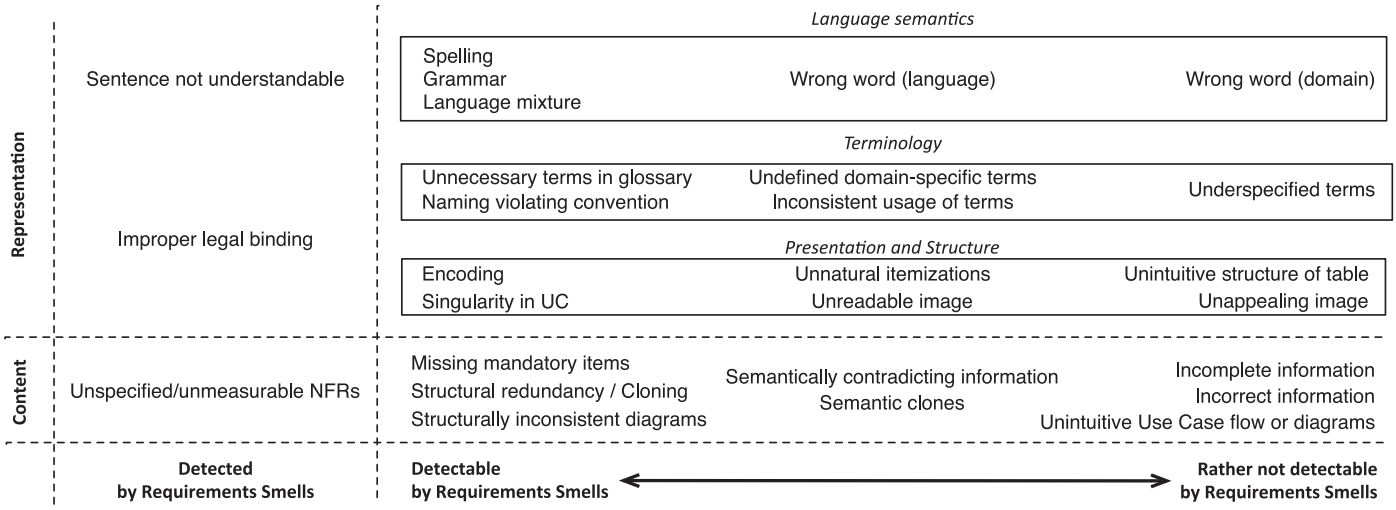


Fig. 11. Findings in requirements reviews, classified by content/representation and detection.

detection produced findings indicating these defects. If no smell indicated the defect, we openly classified the defects. We did not quantify these results, because the resulting numbers would assume and suggest that the distribution of defects is representative for regular projects, which we are unsure about (i.e. because of a high number of spelling and grammatical issues).

The classification of the defects and their comparison with the detected smells resulted in the following list of defects indicated by Requirements Smells:

- **Sentence not understandable.** In some instances, when the defect suggested changing the sentence to improve understandability, these sentences were highlighted especially by the `vague pronouns` and `negative statements` smells.
- **Improper legal binding.** Various requirements artifacts had issues with improper legal binding. In one case, the reviewers recognized this and demanded the use of the term *must*. The `loopholes` smell pinpointed at this issue.
- **Unspecified/unmeasurable NFRs.** Various smells, especially the `superlatives smell`, indicated at defects of underspecification within non-functional requirements.

The remaining defects were not indicated by Requirements Smells.

Interpretation. The quantitative distribution of defects is not necessarily representative for industry projects and, thus, has not been not analyzed. The reviews clearly show that manual inspection discovered the same defects as in the previous research question: Understandability, legally binding terminology and underspecified requirements. These are issues with regards to representation but also the content described in the artifact. We argue that these issues are common for requirements artifacts. Requirements Smells can therefore indicate relevant defects from multiple, independent sources (manual inspection, interviews with practitioners, independent manual reviews) for multiple, independent cases.

Answer to RQ 3. Automatic smell detection can point to issues in both representation (e.g. improper legal binding) and content (underspecified/unmeasurable NFRs). The analysis of the reported defects indicates that more defects could be automatically detected (see section *further discussion on detectability of defects* described next). Nevertheless, just as for static code analysis, we see that automatic analysis cannot indicate all defects and thus must be accompanied by reviews (Wagner et al., 2005). The fourth research question aims at analyzing this aspect in depth.

Further discussion on detectability of defects. During the analysis, if no smells indicated the defect, we openly classified the defects. While discussing the resulting list of defects and the degree to which they are detectable within the group of authors, we came up with a classification which is broader as initially planned while designing the study. This classification considers whether a defect:

- *Already can* be detected
- *Could* be detected, but is not implemented yet in our detection
- *Cannot be detected at the moment*, but should be soon
- *Cannot be detected at all* and probably won't be soon

This classification is purely based on our knowledge of existing related work and our subjective expectations gained during the data analysis process. The classification yielded in a map visualized in Fig. 11. The figure is structured in two dimensions: On the vertical axis, we group the defects into *defects relating to the content*, and *defects relating to representation*. Furthermore, on the horizontal axis, we map the items according to the expected precision and completeness we believe the detection could be (i.e. the classification above). The further left an item, the more precise and complete we expect a smell detection to be; the items on the right we assume to be close to impossible to detect in a general case.

With the defects that our current approach does not reveal, this research question shows that more defects could be detected: These are namely defects with terminology, singularity in use cases and structural issues focusing on the content such as the absence of mandatory elements in the artifact (Kamata and Tamai, 2007), structural redundancy (Juergens et al., 2010) or structural inconsistency between content. It remains unclear how far more enhanced language analysis with more sophisticated NLP and ontologies can enable to understand language. In any case, when a defect remains subtle and vague in its definition, such as an unintuitive structuring or design, we only see potential for automation if a defect can be defined precisely. For problems relating to the domain itself (e.g. incomplete information about the domain or incorrect information with regards to the domain), we consider it impossible to detect issues unless formalizing the concepts of the domain.

6.2.6. RQ 4: How could smells help in the QA process?

After the interviews and analysis, we asked all involved practitioners whether or not they think requirements smell detection is

a helpful support, and whether and how they would integrate it in their context. We asked those questions openly and transcribed the answers for validation by the interviewees and later coding. In the following, we report on the results structured by topics. Where applicable, we provide the verbatim answers in relation to their cases (A, B or C).

Overall Evaluation. In general, all practitioners agreed on the usefulness of the smell detection even if considering different perspectives that arise from their process setting. One practitioner (Case C) reports that he expects one benefit in using smell detection is that it would lead to a reduction of the time spent for effort estimations (in context of agile methods), as the product owner could benefit from the smell detection on the fly and, thus, avoid misinterpretations later.

Quotes on Overall Evaluation

A. *"I think that smells can help to analyze a specification."*
B. *"The method of Requirements Smells is a valuable extension in the area of requirements engineering and gives helpful input concerning the quality of specified requirements in early development phases."*
C. *"I think such a smell detection is of high value to make sure that our team is confronted with already quality assured [user] stories. This can reduce the time in our effort estimations, because the product owner would directly notice on the fly what could lead to misinterpretations later."*

Integration into process. When asked for how the practitioners would integrate the smell detection into their process setting, we got varying answers depending on the process. The practitioner relying more on rich process models (Case B) could imagine using a smell detection either as a support for the person writing the requirements or as part of a more fundamental QA method for the company. But also the practitioner relying more on the agile methods (Case C) could imagine using Requirements Smells as a support for the person writing the requirements or in context of analytical QA. In addition, one potential use is seen in context of problem management. Importantly, all practitioners see the full potential of a smell detection only if integrated in their existing tool chain (see also quotes on constraints and limitations).

Quotes on Integration into Process

B. *"I like to compare Requirements Smells to the "check spelling aid" known e.g. from Microsoft Word. So for me Requirements Smells are intuitive and lightweight and should be used and integrated within requirements engineering and quality assurance processes."*
C. *"As a product owner, I would use a smell detection on the fly […]. In addition, smell detection could help in analytical QA, as it could reveal when a problem occurs repeatedly, either in a project or in the company as a whole."*

Constraints and Limitations. One facet we consider especially interesting when using qualitative data is the chance to reveal further fields of improvement. We therefore concentrate now on the constraints that would hamper the usage of a smell detection. One facet we believe to be important is that practitioners want to avoid additional effort when using smell detection in their context. Furthermore, the practitioner of Case A believes that the automatic smell detection requires a common understanding on the notion of RE quality. He further indicates that the smell detection should explicitly take into account that some criteria cannot be met at every stage of a project.

Quotes on Constraints and Limitations

A. *"First, the people who need to write the specification received training which gives the required performance criteria. Second, abstraction levels must be taken into account during the smell detection process, since at higher abstraction levels different criteria cannot be met (e.g. vague pronouns or subjective language)."*
B. *"As a product owner, I would use a smell detection on the fly provided that it would not mean additional effort [such as by having to use another tool]."*

Answer to RQ 4. Our practitioners provided a general agreement on potential benefits of using smell detection a quality assurance context. When asked how they would integrate the requirements smell detection, they see possibility for both analytical and constructive QA, provided, however, this integration would not increase the required effort, e.g. by integrating the detection into existing tool chains.

6.2.7. Evaluation of validity

We use the structure of threats to validity from (Runeson et al., 2012) to discuss the evaluation of the validity of our study.

Construct validity. In our evaluation, we analyzed Requirements Smells in the terms of false positives, relevance and relation to quality defects. There are threats that the understanding of these terms varies and, thus, the results are not repeatable. Yet, we are confident that our validity procedures described in Sect. 6.1.5 reduced this threat. For the false positives, we classified a subset of the findings independently, and afterwards compared (inter-rater agreement Cohen's kappa: 0.53) and discussed the results. We subsequently reclassified a different subset of findings again, which lead to an inter-rater agreement (Cohen's kappa) of 0.72. For the classification of false negatives, we reclassified one document separately, calculating the percentage of agreement on false positives[14]. This lead to an agreement of 88%.

We consider both of these substantial agreements, especially in the inherently ambiguous and complex domain of RE. Thus, we consider this threat as sufficiently controlled.

Internal validity. A threat to the internal validity of our results is that the experience of the students as well as the practitioners might play a role in their ratings of relevance or detection of quality defects. We mitigated this threat by choosing only practitioners for the ratings and interviews who had several years of experience. The students are only in the second year. We cannot mitigate this threat but consider the effect to be small. There might be some defects not found by the students that could have been indicated by a smell as well as unfound defects undetectable by smells. Hence, future studies will add to the classification but are unlikely to change it substantially. Personal pride could potentially have an impact on the answers to a RQ 2.2, if practitioners are not able to professionally discuss their own work products. In our cases, however,

[14] We did not employ Cohen's kappa here, since the number of true positives (non-smell words) would strongly dominate the result and therefore skew the inter-rater agreement. Instead, we calculated the ratio of findings which both rating teams independently classified as false positive to the number of findings which only one of the teams classified false positive.

all practitioners openly accepted the discussions (as can be seen in their answers). Even though we carefully supervised this threat, we have not found signs of personal bias in the cases involved. Finally, the students might also have been influenced by the review guidelines we provided. Yet, none of the investigated smells was explicitly listed in the guidelines. Instead, the guideline contained rather high-level aspects such as "unambiguity". Although we consider this threat to be a minor one, it is still present.

External validity. As requirements engineering is a diverse field, the main threat to the external validity of our results is that we do not cover all domains and ways of specifying requirements. We mitigated this threat to some degree by covering at least several different domains and study objects, of which some are purely textual requirements artifacts, some use cases, and some user stories. We argue that this represents a large share of today's requirements practices.

Reliability. Our study contains several classifications and ratings performed by people. This constitutes a threat to the reliability of our results. We are confident, however, that the peer debriefing and member checking procedures helped to reduce this threat.

7. Conclusion

In this paper, we defined Requirements Smells and presented an approach to the detection of Requirements Smells which we empirically evaluated in a multi-case study. In the following, we summarize our conclusions, relate it to existing evidence on the detection of natural language quality defects in requirements artifacts, and we discuss the impact and limitations of our approach and its evaluation. We close with outlining future work.

7.1. Summary of conclusions

First, we proposed a light-weight approach to detect Requirements Smells. It is based on the natural language criteria of ISO 29148 and serves to rapidly detect Requirements Smells. We define the term *Requirement Smell* as an indicator of a quality violation, which may lead to a defect, with a concrete location and a detection mechanism, and we also give definitions of a concrete set of smells.

Second, we developed an implementation that is able to detect Requirements Smells by using part-of-speech (POS) tagging, morphological analysis and dictionaries. We found that it is possible to provide such tool support and outlined how such a tool could be integrated into quality assurance.

Third, in the empirical evaluation, our approach showed to support us in automatically analyzing requirements of the size of 250k words. Findings were present throughout all cases but in varying frequencies between 22 and 67 findings per 1000 words. Outliers indicated serious issues. An investigation of the detection precision showed an average precision around 0.59 over all smells, again varying between 0.26 and 0.96. The recall was on average 0.82, but also varied between 0.5 and 0.95. To improve the accuracy, we described concrete improvement potential based on real world, practical examples.

A further analysis of reviews and practitioner's opinions strengthen our confidence that smells indicate quality defects in requirements. For these quality defects, practitioners explicitly stated the negative impact of discovered findings on estimation and implementation in projects. The study also showed, however, that while Requirements Smell detection can help during QA presumedly in a broad spectrum of methodologies followed (including agile ones), the relevance of Requirements Smells varies between cases. Hence, it is necessary to tailor the detection to the context of a project or company. We analyzed this factor in depth, demonstrating that the reason part of a user story contains most findings (absolutely and relatively), but practitioners consider these findings less relevant as they argue that this part is not commonly used in implementation or testing. This raises the question of the relevance of this part at all, at least from a quality assurance perspective, which should be investigated in future work.

Our comparison with defects found in reviews furthermore showed that the Requirements Smell detection partly overlaps with results from reviews. As a result, we provide a map of defects in requirements artifacts in which we give a first indication where Requirements Smells can provide support and where they cannot.

Therefore, we provide empirical evidence from multiple, independent sources (manual inspection, interviews with practitioners, independent manual reviews) for multiple, independent cases, showing that Requirements Smells can indicate relevant defects across different forms of requirements, different domains, and different methodologies followed.

7.2. Relation to existing evidence

Existing approaches in the direction of automatic QA for RE are based on various quality models, including the ambiguity handbook by Berry et al. (2003), the now superseeded IEEE 830 standard (IEEE Computer Society, 1998) and proprietary models. Yet, according to a recent literature review by Schneider and Berenbach (2013), ISO 29148 is the current standard in RE *"that every requirements engineer should be familiar with"*. However, no detailed empirical studies (see Table 1) exist for the quality violations described in ISO 29148. When comparing to similar, related quality violations, also few empirical, industrial case studies exist (see Table 2). Gleich et al. (2010) and Chantree et al. (2006) report for conceptually similar problems, a precision of the detection between 34% and 75% (97% in a special case), and a recall between 2% and 86%. Krisch and Houdek (2015) report a lower precision in an industrial setting. The precision and recall for the detection of the smells, which we developed based on the description in the standard, are in a similar range to the aforementioned. In summary, this work provides a detailed empirical evaluation on the quality factors of ISO 29148, including a deeper understanding of both existing and novel factors.

We also take a first step from the opposite perspective: So far, to all our knowledge, all related work starts from a certain quality model and goes into automation. Our results to RQ 3 provides a bigger picture for understanding in how far quality defects in requirements could be addressed through automatic analysis in general.

Our results to RQ 2.2 furthermore provides evidence for the claim by Gervasi and Nuseibeh (2002) that *"Lightweight validation can discover subtle errors in requirements."* More precisely, our work indicates that automatic analysis can find a set of relevant defects in requirements artifacts by providing evidence from multiple case studies in various domains and approaches. The responses by practitioners to the findings do, to some extent, contradict the claim by Kiyavitskaya et al. (2008) who state that "*any tool […] should have 100% recall*". Practitioners responded very positively on our first prototype and the smells it finds. Yet, obviously, more detailed and broader evaluations, especially conducted independently by other researchers not involved in the development of Smella, should follow.

7.3. Impact/Implications

For practitioners, Requirements Smells provide a way to find certain issues in a requirements artifact without expensive review

cycles. We see three main benefits of this approach: First, the approach, just as static analysis for code, can enable project leads to keep a basic hygiene for their requirements artifacts. Second, the review team can avoid discussing obvious issues and focus on the important, difficult, domain-specific aspects in the review itself. Third, the requirements engineers receive a tool for immediate feedback, which can help them to increase their awareness for certain quality aspects and establish common guidelines for requirements artifacts.

Yet, the low precision for some of the smells might cause unnecessary work checking and rejecting findings from the automatic smell detection. Hence, at least for now, it is advisable to concentrate on the highly accurate smells.

For researchers, this work sharpens the term Requirements Smell by providing a definition and a taxonomy. By implementing and rating concrete smell findings, we also came to the conclusion, however, that not all of the requirements defects from ISO/IEC/IEEE 29148 can be clearly distinguished as Requirements Smells. In particular, the difference between *Subjective Language, Ambiguous Adverbs and Adjectives, Non-verifiable Terms*, and *Loopholes* was not always clear to us during our investigations (see RQ 2.1). Therefore, we, as a community, can take our smell taxonomy as a starting point, but we also need to critically reflect on some smells to further refine the taxonomy.

Finally, empirical evidence in RE is, in general, difficult to obtain because many concepts depend on subjectivity (Méndez Fernández et al., 2014). One issue increasing the level of difficulty in evidence-based research in RE remains that most requirements specifications are written in natural language. Therefore, they do not lend themselves for automated analyses. Requirements Smell detection provides us with a means to quantify the extent of certain defects in a large sample of requirements artifacts while explicitly taking into account the sensitivity of findings to their context. Hence, this allows us to consider a whole new spectrum of questions worth studying in an empirical manner.

7.4. Limitations

We concentrated on a first set of concrete Requirements Smells based on our interpretation of the sometimes imprecise language criteria of ISO/IEC/IEEE 29148. There are more smells, also with different characteristics than the ones we proposed and analyzed. In addition, even though we diversified our study objects over domains, methods and different types of requirements, we cannot generalize our findings to all applicable contexts. We therefore consider the presented results only a first step towards the continuous application of Requirements Smells in software engineering projects.

7.5. Future work

Our work focuses on Requirements Smells based on ISO/IEC/IEEE 29148. Future work needs to clarify and extend this taxonomy based on related work and experience in practice. This also includes the development of other Requirements Smell detection techniques to increase our understanding about which defects can be revealed by Requirements Smells and which defects cannot.

Second, this first study gained first insights into the usefulness of Requirements Smells for QA. We furthermore sketched an integration of Requirements Smells into a QA process. Yet, a full integration and the consequences must be analyzed in depth. In particular, we need to understand whether smell detection as a supporting tool, similar to spell checking, as pointed out by one of our participants, enables requirements engineers to improve their requirements artifacts.

Lastly, Requirements Smells focus on the detection of issues in requirements artifacts. They require a thorough understanding of the impact of a quality defect, which is hence also part of the requirements smell taxonomy. This link must be carefully evaluated and analyzed in practice. Our preliminary works on this topic (Femmer et al., 2014a; Mund et al., 2015) provide first ideas in that direction.

Acknowledgments

We would like to thank Elmar Juergens, Michael Klose, Ilona Zimmer, Joerg Zimmer, Heike Frank, Jonas Eckhardt as well as the software engineering students of Stuttgart University for their support during the case studies and feedback on earlier drafts of this paper.

This work was performed within the project Q-Effekt; it was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS15003 A-B. The authors assume responsibility for the content.

Appendix A. Requirements checklist

Table A.11
Checklist for the students' requirements reviews. Created by Anke Drappa, Patricia Mandl-Striegnitz and Holger Röder based on (Cockburn, 2000) and (Ludewig and Lichter, 2010). Translated from German.

The document is well structured and easy to understand.
All used terms are clearly defined and consistently used.
All external interfaces are clearly defined.
The level of detail is consistent throughout the document.
The requirements are consistent and unambiguous.
The defined requirements are consistent with the state of the art.
All tasks and data have useful identifiers.
Data is not defined redundantly.
The defined relationships between data objects are necessary and sufficient.
The specification of quality attributes is realistic, useful, quantifiable and unambiguous.
The user interface is comfortable and easy to learn.
The use case describes a behavior of the system which is valuable and visible for the actor.
The use case is described in a table which is consistently used for the whole requirements specification.
The use case has a unique ID.
The use case has a unique and expressive name.
The main actor's goal is described in an understandable way.
All actors participating in the use case are specified.
If there is more than one actor, the main actor is identified.
The preconditions of the use case are specified.
The postconditions for the use case are specified.
It is clearly specified how the main actor triggers the main success scenario.
The main success scenario has 3 to 9 steps.
After the main success scenario, the postconditions hold.
The main actor reaches their goal by the main success scenario.
Each step is sequentially numbered.
It is clear which actor is executing the step.
The step does not describe details of the user interface.
The step describes exactly one action of the acting actor.
There are postconditions for each extension.
It is clearly specified in which step the main success scenario deviates into an extension.
The conditions for the deviation into an extension are clearly specified.
After an extension, all postconditions for that extension hold.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.jss.2016.02.047

References

Ambriola, V., Gervasi, V., 2006. On the systematic analysis of natural language requirements with CIRCE. Autom. Software Eng. 13 (1), 107–167. doi:10.1007/s10515-006-5468-2.

Anda, B., Sjøberg, D.I.K., 2002. Towards an inspection technique for use case models. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering. ACM doi:10.1145/568760.568785.

Anderson, D.J., 2010. Kanban. Blue Hole Press.

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2015. Automated checking of conformance to requirements templates using natural language processing. IEEE Trans. Software Eng. 41 (10), 944–968. doi:10.1109/TSE.2015.2428709.

Berry, D., Gacitua, R., Sawyer, P., Tjong, S.F., 2012. The case for dumb requirements engineering tools. In: Requirements Engineering: Foundation for Software Quality. Springer, Berlin, Heidelberg, pp. 211–217. doi:10.1007/978-3-642-28714-5_18.

Berry, D.M., Bucchiarone, A., Gnesi, S., Lami, G., Trentanni, G., 2006. A new quality model for natural language requirements specifications. In: Requirements Engineering: Foundation for Software Quality. Essener Informatik Beiträge.

Berry, D.M., Kamsties, E., Krieger, M.M., 2003. From contract drafting to software specification: linguistic sources of ambiguity. Technical Report. School of Computer Science, University of Waterloo, Waterloo, ON, Canada.

Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D., 2010. A few billion lines of code later: using static analysis to find bugs in the real world. Commun. ACM 53 (2), 66–75. doi:10.1145/1646353.1646374.

Bradner, S., 1997. Key words for use in RFCs to Indicate Requirement Levels - RFC 2119. https://www.ietf.org/rfc/rfc2119.txt.

Bucchiarone, A., Gnesi, S., Pierini, P., 2005. Quality analysis of NL requirements: an industrial case study. In: 13th IEEE International Requirements Engineering Conference, pp. 390–394.

Chantree, F., Nuseibeh, B., Roeck, A.D., Willis, A., 2006. Identifying nocuous ambiguities in natural language requirements. In: 14th IEEE International Requirements Engineering Conference, pp. 59–68. doi:10.1109/RE.2006.31.

Ciemniewska, A., Jurkiewicz, J., Olek, L., Nawrocki, J., 2007. Supporting use-case reviews. In: Business Information Systems. Springer, Berlin, Heidelberg, pp. 424–437. doi:10.1007/978-3-540-72035-5_33.

Cockburn, A., 2000. Writing Effective Use Cases. Addison-Wesley.

Cohn, M., 2004. User Stories Applied: For Agile Software Development. Addison-Wesley Professional.

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., Theofanos, M., 1993. Identifying and measuring quality in a software requirements specification. In: Proceedings First International Software Metrics Symposium, pp. 141–152. doi:10.1109/METRIC.1993.263792.

De Bruijn, F., Dekkers, H.L., 2010. Ambiguity in natural language software requirements: a case study. In: Requirements Engineering: Foundation for Software Quality. Springer, Berlin, Heidelberg, pp. 233–247. doi:10.1007/978-3-642-14192-8_21.

Denger, C., Berry, D., Kamsties, E., 2003. Higher quality requirements specifications through natural language patterns. In: Software: Science, Technology and Engineering. IEEE, pp. 80–90. doi:10.1109/SWSTE.2003.1245428.

van Deursen, A., Moonen, L., van den Bergh, A., Kok, G., 2001. Refactoring test code. CWI.

Fabbrini, F., Fusani, M., Gnesi, S., Lami, G., 2001a. An automatic quality evaluation for natural language requirements. In: Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, vol. 1, pp. 4–5.

Fabbrini, F., Fusani, M., Gnesi, S., Lami, G., 2001b. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In: Proceedings 26th Annual NASA Goddard Software Engineering Workshop. IEEE Computer Society, pp. 97–105. doi:10.1109/SEW.2001.992662.

Fagan, M., 2002. Design and code inspections to reduce errors in program development. In: Software Pioneers. Springer, pp. 575–607.

Falessi, D., Cantone, G., Canfora, G., 2013. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. IEEE Trans. Software Eng. 39 (1), 18–44.

Fantechi, A., Gnesi, S., Lami, G., Maccari, A., 2003. Application of linguistic techniques for use case analysis. Requirements Eng. 8 (3), 161–170. doi:10.1109/ICRE.2002.1048518.

Femmer, H., Kučera, J., Vetrò, A., 2014a. On the impact of passive voice requirements on domain modelling. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, New York, NY, USA, pp. 21:1–21:4. doi:10.1145/2652524.2652554.

Femmer, H., Méndez Fernández, D., Juergens, E., Klose, M., Zimmer, I., Zimmer, J., 2014b. Rapid requirements checks with requirements smells: two case studies. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. ACM, New York, NY, USA, pp. 10–19. doi:10.1145/2593812.2593817.

Femmer, H., Méndez Fernández, D., Wagner, S., Eder, S., 2015. Supplementary online-material: analysis of related work. Created on: 2015-12-22.

Femmer, H., Mund, J., Mendez Fernandez, D., 2015. It's the activities, stupid! A new perspective on RE quality. In: Proceedings of the 2nd International Workshop on Requirements Engineering and Testing, pp. 13–19. doi:10.1109/RET.2015.11.

Fowler, M., Beck, K., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

Génova, G., Fuentes, J.M., Llorens, J., Hurtado, O., Moreno, V., 2011. A framework to measure and improve the quality of textual requirements. Requirements Eng. 18 (1), 25–41. doi:10.1007/s00766-011-0134-z.

Gervasi, V., Nuseibeh, B., 2002. Lightweight validation of natural language requirements. Software: Pract. Exper. 32 (2), 113–133. doi:10.1002/spe.430.

Gleich, B., Creighton, O., Kof, L., 2010. Ambiguity detection: towards a tool explaining ambiguity sources. In: Requirements Engineering: Foundation for Software Quality. Springer, Berlin, Heidelberg, pp. 218–232. doi:10.1007/978-3-642-14192-8_20.

Hauptmann, B., Junker, M., Eder, S., Heinemann, L., Vaas, R., Braun, P., 2013. Hunting for smells in natural language tests. In: Proceedings of the International Conference on Software Engineering, pp. 1217–1220. doi:10.1109/ICSE.2013.6606682.

IEEE Computer Society, 1998. IEEE Recommended Practice for Software Requirements Specifications. https://standards.ieee.org/findstds/standard/830-1998.html.

ISO, IEC, IEEE, 2011. ISO/IEC/IEEE 29148:2011. https://standards.ieee.org/findstds/standard/29148-2011.html.

Juergens, E., Deissenboeck, F., Feilkas, M., Hummel, B., Schaetz, B., Wagner, S., Domann, C., Streit, J., 2010. Can clone detection support quality assessments of requirements specifications? In: Proceedings of the International Conference on Software Engineering, pp. 79–88. doi:10.1145/1810295.1810308.

Juergens, E., Deissenboeck, F., Hummel, B., Wagner, S., 2009. Do code clones matter? In: Proceedings of the International Conference on Software Engineering, pp. 485–495.

Jurafsky, D., Martin, J.H., 2014. Speech and Language Processing, 2nd ed. Pearson Education.

Kamata, M.I., Tamai, T., 2007. How does requirements quality relate to project success or failure? In: 15th IEEE International Requirements Engineering Conference, pp. 69–78. doi:10.1109/RE.2007.31.

Kamsties, E., Berry, D.M., Paech, B., 2001. Detecting ambiguities in requirements documents using inspections. In: Proceedings of the 1st Workshop on Inspection in Software Engineering, pp. 68–80.

Kamsties, E., Peach, B., 2000. Taming ambiguity in natural language requirements. In: Proceedings of the International Conference on System and Software Engineering and their Applications, pp. 1–8.

Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M., 2008. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Eng. 13 (3), 207–239. doi:10.1007/s00766-008-0063-7.

Knauss, E., Flohr, T., 2007. Managing requirement engineering processes by adapted quality gateways and critique-based RE-tools. In: Proceedings of Workshop on Measuring Requirements for Project and Product Success.

Knauss, E., Lübke, D., Meyer, S., 2009. Feedback-driven requirements engineering: the heuristic requirements assistant. In: Proceedings of the International Conference in Software Engineering, pp. 587–590.

Knight, J.C., Myers, E.A., 1993. An improved inspection technique. Commun. ACM 36 (11), 51–61. doi:10.1145/163359.163366.

Kof, L., 2007a. Scenarios: identifying missing objects and actions by means of computational linguistics. In: Proceedings of the 15th IEEE International Requirements Engineering Conference, pp. 121–130. doi:10.1109/RE.2007.38.

Kof, L., 2007b. Treatment of passive voice and conjunctions in use case documents. In: Natural Language Processing and Information Systems, vol. 4592, pp. 181–192. doi:10.1007/978-3-540-73351-5_16.

Körner, S.J., Brumm, T., 2009a. Improving natural language specifications with ontologies. In: Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering. World Scientific, pp. 552–557.

Körner, S.J., Brumm, T., 2009b. Natural language specification improvement with ontologies. Int. J. Semant. Comput. 03 (04), 445–470. doi:10.1142/S1793351X09000872.

Körner, S.J., Brumm, T., 2009. RESI – A natural language specification improver. In: Proceedings of the 2009 IEEE International Conference on Semantic Computing. IEEE, pp. 1–8. doi:10.1109/ICSC.2009.47.

Krisch, J., Houdek, F., 2015. The myth of bad passive voice and weak words: an empirical investigation in the automotive industry. In: 23rd IEEE International Requirements Engineering Conference, pp. 344–351.

Lamsweerde, A.V., 2009. Requirements Engineering. John Wiley & Sons.

Lucassen, G., Dalpiaz, F., Brinkkemper, S., van der Werf, J., 2015. Forging high-quality user stories: towards a discipline for agile requirements. In: 23rd IEEE International Requirements Engineering Conference, pp. 126–135.

Lucia, A.D., Fasano, F., Oliveto, R., Tortora, G., 2007. Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans. Software Eng. Methodol. 16 (4). doi:10.1145/1276933.1276934.

Ludewig, J., Lichter, H., 2010. Software Engineering, 2nd ed. dpunkt.verlag.

Mavin, A., Wilkinson, P., Harwood, A., Novak, M., 2009. EARS (Easy approach to requirements syntax). In: Proceedings of the IEEE International Conference on Requirements Engineering, pp. 317–322. doi:10.1109/RE.2009.9.

Méndez Fernández, D., Mund, J., Femmer, H., Vetrò, A., 2014. In quest for requirements engineering oracles: dependent variables and measurements for (good) RE. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM, pp. 3:1–3:10.

Méndez Fernández, D., Wagner, S., 2015. Naming the pain in requirements engineering: a design for a global family of surveys and first results from Germany. Inf. Software Technol. 57 (1), 616–643.

Menzies, T., Dekhtyar, A., Distefano, J., Greenwald, J., 2007. Problems with precision: a response to "Comments on 'data mining static code attributes to learn defect predictors'". IEEE Trans. Software Eng. 33 (9), 637–640. doi:10.1109/TSE.2007.70721.

Mich, L., Franch, M., Novi Inverardi, P.L., 2004. Market research for requirements analysis using linguistic tools. Requirements Eng. 9 (2), 151. doi:10.1007/s00766-004-0195-3.

Mund, J., Femmer, H., Méndez Fernández, D., Eckhardt, J., 2015. Does quality of requirements specifications matter? Combined results of two empirical studies. In: Proc. of the 9th International Symposium on Empirical Software Engineering and Measurement, pp. 1–10.
Parachuri, D., Sajeev, A., Shukla, R., 2014. An empirical study of structural defects in industrial use-cases. In: Proceedings of the International Conference on Software Engineering. ACM, pp. 14–23.
Porter, M., 1980. An algorithm for suffix stripping. Program 14 (3), 130–137. doi:10.1108/eb046814.
Rago, A., Marcos, C., Diaz-Pace, J.A., 2014. Identifying duplicate functionality in textual use cases by aligning semantic actions. Software Syst. Model. 1–25. doi:10.1007/s10270-014-0431-3.
Runeson, P., Höst, M., 2008. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Eng. 14 (2), 131–164. doi:10.1007/s10664-008-9102-8.
Runeson, P., Höst, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering. Guidelines and Examples. Wiley.
Salger, F., 2013. Requirements reviews revisited: residual challenges and open research questions. In: Proceedings of the 2013 21st IEEE International Requirements Engineering Conference. IEEE, pp. 250–255.
Schmid, H., Laws, F., 2008. Estimation of conditional probabilities with decision trees and an application to fine-grained POS tagging. In: Proceedings of the Conference on Computational Linguistics. Association for Computational Linguistics, pp. 777–784.
Schneider, F., Berenbach, B., 2013. A literature survey on international standards for systems requirements engineering. In: Proceedings of the Conference on Systems Engineering Research, vol. 16, pp. 796–805. doi:10.1016/j.procs.2013.01.083.
Schwaber, K., Sutherland, J., 2011. The scrum guide. Technical Report. Scrum.org.
Shull, F., Rus, I., Basili, V., 2000. How perspective-based reading can improve requirements inspections. Computer 33 (7), 73–79. doi:10.1109/2.869376.
Tjong, S.F., Berry, D.M., 2013. The design of SREE – a prototype potential ambiguity finder for requirements specifications and lessons learned. In: REFSQ. Springer, Berlin, Heidelberg, pp. 80–95. doi:10.1007/978-3-642-37422-7_6.
Toutanova, K., Klein, D., Manning, C.D., 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, vol. 1, pp. 252–259. doi:10.3115/1073445.1073478.June
Wagner, S., Jürjens, J., Koller, C., Trischberger, P., 2005. Comparing bug finding tools with reviews and tests. In: Proceedings of Testing of Communicating Systems. Springer, pp. 40–55.
Wilson, W.M., Rosenberg, L.H., Hyatt, L.E., 1997. Automated analysis of requirement specifications. In: Proceedings of the International Conference on Software Engineering. ACM, pp. 161–171. doi:10.1109/ICSE.1997.610237.
Zelkowitz, M.V., Yeh, R., Hamlet, R.G., Gannon, J.D., Basili, V.R., 1983. The software industry: a state of the art survey. In: Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili, vol. 1, p. 383.
Zhang, M., Hall, T., Baddoo, N., 2011. Code bad smells: a review of current knowledge. J. Software Main. Evol. 23 (3), 179–202. doi:10.1002/smr.

Henning Femmer holds an MSc in Software Engineering with honors from Technical University of Munich, Ludwig-Maximilians University Munich, and the University of Augsburg. His main research interest is the quality of requirements specifications.

Daniel Méndez Fernández studied computer science at the Ludwig-Maximilians University Munich. He received his PhD and subsequently his habilitation in Computer Science from Technical University of Munich. His research covers empirical software engineering with a particular focus on requirements engineering.

Stefan Wagner studied computer science in Augsburg and Edinburgh and holds a PhD in computer science from Technical University of Munich. Since 2011, he is a full professor of software engineering at the University of Stuttgart. His research includes work on software quality, requirements engineering, safety & security engineering and agile/lean/continuous software development.

Sebastian Eder holds an MSc in Software Engineering with honors from Technical University of Munich, Ludwig-Maximilians University Munich, and the University of Augsburg. His main research interest is software maintenance based on software usage.

Publication H: Quality Assurance of Requirements Artifacts in Practice: A Case Study and a Process Proposal

Authors Henning Femmer, Benedikt Hauptmann, Sebastian Eder, Dagmar Moser

Venue 17th International Conference on Product-Focused Software Process Improvement (PROFES2016)

Abstract Requirements artifacts build the basis for various software engineering activities, such as development, testing or effort estimations. As such, the quality of requirements artifacts impacts the efficiency and effectiveness of these activities. Consequently, requirements artifacts should be subject to quality assurance (QA).

Unfortunately, QA of requirements artifacts struggles in practice. We contribute a first industrial case study, in which we found that the main problems in QA for requirements artifacts in this case were a missing common quality understanding, the low feedback speed, low efficiency in the QA process, and, consequently, the lack of creating a sustaining QA processes.

Based on these results, we furthermore contribute a process for requirements artifact QA that is designed to address these problems. We discuss feasibility and impact of the process with industry, who acknowledge its potential to increase efficiency and to provide a more sustaining QA process in practice.

Extended Summary This paper is summarized in Section 4.3.1.

Authors Contributions I co-designed and co-executed the interviews, and analyzed and co-reported the results.

Reprint Denied The reprint of this publication was rejected on open-access platforms. The publication can be found at https://link.springer.com/. The details are provided below.

In: P. Abrahamsson et al. (Eds.): PROFES 2016, LNCS 10027, pp. 506–516, 2016.
DOI: 10.1007/978-3-319-49094-6 36
Product-Focused Software Process Improvement, Quality Assurance of Requirements Artifacts in Practice: A Case Study and a Process Proposal, 10027, 2016, pp. 506–516, Henning Femmer, Benedikt Hauptmann, Sebastian Eder, Dagmar Moser

Publication I: Which requirements artifact quality defects are automatically detectable? A case study

Authors Henning Femmer, Michael Unterkalmsteiner, Tony Gorschek

Venue Accepted for publication at the Fourth International Workshop on Artificial Intelligence for Requirements Engineering (AIRE'17) at the 2017 IEEE 25th International Requirements Engineering Conference (RE)

Abstract The quality of requirements engineering artifacts, e.g. requirements specifications, is acknowledged to be an important success factor for projects. Therefore, many companies spend significant amounts of money to control the quality of their RE artifacts. To reduce spending and improve the RE artifact quality, methods were proposed that combine manual quality control, i.e. reviews, with automated approaches.

So far, we have seen various approaches to automatically detect certain aspects in RE artifacts. However, we still lack an overview what can and cannot be automatically detected.

Starting from an industry guideline for RE artifacts, we classify 166 existing rules for RE artifacts along various categories to discuss the share and the characteristics of those rules that can be automated. For those rules, that cannot be automated, we discuss the main reasons.

We estimate that 53% of the 166 rules can be checked automatically either perfectly or with a good heuristic. Most rules need only simple techniques for checking. The main reason why some rules resist automation is due to imprecise definition.

By giving first estimates and analyses of automatically detectable and not automatically detectable rule violations, we aim to provide an overview of the potential of automated methods in requirements quality control.

Extended Summary This paper is summarized in Section 4.4.

Authors Contributions I co-designed and co-executed the study, and analyzed and co-reported the results.

Publication Please find below the preprint accepted for publication.

Which requirements artifact quality defects are automatically detectable? A case study

Henning Femmer
Institut für Informatik
Technische Universität München, Germany
femmer@in.tum.de

Michael Unterkalmsteiner, Tony Gorschek
Software Engineering Research Lab,
Blekinge Institute of Technology, Sweden
{mun,tgo}@bth.se

***Abstract*—[Context:] The quality of requirements engineering artifacts, e.g. requirements specifications, is acknowledged to be an important success factor for projects. Therefore, many companies spend significant amounts of money to control the quality of their RE artifacts. To reduce spending and improve the RE artifact quality, methods were proposed that combine manual quality control, i.e. reviews, with automated approaches. [Problem:] So far, we have seen various approaches to automatically detect certain aspects in RE artifacts. However, we still lack an overview what can and cannot be automatically detected. [Approach:] Starting from an industry guideline for RE artifacts, we classify 166 existing rules for RE artifacts along various categories to discuss the share and the characteristics of those rules that can be automated. For those rules, that cannot be automated, we discuss the main reasons. [Contribution:] We estimate that 53% of the 166 rules can be checked automatically either perfectly or with a good heuristic. Most rules need only simple techniques for checking. The main reason why some rules resist automation is due to imprecise definition. [Impact:] By giving first estimates and analyses of automatically detectable and not automatically detectable rule violations, we aim to provide an overview of the potential of automated methods in requirements quality control.**

***Index Terms*—Requirement Engineering, Artifact Quality, Automated Methods**

I. Introduction

Requirements Engineering (RE) artifacts play a central role in many systems and software engineering projects. Due to that central role, the quality of RE artifacts is widely considered a success factor, both in academia, e.g. by Boehm [1] or Lawrence [2], and also by practitioners [3].

As a result, companies invest heavily into quality control of RE artifacts. Since RE artifacts are written mostly in natural language [4], quality control is usually applied manually, e.g. in the form of manual reviews. However, besides all of its advantages, manual quality control is slow, expensive and inconsistent, heavily dependent on the competence of the reviewer. One obvious approach to address this is combining manual reviews with automated approaches. The goal of a so-called *phased inspection* [5], [6] is to reduce the effort in manual reviews and to improve the review results by starting into the review with a better (e.g. readable) artifact.

Therefore, various authors have focused on automatically detecting quality defects, such as ambiguous language (i.a. [7], [8], [9], [10]) or cloning [11]. However, it is still an open question to what degree quality defects can be detected automatically or require human expertise (i.e. manual work). In previous work [10], we took a bottom-up perspective by qualitatively analyzing which of the quality review results could be automatically detected.

Research Goal: In this work, we take a top-down perspective by focusing on requirements writing guidelines from a large company. Furthermore, we systematically classify and quantify which proportion of the rules can be automated.

II. Related Work

Researchers and practitioners have been working on supporting quality assurance with automated methods (at least) since the end of the 1990's [7]. We want to give only a brief, non-exhaustive summary here. Please refer to our previous work [10] for a more detailed analysis.

Defect types: Most works in this area focus on the detection of various forms of ambiguity, e.g. [8], [12], [13], [14]. Other works try to detect violations of syntactic [11] or even semantic duplications [15]. Other works focus on correct classifications [16] or on the question whether an instance follows given structural guidelines, e.g. for user stories [9] or for use cases [17].

Criteria: The aforementioned works used different sets of criteria. Most prominently are definitions of ambiguity [18], previously summarized lists of criteria [19], or requirements standards [10], [20].

Techniques: So far, various techniques have been applied, including machine learning [16], [21] and ontologies [22]. However, Arendse and Lucassen [23] hypothesize that we might not need sophisticated methods for most aspects of quality. In this paper, we provide data regarding this hypothesis. All in all, few works have tried to take a different viewpoint and understand what *cannot* be automatically checked. In previous work [10], we approached this question in a qualitative manner, by looking not at definitions, but at instances of defects. We did not quantify the portion of automatically discoverable defects, since this depends heavily on the requirements at hand (which defects does an author introduce and a reviewer find?).

Research Gap: Various authors have shown how to automatically detect individual quality defects. In previous work [10] we qualitatively analyzed which requirements quality defects can be detected. In this work, we provide first

evidence, based on requirements writing rules used in a large organization, on the proportion between automatically / not automatically detectable requirements quality issues.

III. Study Design

We conducted this study in a research collaboration with the Swedish Transport Administration (STA), the government agency responsible for planning, implementing and maintaining long-term rail, road, shipping and aviation infrastructure in Sweden. In particular, we studied their requirements guidelines that were developed by editors who review and quality assure specifications. A total of 129 rules were analyzed in this paper. While our long-term goal in this research collaboration, is described in more detail elsewhere [24], the specific research goal of this paper is to *characterize requirements writing rules with respect to their potential to be automatically checked from the viewpoint of a requirements quality researcher in the context of an industrial requirements quality control process*. From this goal definition we derive our research questions:

RQ1: How many rules for natural language requirements specifications can be automated?

RQ2: To what degree can rules be categorized into groups and to what degree can these groups be eligible for automation?

RQ3: What information is required to automatically detect rule violations?

RQ4: Which rules resist automation and why?

A. Rule classification

A lack of classification schema for requirements writing rules prompted us to formulate the following schema (see Tbl. I).

1) Rule type: We distinguish between the lexical, grammatical, structural and semantic rule type (see rules 160, 56, 78 and 81 in Tbl. I). A lexical rule refers to constraints on the use of certain terms or expressions that may induce ambiguity, reduce understandability or readability. Similarly, a grammatical rule refers to constraints on sentence composition. A structural rule refers to the form in which information is presented and formatted. Finally, a semantic rule refers to constraints on the text content and meaning.

2) Rule context: We introduced this dimension to characterize in which context of the requirements specification the rule is relevant. An appropriate automated check flags only violations that occur in the correct context, e.g. in requirements (if they are separated from informative text), figures, tables, references, headings, enumerations, comments.

3) Information scope: This dimension describes the scope that needs to be considered in order to decide whether the rule is violated or not. We defined five levels: word/phrase, sentence, section, document and global. For example, to check rule 56 in Tbl. I, it is enough to inspect a sentence. However, rule 24 requires access to information that is not in the requirements specification, hence we classified it as global information scope. This characterization provides indication that can be used to estimate the relative required effort to implement the automated check of the rule.

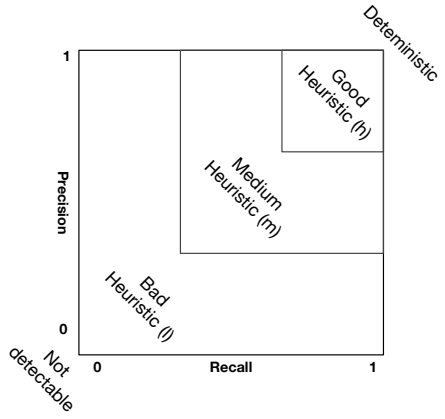


Fig. 1. The categories of detection accuracy as used in this study

4) Necessary information: This dimension describes NLP-based and domain-specific information needed to detect rule violations. NLP-based information refers to language and document structure, such as Part-of-Speech (POS) tags, lemmas and word stems, morphological tags, parse trees and meta-data on formatting. Domain-specific information is only available in the specific domain in which the rules apply, e.g. lists of referenced documents or a domain model / ontology. For example, rule 50 in Tbl. I can be decided with POS tags while rule 56 requires a parse tree that indicates where the subject is positioned in the sentence.

5) Detection accuracy: This dimension provides a rough estimate, based on the experiences of previous work [20], on the expected accuracy for detecting rule violations. We have defined a five-level scale, illustrated in Fig. 1, spanning from deterministic, i.e. 100% detectable, to not detectable at all. Good heuristics feature both high recall and precision, while bad heuristics always trade-off between precision and recall. For example, while assigning POS tags is a probabilistic algorithm, we classified rule 50 in Tbl. I as a good heuristic since this particular problem has been solved before, with demonstrably high precision and recall. We classified rule 81, on the other hand, as bad heuristic since, while conceptually feasible, we lack an accurate solution, i.e. a technique to extract a domain model and use that to determine whether a requirement statement contains supplemental information. Then, there are also rules that we do not expect to be automatically detectable at all (rule 54), because they turn out to be challenging, even in manual reviews. We classified these not automatically detectable rules along main reasons (categories resulted from previous work [10], see Tbl. III).

B. Data Collection, Classification and Analysis

We received a total of 192 writing rules from STA, of which we filtered unapproved rule ideas (63), resulting in 129 original rules. In case a rule contained discernible sub-rules, we split them up to facilitate the classification, resulting in 166 classified rules. We then developed an initial version of the classification schema illustrated in Section III-A. While all

TABLE I
CLASSIFICATION SCHEMA WITH RULE EXAMPLES

ID	Rule	Type	Context	Scope	Necessary information	Detection accuracy
160	The term "function" shall be used instead of the term "functionality".	Lexical	Anywhere	Word/Phrase	Lemma / Dictionary	Deterministic
56	Requirements shall start with the subject.	Grammatical	Requirement	Sentence	Parse tree	Heuristic (h)
78	Text consisting of a definition shall be preceded with the identifier "Definition:".	Structural	Requirement	Section	Lemma / Dictionary	Heuristic (m)
81	If a functional requirement is supplemented with additional information to clarify how the requirement can be met, the additional information must be formulated as a separate requirement.	Semantic	Requirement	Section	Domain model	Heuristic (l)
24	References to other documents in the specification are done by reference to the document title.	Structural	Anywhere	Global	Regular expressions, Document list	Deterministic
50	Requirements must be understandable independently, i.e. the subject must be indicated in the respective requirements (the subject must not be only defined in the section title).	Semantic	Requirement	Sentence	POS tags	Heuristic (h)
54	The introductory section of the specification shall not contain any requirements.	-	-	-	-	Not detectable

dimensions and the categories for type and detection accuracy were defined a-priori, the categories for context, scope and necessary information were identified during the classification process. During this first workshop we classified 39 rules, stabilizing the schema and fostering our shared understanding. Then, the second author proceeded to classify the remaining 127 rules alone. The first author sampled 20 rules from this set, independently classified them and calculated the inter-rater agreement ($\kappa = 0.79$) which is considered substantial [25]. The first author then reviewed all 127 rules, marked those where he disagreed, and finally consolidated all classifications with the second author in a second workshop.

We then used the classifications of accuracy for RQ1, the type, context and scope for RQ2, the necessary information for RQ3, and the reasons for RQ4.

IV. RESULTS

RQ1: How many rules for natural language requirements specifications can be automated?

In Fig. 2, we show the results from classifying the estimated detection accuracy of the rules. We estimate that 41% of the rules can be deterministically checked, meaning that an algorithm finds each violation. 34% of the rules are heuristic, with 12% of high accuracy, and 11% of medium and low accuracy. We estimate that the remaining 25% cannot be checked at the current state of art and at the current state of the rule definitions.

Discussion: Whether rules can be automatically detected is not a binary question. In fact, it depends on the context. However, most rules we can put into a certain category, indicating their potential to be automatically checked. We were surprised by the large number of rules that can be automated. This indicates the potential for automation, as we will discuss in future work.

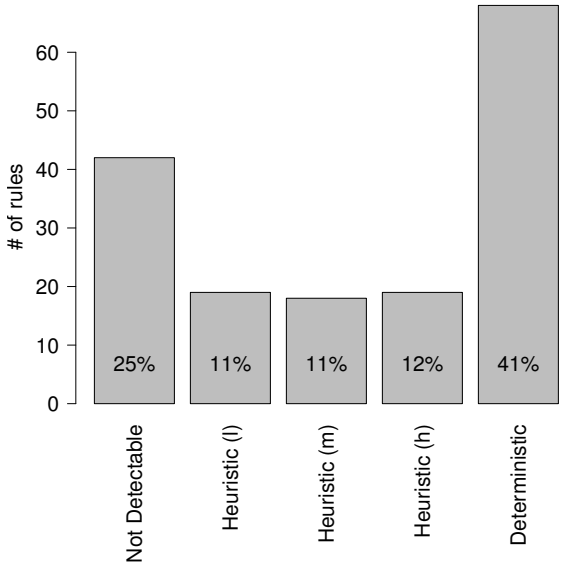


Fig. 2. Frequency of rules falling into one of the detection accuracy categories.

RQ2: To what degree can rules be categorized into groups and to what degree can these groups be eligible for automation?

In Fig. 3, we show the results from classifying the automatically detectable rules by their type and estimated detection accuracy. The results indicate an estimated high detection accuracy for structural and lexical rules, medium accuracy for grammatical rules, and medium to low accuracy for semantic rules. Fig. 4 shows that most rules are at the level of words or phrasing or at the level of sentences. Lastly, Fig. 5 shows that most rules hold anywhere or specifically concern the requirements of the RE artifact.

Discussion: The further a rule goes into semantic aspects, the harder it is to detect violations. For structural rules, e.g. where

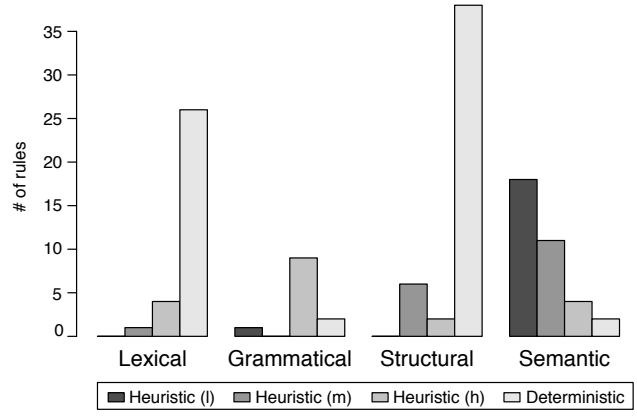


Fig. 3. Estimated detection accuracy for each category.

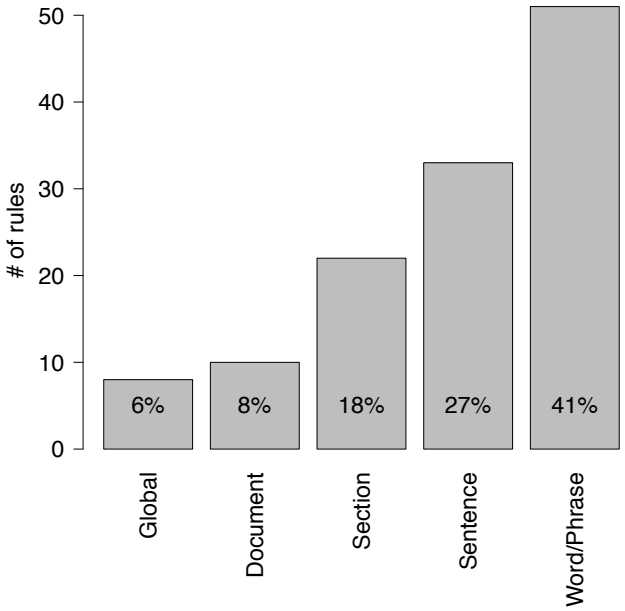


Fig. 4. Distribution of the scope of the automatically detectable rules.

a certain piece of information should be placed, there are a few rules for which violations are difficult to check automatically. For example, to understand whether a certain text should be tagged as a requirement requires context understanding. We describe further reasons for rules not being automatically detectable in RQ4.

RQ3: What information is required to automatically detect rule violations?

To understand what techniques are required to automatically detect violations of guideline rules, we classified each rule with the required information for this rule. Each required information then leads to a certain technique. For example, if the lemmas of the words are required, we obviously need a lemmatization technique. Tbl. II shows the results for this analysis. The three most common techniques are the following: In 47% of the cases, lemmatization is required to detect a violation of a rule. In a further 35% of cases only the pure text and regular expressions are needed. Next, formatting information is required in 22% of the cases.

Discussion: This analysis supports the hypothesis of Arendse and Lucassen [23] that in most cases, we do not need sophisticated methods to detect violations of rules.

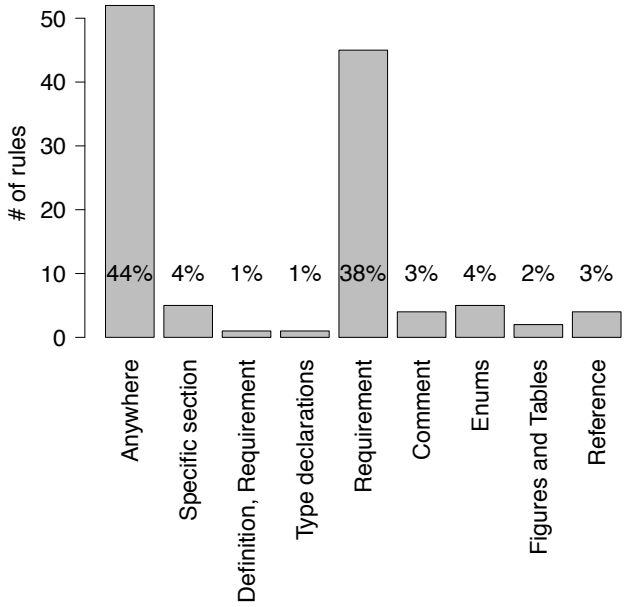


Fig. 5. Context of the automatically detectable rules.

TABLE II
FREQUENCY OF REQUIRED INFORMATION (MULTIPLE SELECTIONS)

Information	Occurrences	Share of Rules
Lemmas / Dictionaries	58	47 %
Pure Text (Reg. Expression)	43	35 %
Formatting	27	22 %
Domain Models	11	9 %
Part of Speech Tags	11	9 %
Lists of [X]	8	6 %
Morphology	5	4 %
Parse Trees	3	2 %
Word Stems	3	2 %
Tokens / Sentences	3	2 %
Named Entities	1	1 %

RQ4: Which rules resist automation and why?

When analyzing the *not automatically detectable* rules of RQ1, the reasons were distributed as shown in Tbl. III (classification extends previous work [10]). The major reason was, in our studied case, that the rules themselves are still imprecise or unclear. Examples for this are rules such as *"Requirements must be accurate, unambiguous, comprehensive, consistent, modifiable, traceable."* (this was one single rule) or *"Requirements should contain enough information."* These rules cannot be checked either manually or automatically. One could even argue that they convey little value. Such imprecise or unclear

rules are the reason for 81% of the not automatically detectable rules (see Tbl. III). In 12% of the cases, an automation would need profound domain knowledge to automatically detect a violation. An example is that requirements about certain system parts must first state that these parts exist. However, to understand which parts this refers to, we would need to know the domain. This means that only domain experts can manually detect violations to these rules. In one case, respectively, the rule requires deep semantic understanding of the text (e.g. to detect logical contradictions written in natural language in different paragraphs), the system or even the process scope.

TABLE III
SHARE OF REASONS THAT PREVENT AUTOMATED DETECTION

Reason	Frequency	Share
R_1: Rule unclear or imprecise	34	81 %
R_2: Deep semantic text understanding	1	2 %
R_3: Profound domain knowledge	5	12 %
R_4: System scope knowledge	1	2 %
R_5: Process status knowledge	1	2 %
Sum	42	100 %

Discussion: Deep computational problems do not seem to be the major cause for why we see no chance in checking a certain rule, rather imprecise rules themselves.

V. DISCUSSION

A. Share of automatically detectable defects

In our study, we found that a substantial number of requirements writing rules can be automatically checked. This is a top-down perspective and as such helps to quantify the share of defects that can be automatically detected. However, this does not necessarily transfer to the share of defects found in reviews. This is for the following reasons: First, defects created by requirements engineers are not equally distributed over the guideline rules. Furthermore, the defects introduced by requirements engineers very much depend on the individual person, company, and project. Second, defects discovered by reviewers are not necessarily equally distributed over the guideline rules. Therefore, we argue to consider both perspectives, i.e. the share of defects based on guidelines and the share of defects existing in practice, when discussing the potential of automated requirements quality assurance.

B. The 100%-Recall Argument

There is an ongoing debate in the scientific community whether automated checks in quality assurance need 100% recall to be useful in practice. Some authors (i.a. [26], [27], [28]) argue that if an approach does not achieve perfect recall, this leads to either the reviewer does not check the rule anymore, which would lead to unchecked defects, or the reviewer has to go through the whole document anyways, and thus, the automated analysis has no benefits. We disagree with this view for two reasons. First, we argue that in industrial practice, reviewers rarely go through the artifact rule by rule. Therefore, there is no such thing as *omitting a certain rule*. Reviewers see the guidelines rather as a supporting instrument, and thus anything that reminds them of certain rules, increases the quality. Our second argument also refers to the status quo today. The best automated quality support that is widely used are spell and grammar checks. Both do not have 100% recall. So, if recall is a problem, why do we use spell and grammar checks every day? In our experience from introducing automated analyses at various companies in industry, practitioners were more worried about precision than recall. They are convinced of the value (*"Anything helps!"*), and care more for acceptance with the end users. Here, the core aspect is usability in the form of few false positives, ergo: precision (cf. also similar discussing in static code analysis [29]).

C. Threats to Validity

There are two major threats to validity. Regarding internal validity, we classified the rules according detection accuracy. We did so because it was not feasible within the scope of this work to do a precision & recall analysis for each guideline rule. However, the first author has been translating guideline rules into automated analyses for 4 years. Thus, we are confident that the results reflect the real precision and recall after implementation. In addition, we created rough categories to gain an overview, not a precise analysis for each rule. To evaluate this aspect, we independently classified a subset of 10% of the rules and calculated a weighted Cohen's kappa of the resulting classification ($\kappa = 0.79$). This agreement fosters our confidence in the resulting classification.

The second threat relates to external validity. Since we analyzed a large guideline used at STA, we do not know whether the results generalize from this partner. We have, however, previously informally checked a guideline from another industry partner in a different domain. Here we came to the same share of not automatically detectable rules (25%). Future work should broaden the study to different guidelines.

VI. RESEARCH AGENDA

The current paper provides an estimation of the extent to which industrial requirements quality rules can be automatically checked. We plan to continue our research as follows.

Complete the rule classification. 34 of the studied rules were imprecise or unclear. Unfortunately, the authors of the writing guidelines were not available for feedback during the course of this study. We want to deepen our understanding on the nature of the imprecision of these rules. In addition, we had no access regarding the relevance, value, and frequency of violations of the rules. This could provide insights how rules that can be automatically checked potentially contribute to review effort reduction. In addition, the classification scheme used in RQ2 was beneficial for this study and worked fine regarding the first three categories (lexical, grammatical, structural). However, the scheme created some discussion around the semantic category. The reason is that most rules intertwine semantic and syntactic aspects: Since requirements artifacts are not automatically compiled like code, the point of syntactic rules is

only to prevent semantic issues. Therefore, future work should extend this classification scheme to clarify this aspect, e.g. by decoupling the two aspects.

Implement and statically validate rules. We have already begun to implement some rules that are based on dictionary lookups using an existing requirements smell detection framework [10]. While most of the rules can be implemented with simple techniques, we also plan to experiment with more advanced NLP techniques where we expect challenges in the detection accuracy. For example, violations to rule 81 in Table I could be detected by using topic models enhanced with domain knowledge [30]: requirements that contain distant topics or several closely related topics indicate candidates for rule violations. To validate the implemented rules, we can exploit the fact that at STA, the rules were developed based on experience, i.e. there exist versions of requirements that contain rule violations. We can fine-tune and validate the detection against this set. We also plan to provide an analysis of the potential benefits of using automated requirements quality control. To achieve this, we analyze historic requirements (where the current rules were not applied) and study the effort spent on discussing and repairing these violations.

Validation in Use. We plan to evaluate the efficiency and effectiveness of automated requirements quality assurance in use, i.e. in the environment of STA with the support of their requirements editors. One important question to answer is whether we can control the number of false positives, a crucial aspect for the adoption of tool support in industry that has also been observed in other areas, such as static bug detection [29].

Repository for requirements writing rules. Finally, we, as a community, should establish a repository of precise general and validated requirements rules. Such a repository can be created by replicating the work proposed in this paper in different contexts and, at the same time, advance the techniques for detecting rule violations.

VII. Conclusions

It is unclear what proportion of quality defects can be automatically detected. Therefore, in this work, we classify rules from a large, fine-grained requirements writing guideline from one of our industry partners. The results indicate that a surprisingly large proportion of rules (41%) can be automatically analyzed. 53% can be analyzed deterministically or with a good heuristic. One reason for this was that these rules contain many structural rules, which require just an analysis of formatting information or pure text. If we take also those rules into account where we have a medium heuristic, we could even tackle 64% of the rules. However, our analysis also shows that 36% of the rules have no or little chance to be automated. While being just first evidence, this analysis indicates that there is a substantial proportion of guideline rules (our intermediate for quality defects) that can be automatically checked. However, the analysis also indicates that there is little hope that we can completely replace manual reviewing with automated reviews. Combining automated and manual quality assurance, as proposed by others [5], and also ourselves [6] could be the promising compromise.

Acknowledgements

This work was performed within the project Q-Effekt and ERSAK; it was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS15003 A-B and by the Swedish Transport Administration. The authors assume responsibility for the content. The authors thank Jonas Eckhardt for comments on an earlier draft of this paper.

References

[1] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[2] B. Lawrence, K. Wiegers, and C. Ebert, "The top risks of requirements engineering," *IEEE Software*, pp. 62–63, 2001.

[3] D. Méndez Fernández and S. Wagner, "Naming the pain in requirements engineering: A design for a global family of surveys and first results from germany," *Information and Software Technology*, vol. 57, pp. 616–643, 2015.

[4] L. Mich, F. Mariangela, and P. L. Novi Inverardi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering Journal*, vol. 9, no. 1, pp. 40–56, 2004.

[5] J. C. Knight and E. A. Myers, "An improved inspection technique," *Communications of the ACM*, vol. 36, no. 11, pp. 51–61, 1993.

[6] H. Femmer, B. Hauptmann, S. Eder, and D. Moser, "Quality assurance of requirements artifacts in practice: A case study and a process proposal," in *PROFES*, 2016, pp. 506–516.

[7] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," in *ICSE*, 1997, pp. 161–171.

[8] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An automatic quality evaluation for natural language requirements," in *REFSQ*, 2001.

[9] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, "Improving agile requirements: the quality user story framework and tool," *Requirements Engineering*, vol. 21, no. 3, pp. 383–403, 2016.

[10] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[11] E. Juergens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schaetz, S. Wagner, C. Domann, and J. Streit, "Can Clone Detection Support Quality Assessments of Requirements Specifications?" in *ICSE*, 2010.

[12] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Application of linguistic techniques for Use Case analysis," *Requirements Engineering*, vol. 8, no. 3, pp. 161–170, 2002.

[13] E. Knauss, D. Lübke, and S. Meyer, "Feedback-Driven Requirements Engineering : The Heuristic Requirements Assistant," in *ICSE*, 2009.

[14] G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado, and V. Moreno, "A framework to measure and improve the quality of textual requirements," *Requirements Engineering*, vol. 18, no. 1, pp. 25–41, sep 2011.

[15] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 18–44, 2013.

[16] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *3rd International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2016.

[17] B. Alchimowicz, J. Jurkiewicz, J. Nawrocki, and M. Ochodek, "Towards use-cases benchmark," *Software Engineering Techniques*, 2011.

[18] D. M. Berry, E. Kamsties, and M. M. Krieger, "From Contract Drafting to Software Specification : Linguistic Sources of Ambiguity," 2003.

[19] D. M. Berry, A. Bucchiarone, S. Gnesi, G. Lami, and G. Trentanni, "A new quality model for natural language requirements specifications," in *REFSQ*, 2006, pp. 1–12.

[20] H. Femmer, D. Méndez Fernández, E. Juergens, M. Klose, I. Zimmer, and J. Zimmer, "Rapid requirements checks with requirements smells: Two case studies," in *International Workshop on Rapid Continuous Software Engineering*, 2014, pp. 10–19.

[21] H. Yang, A. D. Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements Engineering*, vol. 16, no. 3, pp. 163–189, 2011.

[22] S. J. Körner and T. Brumm, "Natural Language Specification Improvement With Ontologies," *International Journal of Semantic Computing*, vol. 3, no. 4, pp. 445–470, 2009.

[23] B. Arendse and G. Lucassen, "Toward tool mashups: Comparing and combining NLP RE tools," in *3rd International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2016, pp. 26–31.

[24] M. Unterkalmsteiner and T. Gorschek, "Requirements quality assurance in industry: why, what and how?" in *REFSQ*, 2017.

[25] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[26] D. M. Berry, R. Gacitua, P. Sawyer, and S. F. Tjong, "The case for dumb requirements engineering tools," *Lecture Notes in Computer Science*, vol. 7195 LNCS, pp. 211–217, 2012.

[27] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, no. 3, pp. 207–239, jul 2008.

[28] S. F. Tjong and D. M. Berry, "The design of SREE - A prototype potential ambiguity finder for requirements specifications and lessons learned," in *REFSQ*, 2013, pp. 80–95.

[29] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," *IEEE Software*, vol. 25, no. 5, pp. 22–29, 2008.

[30] D. Andrzejewski, X. Zhu, and M. Craven, "Incorporating domain knowledge into topic modeling via dirichlet forest priors," in *ICML*. ACM, 2009, pp. 25–32.