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

Conference Paper · May 2015

DOI: 10.1109/RET.2015.11

3 authors, including:

Henning Femmer
Technische Universität München
46 PUBLICATIONS **962** CITATIONS
SEE PROFILE

Daniel Méndez Fernández
Blekinge Institute of Technology
217 PUBLICATIONS **2,926** CITATIONS
SEE PROFILE

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.

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

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, 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

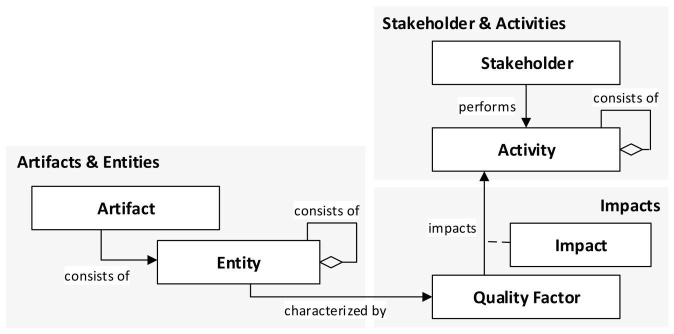


Fig. 1. Metamodel of the activity-based RE quality model based on [9]

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

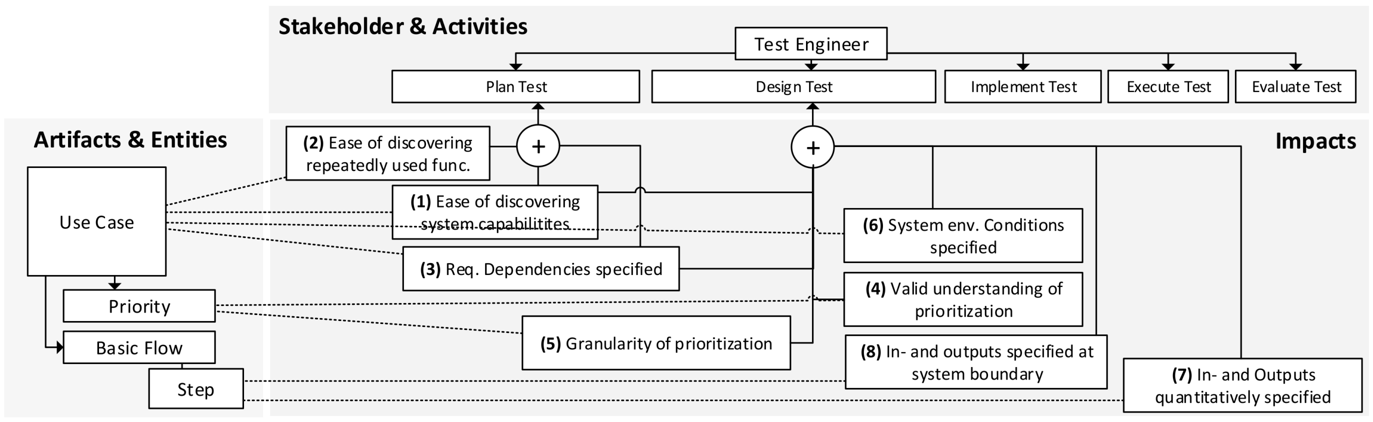
$$\textit{Valid understanding of prioritization @use case} \xrightarrow{+} \textit{Design Test}$$

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.



Stakeholder & Activities
Test Engineer
Plan Test
Design Test
Implement Test
Execute Test
Evaluate Test
Artifacts & Entities
Impacts
Use Case
Priority
Basic Flow
Step
(2) Ease of discovering repeatedly used func.
(1) Ease of discovering system capabilitites
(3) Req. Dependencies specified
(5) Granularity of prioritization
(6) System env. Conditions specified
(4) Valid understanding of prioritization
(8) In- and outputs specified at system boundary
(7) In- and Outputs quantitatively specified

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.

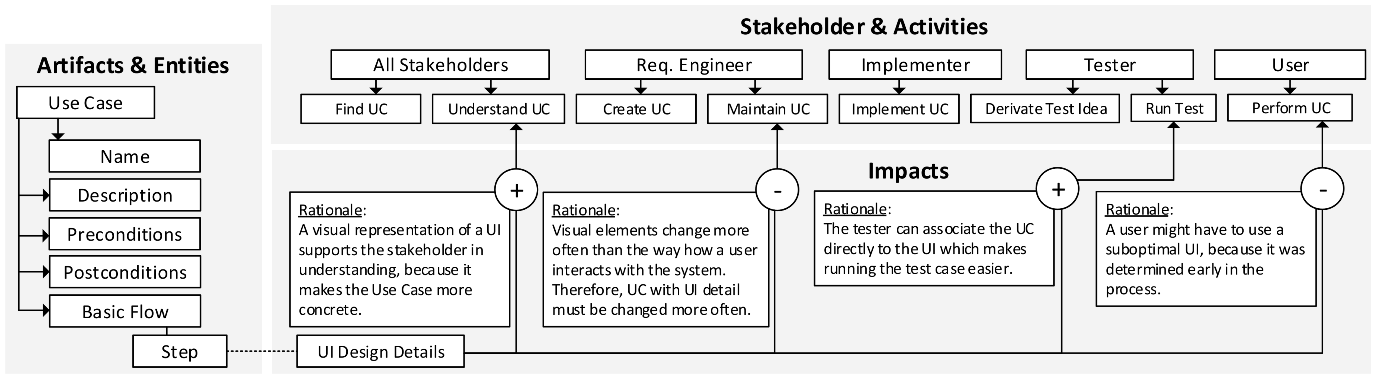

Artifacts & Entities
Use Case
Name
Description
Preconditions
Postconditions
Basic Flow
Step
UI Design Details
Stakeholder & Activities
All Stakeholders
Req. Engineer
Implementer
Tester
User
Find UC
Understand UC
Create UC
Maintain UC
Implement UC
Derivate Test Idea
Run Test
Perform UC
Impacts
+
-
+
-
Rationale:
A visual representation of a UI supports the stakeholder in understanding, because it makes the Use Case more concrete.
Rationale:
Visual elements change more often than the way how a user interacts with the system. Therefore, UC with UI detail must be changed more often.
Rationale:
The tester can associate the UC directly to the UI which makes running the test case easier.
Rationale:
A user might have to use a suboptimal UI, because it was determined early in the process.

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.