Measuring the Quality of System Specifications in Use Case Driven Approaches

Alexander Rauh[1], Wolfgang Golubski[2], Stefan Queins[3]

Abstract: One of the biggest challenges of a requirements analyst is to generate and provide a high-quality system specification in order to support other disciplines during system development. Today, there are only few mechanisms to measure the quality of requirements with less effort for the analyst. The following paper describes a meta-modeling and model-to-model transformation approach to formally evaluate different quality characteristics of system specifications like consistency and completeness in use case driven requirements analysis processes with less effort for the requirements analyst. Therefore, the mentioned concept integrates the information contained in different representations of requirements into a common requirements model and analyzes quality characteristics of the specification in two steps. In the first step, every representation within the specification will be evaluated separately according to predefined representation specific rules. In the second step after requirements integration, algorithms analyze the quality of the integrated information and calculate the overall characteristics of the specification.

Keywords: Requirements Quality, Specification Quality, Requirements Modeling, Meta-Modeling, Model-to-Model Transformation

1 Introduction

In systems engineering the system specifications are foundations for nearly every kind of discipline during development and after sales [Wa15]. Design, architecture and implementation transform the requirements of the specification into a set of components to realize the system and satisfy the needs of the customers. Testing and verification check the developed system against the system specification and ensure that the quality of the system fits the expectations. Additionally, during after sales the requirements of the system support the maintenance discipline to understand and improve the system's realization. In the context of this paper the term system addresses software systems as well as more technical systems consisting of software, mechanic and electric parts like cars.

For every purpose mentioned above high-quality requirements according to the characteristics listed in IEEE29148:2011 [IE11] have to be collected during

[1] University of Applied Sciences Zwickau, Dr.-Friedrichs-Ring 2a, 08056 Zwickau, alexander.rauh@fh-zwickau.de
[2] University of Applied Sciences Zwickau, Dr.-Friedrichs-Ring 2a, 08056 Zwickau, wolfgang.golubski@fh-zwickau.de
[3] SOPHIST GmbH, Vordere Cramergasse 13, 90478 Nuremberg, stefan.queins@sophist.de

requirements elicitation. Especially, in case that these requirements will be used as direct input for source code generation as mentioned in [Sm15] or if simulating the system under consideration before development as explained in [Po12]. At the moment, there are only a few approaches which aim at the assurance of the quality of system specifications. Furthermore, these approaches do not provide any mechanisms to evaluate this quality by numbers, need high effort to calculate some numbers or only evaluate some samples of a specification instead of the overall quality.

The concept discussed in this paper explains how to formally measure different quality characteristics of system specifications for a use case driven requirements analysis process using common notations for requirements documentation. In addition to the measurements this approach provides the sources of the defects in the documented requirements.

Following this introduction, there is a section to discuss some related works and already known approaches to measure the quality of requirements. The third section describes a concept and a process for requirements integration in order to measure the quality of system specifications. After that, there is a discussion of already existing quality characteristics of requirements and requirements specifications. Furthermore, interdependencies between the characteristics are explained. The fifth section describes the quality measurement process during requirements integration which checks a specification against defined requirements modeling and documentation rules. After an example to show the results of measuring the quality of a specification, the benefits of this approach will be explained. In the last section, there are some open issues which may be relevant for further researches.

2 Related Works

There are different approaches which provide to measure the quality of a system specification or support the requirements analyst to document a consistent and ideally complete set of requirements.

The first related approach described in [Go11] analyzes consistency and completeness of a specification via trace relations between the requirements. Therefore, the requirements analyst has to identify interdependency between the requirements and has to traces them manually. A tool evaluates the types of the relations and reports contradictions. To manage the trace relations defines [Go11] a meta-model for requirements which is similar to the meta-model of the Requirements Interchange Format (ReqIF) [OM16]. The approach mentioned in [Go11] evaluates only the quality of the trace relations between requirements but does not formally analyze the quality of the requirements content.

Another tool [Fa01] evaluates the quality of requirements in natural language. This tool analyzes the textual requirements for keywords and assigns quality attributes to these keywords. The quality attributes differ from the quality characteristics defined by IEEE 29148:2011 [IE11] and aim at defects in the language of the requirement sentence. For example, undefined multiplicities and vague terms are reported as defects. The tool in [Fa01] only supports textual requirements. Other representation types for requirements documentation like UML cannot be used for analysis purposes.

A fourth similar approach defined by [Da93] also evaluates the quality of textual requirements according to predefined attributes but does not provide an algorithm to measure these attributes formally. The concept only explains techniques for requirements analyst to evaluate the quality attributes manually. Disadvantage of the approach described in [Da93] is the very high effort for the analyst. Additionally, only textual requirements can be used for quality analysis.

Some other approaches consider less common representation types to evaluate the quality system specifications. Furthermore, these approaches do not provide possibilities to add more common representations.

For example, [Kr09] analyzes the consistency of requirements by capturing textually requirements, creating a UML use case model from these requirements manually and converting these models into a problem ontology. The consistency of this problem ontology is evaluated via reasoning and is matched to a domain ontology representing the domain knowledge related to the system's domain to discover further contradictions. UML use case models support only abstract views onto a system. For the detailed view onto the system's functionality other UML diagram types like activity diagrams or state charts have to be used, but are not supported by[Kr09].

A last similar approach described in [He96] applies algorithms for consistency checks to the formal Software Cost Reduction (SCR) tabular notation, but does not support more common requirements representation types like UML or textual requirements. Thereby, before applying this approach onto a specification, the requirements analyst has to transform a common system specification into the SCR notation and has to spend a lot of additional effort.

Approaches which use formal representation types of requirements like VHDL, MATLAB Simulink or different temporal logics are not discussed in this paper. These representation types are used in very specific domains but are usually not used for common system specifications.

3 Requirements Integration Concept for Quality Measurements

The idea to measure the quality of a system specification is to integrate information contained in the requirements into a common database and to evaluate the overall quality of this information. Thereby, representation specific information is encapsulated and the quality of the content described by requirements will be evaluated. Although today's requirements management tools provide mechanisms to store different views as described by [Kr95] or [Cz15] onto the system under consideration in a common model are the information within these views only loosely coupled. Interrelations are often simple references which the requirements analyst has to create and manage manually. Additionally, the tools do not check whether there is a relation in between the content of the referenced parts or not. Hence, the idea is to extract the information of these several views and to integrate it in a common requirements model which includes formal relations between the information. Once integrated, algorithms can evaluate this requirements model according to predefined rules which address several quality issues of the overall system specification like consistency and completeness.

This mentioned integration is realized via a meta-modeling and model-to-model transformation concept. Therefore, the concept is divided into a representation layer, an interpretation layer and an integration layer. Every layer contains at least one meta-model and one or more instances of these meta-models. In between these layers model-to-model transformations are applied.

The first layer is the representation layer which contains the system specification use for requirements integration purposes. In the context of this approach such a system specification is a set of different models which store the requirements for the system under consideration. These models are instances of common meta-models used for requirements documentation like the UML meta-model and its several diagram types or template-based textual requirements. Notation or representation type of requirements may be used as synonym for the meta-models of the representation layer.

The interpretation layer consists of representation type specific interpretation meta-models and one instance of each of these meta-models. In contrast to the UML there is one dedicated interpretation meta-model for the common UML diagram types for requirements documentation. The idea of the interpretation layer is to evaluate the quality, especially the syntactic correctness, of each view onto the system under consideration separately before requirements integration. Thereby, the requirements analyst gets feedback to the quality of each view and has the possibility to adjust the requirements before an overall quality evaluation. For example, this layer allows evaluating the use case view onto the system independent of the system's information model where the terms, used in the use cases, are defined.

The third layer is the integration layer consisting of a function-oriented meta-model for requirements and one instance of this meta-model which contains the integrated information of the different views onto the system under consideration. Algorithms evaluate the overall quality of the system specification according to this meta-model, to the representation specific and the comprehensive modeling rules. Function-oriented means that this meta-model only defines the structure of information and terms of functional requirements including the quality of service requirements of these functions. Organizational requirements related to the development process or project constraints like time and budget as mentioned in [Dö11] are not part of this meta-model. The function-oriented meta-model of the integration layer was already described and published in [Ra17] and is not explained in the context of this paper. Fig. 1 shows these layers and their interrelations including the most significant terms.

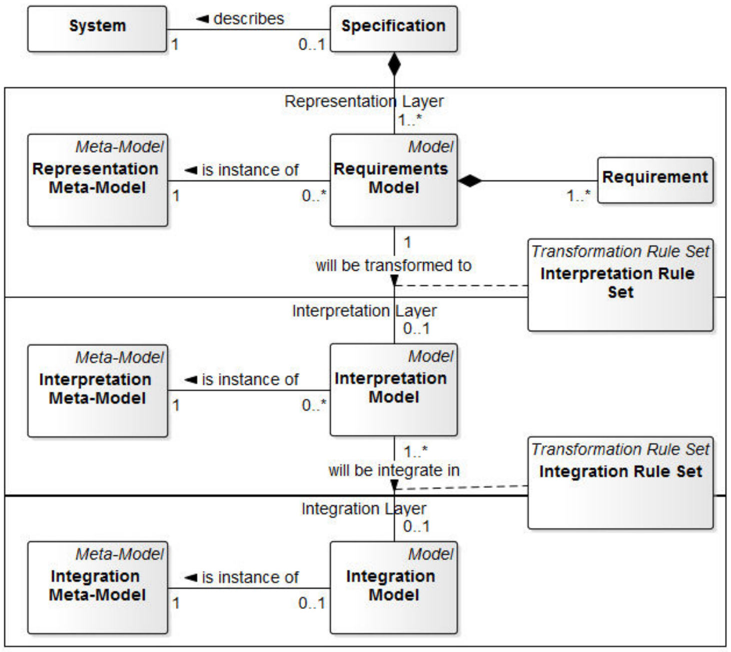

Fig. 1: Layer concept for requirements integration

In order to formally calculate the quality of the system specification several rules define how to use the different notations for requirements documentation. On the one hand, there are representation specific rules which address the usage of model elements within one view e.g. how to name a use case. Violations against these rules will be checked during transformation of the models in the representation layer into the models of the interpretation layer when applying the Interpretation Rule Set. For each violated rule, a so-called defect will be created. These representation specific defects are used to calculate quality characteristics for each view separately. On the other hand, there are representation comprehensive rules that aim at the interrelations of information in between the different views. After requirements integration algorithms check the integrated information according to these comprehensive rules and generate also defects in case of rule violations. Furthermore, the overall quality characteristics of the system specification will be measured. Fig. 2 gives an overview on the single steps of the requirements integration process which was described within this section. For each step, the stereotypes show the actor who performs this action.

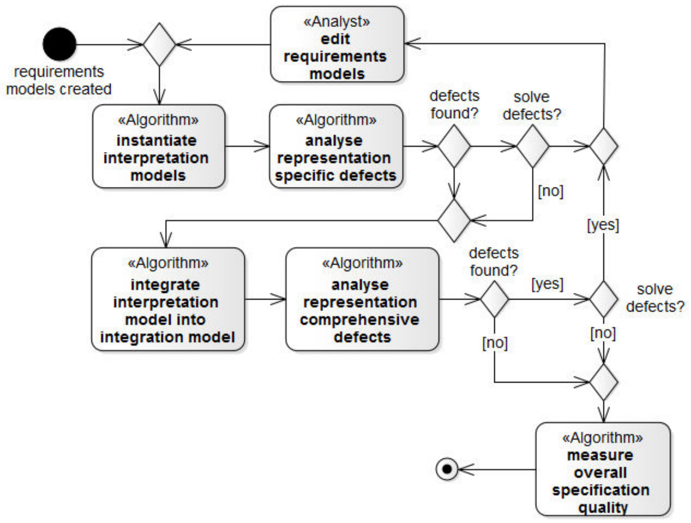
requirements models created
«Analyst» edit requirements models
«Algorithm» instantiate interpretation models
«Algorithm» analyse representation specific defects
defects found?
solve defects?
[no]
[yes]
«Algorithm» integrate interpretation model into integration model
«Algorithm» analyse representation comprehensive defects
defects found?
[yes]
solve defects?
[no]
[no]
«Algorithm» measure overall specification quality

Fig. 2: Process for requirements integration and quality measurement

Depending on the requirements analysis process and the guideline for requirements documentation in a specific development project it may be necessary to adjust the meta-models of the interpretation layer and the predefined requirements documentation rules in order to apply this concept. The fifth section shows some examples for representation specific and comprehensive documentation rules and shows a related interpretation meta-model derived from the use case driven requirements analysis process according to [Cz15].

4 Characteristics of Requirements and System Specifications

In order to define the term quality in a more precise way, this section discusses several characteristics of requirements and sets of requirements. The standard IEEE 29148:2011 serves as foundation for the discussion. First of all, [IE11] differs between characteristics for single requirements and requirements documents. Single requirements have to be:

- Necessary
- Implementation Free
- Unambiguous
- Consistent
- Complete
- Singular

- Feasible
- Traceable
- Verifiable

Some definitions of the characteristics listed above refer not only a single requirement but also consider the context of this requirement. For example, consistency means that the requirement is free of conflicts to other requirements [IE11]. Similar to consistency states the definition of completeness that the requirement does not need further refinements [IE11]. In the context of a system specification it is impossible to determine completeness without analyzing the requirements context which means other requirements addressing the same subject. In addition to the quality characteristics for single requirements IEEE 29148:2011 defines the following criteria for sets of requirements:

- Complete
- Consistent
- Affordable
- Bounded

In order measure the quality of a system specification, a separation between single requirements and a set of requirements is not necessary. Therefore, quality characteristics in the context of this approach always refer to a set of requirements and defects within the requirements have influence onto one or more of these characteristics.

One goal of the concept mentioned in this paper is to evaluate the quality of a system specification automatically using algorithms. But a few of the previously listed characteristic cannot be checked by a tool but only by the requirements analyst himself. For example, it is not possible to determine feasibility and affordability of requirements without knowledge and experience from similar development projects. Additionally, one essential characteristic is missing which provides the foundation for automatic quality checks by algorithms. To apply algorithms onto requirements these requirements have to be syntactically correct. IEEE 29148:2011 only defines semantic correctness as a task to be established during requirements analysis and maintenance. Semantic correctness means that the requirements express the intentions of the stakeholders [IE11].

In addition to this mentioned dependency, there are further interrelations between the quality characteristics. Inconsistency leads to the issue that completeness and necessity cannot be determined by algorithms. For example, if there is an actor with no associations to any use case within the use case view onto the system under consideration this actor might be not necessary or the use case view is incomplete due to a missing association. Whether this defect addresses necessity or completeness depends on the solution of the analyst to solve this defect. Algorithms are not able to make this decision.

Unambiguity has interrelations to consistency, necessity and completeness. As defined by IEEE 29148:2011 means unambiguity that there are no possibilities for different interpretations of information. Inconsistency, incompleteness and violations of necessity cause such possibilities for interpretation.

The concept described in this paper supports the measurement of correctness, completeness, consistency, unambiguity and necessity.

5 Rule-based Quality Measurement of Requirements

The quality of a system specification is measured according to predefined requirements documentation rules. Each documentation rule has assigned at least one of the quality characteristics listed in the previous section. Rule violations lead to defects which decrease the assigned characteristics.

For the application of this concept onto a system specification for validation purpose the rules for requirements documentation, the related interpretation meta-models and the required transformations were defined for the use case driven requirements analysis process according to [Cz15]. The snippets below show two different rules for requirements documentation including the assigned quality characteristics to give an idea of these rules. The first rule is representation specific and addresses the documentation of use cases. The second rule is comprehensive and addresses and interrelation between use cases and the information model of the system's domain.

Rule 1: Names of UML Use Cases follow the structure <verb> [adjective] <noun>.

Assigned quality characteristics:	Correctness
Criticality:	high

Rule 1 addresses the naming of use case. The requirements analysis process in [Cz15] recommends that the name of a use case should consist of a process and an object of the system's domain. UML does not even constrain the naming [OM15]. The rule above provides the option to add an adjective between the verb for the process and the noun for the object. This adjective could be used to add further information to the object like a state of this object. For example, a use case could be named like "archive existing user profile". In order to simplify the implementation of this rule, a name of a use case could only consist of two or three tokens. The first token is the verb which defines the process. If the name contains three tokens, the second token is a state and third is the object of the domain. Domain objects which consist of more than one token will be named in camel case e.g. "UserProfile". For further researches, this simplification could be eliminated by the integration of a natural language processing (NLP) tool like [Bj10]. This tool would also provide mechanisms to categorize the types of the words and to analyze the grammar of natural language parts in the representation models.

As mentioned before, there is at least one quality characteristic of the previous section assigned to each of the requirements documentation rules. If such a documentation rule is violated, the related quality characteristic will be decreased. For example, in case that the name of a use case violates rule 1, the correctness of system specification is affected.

The criticality of the modeling rules classifies the impact of a rule violation onto the further requirements integration process. High criticality means that the related representation model element cannot be integrated into the integration model. Low criticality violations have no effects onto the requirements integration but cause also a loss in the requirements quality.

Rule 2: The noun in the name of a use case is defined as a class within the information model of the system specification.

Related representations:	UML Use Case Diagram, UML Class Diagram
Assigned quality characteristics:	Consistency, Completeness
Criticality:	low

Rule 2 is a comprehensive modeling rule which addresses the usage of interrelations between UML use case diagrams and UML class diagrams for information modeling. The nouns in the names of use cases represent objects of the system's domain and thereby should be part of the information model. Consistency and completeness will be decreased, if rule 2 is violated. The impact of such a violation onto the requirements integration process is low.

In addition to these kinds of modeling rules, the mentioned requirements integration concept provides interpretation meta-models for the representation types required for a use case driven requirements analysis process according to [Cz15]. Fig. 3 shows the interpretation meta-model for UML use case diagrams. This meta-model is derived from eight use case specific requirements documentation rules.

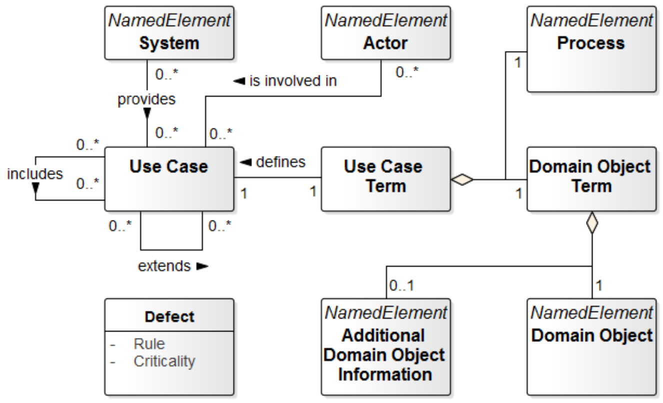

Fig. 3: Interpretation meta-model for UML use case diagrams

The core element of this meta-model is the *Use Case*, which represents functionality of the assigned *System* at high level of abstraction. In order to provide fault tolerance during instantiation of the use case interpretation model, the cardinalities of the relations between *Use Case* and *System* and *Use Case* and *Actor* according to the definition of use cases in [Ja92] are loosened from obligatory to optional.

In contrast to the UML, which defines the name of a use case as a non-empty string [OM15], the meta-model shown in Fig. 3 encapsulates the name of a use case as the separate class *Use Case Term*. Furthermore, the use case term and its parts depicts documentation rule 1 which defines the structure of the names of use cases.

Includes- and *extends*-associations of the use cases are simplified in the interpretation meta-model compared to the UML meta-model. The details of include- and extend-associations between use cases have to be documented in the control flows of the dedicated use cases. These control flows are parts of the activity of the system specification. Hence, there is no need to store further details for include- and extend-associations of use case diagrams within the interpretation model. The *Defect* will be used to store violations of use case specific requirements documentation rules. Such a defect persists the rule which was violated and the criticality of the violation in order to give the requirements analyst advises for editing the use case model.

Further interpretation meta-models for UML class diagrams, state charts and activity diagrams as well as the interpretation meta-models for glossary entries, functional and non-functional textual requirements will be presented later during researches due to the limited space in this paper.

Overall to measure the quality of system specifications there are 27 representation specific documentation rules, 13 comprehensive documentation rules and interpretation meta-models for:

- UML Use case diagrams
- UML Activity diagrams
- UML State Charts
- UML Class diagrams and glossary entries
- Template-based textual functional requirements
- Template-based textual non-functional requirements

The representation specific documentation rules are included in the Interpretation Rule Sets shown in Fig. 1 and the Integration Rule Set contains the comprehensive documentation rules. Tab. 1 lists the count of representation specific and comprehensive rules including the count of related quality characteristics.

Addressed	Rule	Affected Quality Characteristics				
Representation Type	**Count**	**Completeness**	**Correctness**	**Consistency**	**Unambiguity**	**Necessity**
Use Case Diagrams	8	1	7	1	4	2
Activity Diagrams	4	3	4	0	2	0
Class Diagrams	2	0	2	0	1	0
Glossary Entries	4	1	1	2	2	0
State Charts	3	0	3	0	0	0
Textual functional requirements	3	0	3	1	1	0
Textual non-functional requirements	3	0	3	1	0	0
Representation Comprehensive	13	10	13	13	0	0

Tab. 1: Representation specific and comprehensive modeling rules and quality characteristics

These meta-models are implemented using Eclipse EMF. The model-to-model transformations are realized using a combination of the ATLAS Transformation Language (ATL) and Java. The transformation rule sets are implemented in ATL. Java was used to coordinate the transformation steps and to analyze the defects within the interpretation layer and the integration layer in order to calculate the numbers for the quality characteristics.

6 Example for Integration Results

In order to explain the possibilities and results of the approach mentioned in the previous sections, the integration concept was applied onto the following two simple diagrams of a system specification of an online shop. The representation specific defects and the comprehensive defects of the specification are explained. Fig. 4 shows a use case diagram on the left and an information model on the right which were integrated. Additionally, there are two activity diagrams for “create CustomerAccount” and “buy

Article" within the specification which will be referenced to explain the integration results.

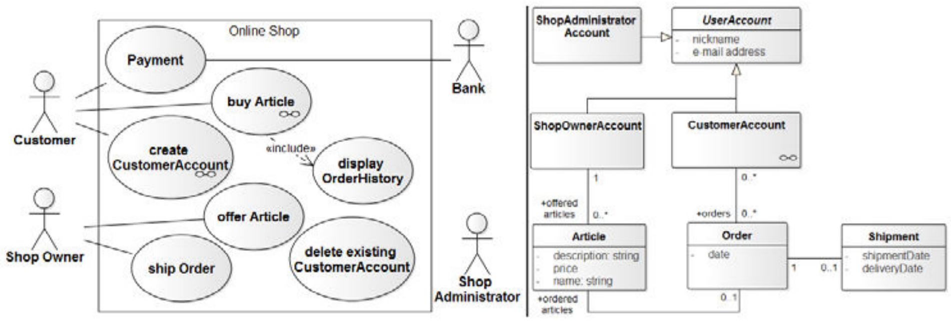

Fig. 4: Use cases and information model of an online shop

Tab. 2 shows the defects which were generated during instantiation of the interpretation model for the use case diagram in Fig. 4.

ID	Element	Defect	Related Quality Characteristics	Criticality
1	Payment	Incorrect name of use case	Correctness	high
2	display OrderHistory	missing association to any actor	Correctness, unambiguity	low
3	delete existing CustomerAccount	missing association to any actor	Correctness, unambiguity	low
4	Shop Administrator	missing association to any use case	Correctness, unambiguity	low

Tab. 2: Defects in the use case interpretation model

The first defect is the result of violated rule 1 which is explained in the previous section in detail. The name of the use case "Payment" does not consists of a verb and a noun but only contains a nominalization. This syntactic incorrectness leads to a high criticality of the first defect because the related use case cannot be instantiated within the interpretation model and, thereby, will be ignored in the further integration process.

The second and the third row address the violation of a documentation rule derived from the definition of use cases according to [Ja92]. Both use cases do not have an association to any actor and, thereby, may not be of value. Besides the impact onto the correctness of the use cases lead both defects to an ambiguity. The defect related use cases may be documented incomplete or they may be unnecessary. The requirements integration could be performed for both use cases. Hence, the criticality of the defects in the second and third row is evaluated as low.

The defect in the fourth row addresses the "Shop Administrator" without an association to any use case. The violation is similar to the previously explained missing associations. It has also an impact onto the correctness and the unambiguity of the specification but the requirements integration could be performed.

The class diagram on the left in Fig. 4 does not violate any representation specific documentation rules. After the instantiation of the interpretation models for the use cases and the information model the requirements analyst could edit the defects listed in Tab. 2 or the requirements integration could proceed. Tab. 3 shows the representation comprehensive defects after integration of the diagrams in Fig. 4.

ID	Element	Defect	Related Quality Characteristics	Criticality
1	display OrderHistory	noun OrderHistory is not part of the information model	Consistency, Completeness	low
2	display OrderHistory	activity "display OrderHistory" is missing	Consistency, Completeness	low
3	ship Order	activity "ship Order" is missing	Consistency, Completeness	low
4	offer Article	activity "offer article" is missing	Consistency, Completeness	low
5	delete existing CustomerAccount	activity "delete existing CustomerAccount" is missing	Consistency, Completeness	low
6	buy Article	control flow of activity "buy Article" does not contain call-behavior of activity "display OrderHistory"	Correctness, Consistency	low

Tab. 3: Representation comprehensive defects of the integrated requirements

The first defect is the result of a violation of rule two of the previous section. It aims at the interrelation between the nouns in the names of use cases and the classes in the information model. As shown in Fig. 4 exists no class “OrderHistory” within the information model. Consistency is decreased because of the contradictions between the different views and completeness is impacted due to obvious missing information.

The defects in rows two to five address the missing refinements of the use cases. Each use case has to be refined by one activity which defines the single steps of the control flow when executing the related use case. These three defects have also lead to inconsistencies as well as incompleteness of the specification.

The last row shows a violation of a documentation rule which aims at the semantic of includes-associations between use cases and their influence onto the control flows of these use cases. The activity of the included use case has to be part in the activity of the including use case as a call-behavior action. Due to the missing activity of “display OrderHistory” shown in the second defect, there is no possibility to add such a call-behavior action within the control flow of “buy Article”. The contradiction between the use case view and the activity view of these use cases is obviously an inconsistency. The correctness of the specification is also affected because of the wrong documentation of the includes-association. The six defects in Tab. 3 have a low criticality. They do not prevent the requirements integration.

7 Benefits

This approach provides mechanisms to formally measure the quality of systems specifications created during use case driven analysis process according to [Cz15] and reports the defects to the requirements analyst. There are two different kinds of measurements. On the one hand, the quality of each view onto the system is calculated separately. Thereby, the requirements analyst can focus and improve one dedicated view. On the other hand, the requirements integration provides to check the overall quality of the requirements in all views. Especially, the interrelations between the dedicated views onto system are hard to be checked manually and take a very high effort. The mentioned concept supports to check these interrelations automatically with less effort.

Regarding the identified defects of a system specification, the requirements analyst can decide which of these defects he wants to fix. Thereby, the level of the quality of a specification can be adjusted to the projects needs and constraints. For example, the level of quality for a specification for safety critical systems (e.g. cars or airplanes) has to be quite higher than for less safety critical systems (e.g. business software).

The process to measure the quality of a system specification is very lightweight, which allows the requirements analyst to apply the concept in different kinds of projects. For example, the concept can be applied iterative in agile development processes as well as in more traditional processes when reaching milestones.

8 Conclusion & further Researches

The concept mentioned in this paper measures the quality of system specifications according to predefined requirements modeling rules. For each rule, there is at least one quality characteristic assigned, which will be decreased if the related rule is violated. The modeling rules are classified in representation specific and representation comprehensive rules. The representation specific rules are checked during instantiation of the interpretation models when applying the Interpretation Rule Set onto the representation models. After that, the requirements analyst has the possibility to fix violations before the interpretation models will be integrated. After requirements integration, the representation comprehensive rules including the interrelations between the different views onto the systems are analyzed and the overall quality of the system specification is calculated.

For further researches, the definition of metrics for specifications might be interesting. These metrics could be calculated from the numbers of rule violations per type of model element in the requirements models. Thereby, the maturity of a system specification could be determined periodically during the requirements analysis process.

In order to eliminate simplifications of the natural language parts in the representation models, the integration of natural language processing tools to analyze parts of the natural language in the requirements models seems to be necessary. Thereby, the modeling rules for naming model elements like use case and activities could be more flexible and would provide an additional benefit.

The example section lists the result when applying the described concept onto a small system specification which consists of four diagrams with some interrelations. For further validation purposes, the application of this approach onto larger specifications or even during a requirements analysis process is necessary. Thereby, the integration results can be verified and compared to each other and issues of the current concept could become transparent.

References

[Bj10] Björkelund, Anders; Bohnet, Bernd; Hafdell, Love; Nugues, Pierre (2010): A High-performance Syntactic and Semantic Dependency Parser. In: Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations. Stroudsburg, PA, USA: Association for Computational Linguistics (COLING '10), S. 33–36. Online available http://dl.acm.org/citation.cfm?id=1944284.1944293.

[Cz15] Cziharz, Thorsten; Hruschka, Peter; Queins, Stefan; Weyer, Thorsten (2015): Handbook of Requirements Modeling IREB Standard. Version 1.1. Online available https://www.ireb.org/content/downloads/17-handbook-cpre-advanced-level-requirements-modeling/ireb_cpre_handbook_requirements-modeling_advanced-level-v1.1.pdf.

[Da93] Davis, A.; Overmyer, S.; Jordan, K.; Caruso, J.; Dandashi, F.; Dinh, A. et al. (1993): Identifying and measuring quality in a software requirements specification. In: [1993] First International Software Metrics Symposium. Baltimore, MD, USA, 21-22 May 1993, S. 141–152.

[Dö11] Dörr, Jörg (2011): Elicitation of a complete set of non-functional requirements. Stuttgart: Fraunhofer-Verl (PhD theses in experimental software engineering, 34).

[Fa01] Fabbrini, Fabrizio; Fusani, Mario; Gnesi, Stefania; Lami, Giuseppe (2001): An automatic quality evaluation for natural language requirements. In: Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ, Bd. 1, S. 4–5. Online available http://fmt.isti.cnr.it/WEBPAPER/P11RESFQ01.pdf.

[Go11] Goknil, Arda; Kurtev, Ivan; van den Berg, Klaas; Veldhuis, Jan-Willem (2011): Semantics of trace relations in requirements models for consistency checking and inferencing. In: Softw Syst Model 10 (1), S. 31–54.

[He96] Heitmeyer, Constance L.; Jeffords, Ralph D.; Labaw, Bruce G. (1996): Automated consistency checking of requirements specifications. In: ACM Trans. Softw. Eng. Methodol. 5 (3), S. 231–261.

[IE11] Institute of Electrical and Electronics Engineers. 2011: Systems and software engineering -- Life cycle processes --Requirements engineering.

[Ja92] Jacobson, Ivar (1992): Object-oriented software engineering: A use case driven approach. [New York] and Wokingham and Eng and Reading and Mass: ACM Press and Addison-Wesley Pub.

[Kr09] Kroha, Petr; Janetzko, Robert; Labra, José Emilio (2009): Ontologies in Checking for Inconsistency of Requirements Specification. In: Third International Conference on Advances in Semantic Processing (SEMAPRO). Sliema, Malta, S. 32–37.

[Kr95] Kruchten, Philippe (1995): The 4+1 View Model of Architecture: IEEE Software.

[OM15] Object Management Group, Inc. (2015): Unified Modeling Language. Version 2.5. Online available http://www.omg.org/spec/UML/.

[OM16] Object Management Group, Inc. (2016): Requirements Interchange Format™ (ReqIF™). Version 1.2. Online available http://www.omg.org/spec/ReqIF/1.2.

[Po12] Pohl, Klaus; Achatz, Reinhold; Hönninger, Harald; Broy, Manfred (2012): Model-based engineering of embedded systems: The SPES 2020 methodology. Berlin and New York: Springer.

[Ra17] Rauh, Alexander; Golubski, Wolfgang; Queins, Stefan (2017): A requirements meta-model to integrate information for the definition of system services. In: 2017 IEEE Symposium on Service-Oriented System Engineering: IEEE / Institute of Electrical and Electronics Engineers Incorporated.

[Sm15] Śmiałek, Michał; Nowakowski, Wiktor (2015): From Requirements to Java in a Snap. Model-Driven Requirements Engineering in Practice. Cham: Springer International Publishing (EBL-Schweitzer).

[Wa15] Walden, David D.; Roedler, Garry J.; Forsberg, Kevin; Hamelin, R. Douglas; Shortell, Thomas M. (2015): Systems engineering handbook: A guide for system life cycle processes and activities. 4th edition.