PAPER Special Section on Knowledge-Based Software Engineering

# Finding Incorrect and Missing Quality Requirements Definitions Using Requirements Frame

**Haruhiko KAIYA**[†,††a)], ***Nonmember and*** **Atsushi OHNISHI**[†††], ***Member***

**SUMMARY** Defining quality requirements completely and correctly is more difficult than defining functional requirements because stakeholders do not state most of quality requirements explicitly. We thus propose a method to measure a requirements specification for identifying the amount of quality requirements in the specification. We also propose another method to recommend quality requirements to be defined in such a specification. We expect stakeholders can identify missing and unnecessary quality requirements when measured quality requirements are different from recommended ones. We use a semi-formal language called X-JRDL to represent requirements specifications because it is suitable for analyzing quality requirements. We applied our methods to a requirements specification, and found our methods contribute to defining quality requirements more completely and correctly.

***key words:*** *requirements analysis, case frame, quality requirements, spectrum analysis*

## 1. Introduction

In order to write a requirements specification in high quality, we have to take the following points into account; correctness, consistency, unambiguity, completeness, rank of importance, stability, verifiability and traceability [1]. For functional requirements, these points are taken into account well, but there are still research challenges in non-functional or quality requirements. Quality requirements describe how well functions are accomplished [2], and they are very important. Some systems such as online computer aided instruction (CAI) systems or online shopping sites should be highly reliable, while others such as web browsers and desktop publishing systems should be usable. If quality requirements are not correctly specified in a requirements specification, software system may not be correctly developed. Because there are more problems in quality requirements than in functional requirements, the special issue was published in IEEE Software [2]. This introductory article of the issue focuses on the following three problems; implicit understanding of stakeholder, trade-offs among quality requirements and difficulty to measure and to track quality requirements.

We have focused on the last problem about measurement and traceability of quality requirements, and proposed a technique called "spectrum analysis for quality requirements" [3]. By using this technique, we can identify what kinds of quality requirements are stated in a requirements specification. Based on the amount of stated quality requirements, we can also identify the importance degree of each quality requirement in the specification. Because the input of this technique is the list of sentences, the technique can be easily applied to a requirements document, a design document, source code explanations and so on. We can thus achieve traceability about quality requirements over different software development phases [4]. However, this technique cannot perform full-automatically because it depends on expertise of an analyst who uses this technique. For example, the analyst has to investigate the meaning of natural language sentences even with the help of natural language processing techniques.

Using semi-formal notations is one of well-known ways for requirements analysis and definitions [5], [6]. Such kinds of notations enable us to explicitly specify the semantic structure in each requirement. Some of them have supporting techniques to convert a natural language sentence to their semi-formal notations [7]. Because whether quality requirements are mentioned in a requirement largely depends on such a semantic structure, semi-formal notations are suitable for quality requirements analysis. We have also proposed a requirements frame model that includes a semi-formal notation for requirements [8]. Our notation is based on the case grammar [9], and each case has its type, that is suitable for software requirements definition.

In this paper, we will present a method to measure quality requirements written in our semi-formal notation, and show its usefulness. With the help of the semi-formal notation for requirements, we find we can recommend what kinds of quality requirements should be defined in a requirement as well as we can measure quality requirements in it. We will thus present another method to recommend quality requirements to be defined in a specification. In addition, differences between recommended quality requirements and measured ones help us to find incomplete and incorrect parts in requirements. We assume there are some missing quality requirements if recommended quality requirements are larger than measured ones. We also assume there are some incorrect quality requirements if measured ones are larger than recommended ones. To confirm this assumption is also the goal of this paper.

The rest of this paper is organized as follows. In the

Manuscript received June 29, 2011.
Manuscript revised October 31, 2011.

[†]The author is with Shinshu University, Nagano-shi, 380–8553 Japan.
[††]The author is with National Institute of Informatics, Tokyo, 101–8430 Japan.
[†††]The author is with the Department of Computer Science, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.
a) E-mail: kaiya@shinshu-u.ac.jp

DOI: 10.1587/transinf.E95.D.1031

Copyright © 2012 The Institute of Electronics, Information and Communication Engineers

next section, we explain the syntax and semantics of requirements frames, which includes a requirements definition language. In Sect. 3, we briefly introduce spectrum analysis for quality requirements, which is useful for comparing measured quality requirements and recommended quality requirements. We then explain a method to measure a specification for identifying quality requirements and another method to recommend quality requirements to be defined in the specification in Sect. 4. In Sect. 5, we show a case study to demonstrate our methods are useful to confirm the completeness and correctness of a requirements specification. We then review related works in Sect. 6. Finally, we summarize our current results and show the future issues.

## 2. Requirements Frames

### 2.1 Requirements Frame Model

Consider requirements of a library system. To simplify the problem, we focus on the requirements for book retrieval. That is:

> There exist users, cards of retrieval of books, and identifier (ID) number of each book. Users are human-type objects. Cards and ID are data-type objects. Cards are classified into authors-cards that are sorted by author's name in alphabetical order, and title-cards sorted by title. A user can retrieve books with these cards.

A requirements definer first identifies objects (nouns), object types (attributes) in a target system. Secondly he defines operations among objects (verbs) and roles of the operations (cases [9]), and then constructs requirements sentences. The "cases" mean concept about agents, objects, goals of the operations [10]. Thus, a requirement sentence includes nouns and verbs as its components, and there exist roles of objects as relations among the components. A particular functional requirement may be defined with several sentences.

Our requirements model has been developed to easily represent the above structures. It involves three kinds of frames. These are a frame of noun level, a frame of sentence level, and a frame of function level.

#### 2.1.1 Noun Frame

The first frame is the Noun Frame, a frame whose components are nouns and their types. Table 1 shows typical noun types provided to specify file-oriented software requirements. A new noun appearing in a requirements description will be classified into one of these types. According to a domain, an organization and/or a technology, we may extend these types.

#### 2.1.2 Case Frame

The second frame is the Case Frame, a frame whose components are nouns and verbs and cases. We provide typical cases; agent, goal, instrument, key, object, operation and source. We also provide several typical concepts including data flow, control flow, data creation, file manipulation, data comparison, and the structure of data/file/function. There are several verbs to represent one of these concepts. For example, to specify a concept data flow, we can use input, output, print out, display, send and so on. A requirements definer can use any verbs as far as it can be categorized in these concepts provided.

We may also extend the kinds of concepts according to a domain, an organization and/or a technology. In other words, we prepare these concepts to specify requirements of a file-oriented software domain. When a user wants to write requirements of another domain, he may need a verb not categorized into these concepts. In such a case, he can use a new verb if he defines its case structure [10]. Since a newly defined verb, its concept, and its case structure can be registered in the verb dictionary, he can use his own verbs as well as provided verbs.

The typical concepts (11 verb type concepts and 6 adjective type concepts) are shown in Table 2.

The Case Frame defines case structures of these concepts. Each concept has its own case structure. For example, the data flow (DFLOW) concept has object, source, goal, and instrument cases. The object case corresponds to a data which is transferred from the source case to the goal case. So, a noun assigned to the object case should be a data type noun. A noun in the source or goal cases should be either a human or a function type noun. If and only if a human type noun is assigned to source or goal cases, some device type noun should be specified as an instrument case. These are

**Table 1** Noun types of the noun frame.

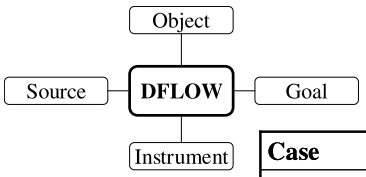
| Noun Type | Meaning |
|---|---|
| human | active and external object |
| function | active and internal object |
| file | passive object of information set |
| data | passive object of a single information |
| info | information in the real world |
| control | passive object for control transition |
| device | passive object of an instrument |
| system | the system to be defined |
| extsystem | external systems related to the system to be defined |
| number | numerical data or information |

**Table 2** Concept of the case frame.

| Concept | Meaning |
|---|---|
| DFLOW | Data flow |
| CFLOW | Control flow |
| ANDSUB | And-tree structure |
| ORSUB | Or-tree structure |
| GEN | Data creation |
| SET | Set the value to data |
| RET | Retrieve a record in a file |
| UPDATE | Update a record in a file |
| DEL | Delete a record in a file |
| INS | Insert a record in a file |
| MANIP | File manipulation |
| EQ, NE, LT, GT, LE, GE | Logical operators |

**Table 3** Analysis of a requirements sentence "A user enters a retrieval command with a terminal."

| Concept | Object | Source | Goal | Instrument |
|---|---|---|---|---|
| DFLOW | A retrieval command | user | *NOT specified* | terminal |




| Case | Noun Type |
|---|---|
| Object | data |
| Source | function, human |
| Goal | function, human |
| Instrument | device |

**Fig. 1** Case structure of data flow (DFLOW).

illustrated in Fig. 1.

The Case Frame enables to detect illegal usages of data and lack of cases. Suppose a requirement sentence, "A user enters a retrieval command with a terminal." Since the objective is "a retrieval command" that is data type noun, "enters" should be categorized into the DFLOW concept. With the Case Frame of the DFLOW, this sentence will be analyzed as shown in Table 3.

In this sentence the goal case noun is not specified. If indispensable case is not specified, previously specified nouns of the same type become candidates of the omitted case. In this way, a requirement sentence is transformed into an internal representation named CRD (Conceptual Requirements Description). CRD is exactly based on the Case Frame.

As shown in Tables 1 and 2, some terms in requirements frame model can be used in the design stage: file, control flow, structure of data/file/functions and so on. According to IEEE 830 standard [1], such terms may be used in a software requirements specification (SRS). The standard also shows a sample SRS format (A.7) including data flow diagrams, which contain such terms. Such a data flow diagram is used to improve the understanding of a complex functional requirement. The standard also mentioned the design of the functional requirement does not have to be partitioned in the same way of such a data flow diagram in SRS. In addition, actual examples [11] frequently use such terms. According to these facts, SRS may contain such terms that can be used in the design phase.

### 2.1.3 Function Frame

The third frame is the Function Frame, a frame whose components are requirements sentences. Let us consider a function, "output a result of retrieval" which consists of subfunctions, file retrieval and data output. These two subfunctions are connected to each other by the fact that the same data type object is used in the goal case of file retrieval and in the agent case of data output. By providing "output a result of retrieval" as an indispensable function, we can point out an error when this function is absent.

In the Function Frame, there are twelve essential functions, such as data processing, external data I/O, file definition, file manipulation, and so on.

These functions are classified into 3 types; 1) essential functions of information systems, 2) essential functions of file system, and 3) essential functions of record operations.

We can check whether a certain file-oriented requirements description satisfies all the properties and detect lack of functions. The consistency of the description is also checked with this frame, such as each data flow should begin as an external input and should end as an external output, a function should receive its sub-function's inputs, and so on [8]. We do not support users' addition of new essential functions, because it is difficult to automatically create a checking program corresponding to user's new essential function.

## 2.2 Requirememts Language: X-JRDL

We have developed a text-base requirements language named X-JRDL [8]. This language is based on the Requirements Frame model. We can analyze requirements specification written with X-JRDL using natural language processing techniques as follows.

In X-JRDL, both a compound sentence that contains two independent clauses joined by a coordinator, such as "for," "and," "but" and so on, and a complex sentence that has an independent clause joined by one or more dependent clauses can be written. A complex sentence always has a subordinator such as "if," "when," "which" and so on. Compound sentences and complex sentences will be divided into simple sentences each of which has just one verb for analysis with Case Frame. These simple sentences are transformed into CRD.

An X-JRDL description is analyzed through three interpreters. Since X-JRDL allows compound sentences and complex sentences, a surface interpreter divides them into a set of simple sentences. Another interpreter, word interpreter, fulfills a case structure consulting with dictionaries. Since a noun is interpreted with its type, the noun dictionary holds a name and its type. A verb (or an adjective) is interpreted with its corresponding concept. In the case of pronoun and omission of nouns, its type will be guessed with the Case Frame. By a sentence interpreter, a simple sentence is transformed into CRD with checking lacks of indispensable cases.

X-JRDL allows using pronouns and omission of nouns. We frequently come across such features in Japanese sen-

**Table 4** Verbs reserved in the dictionary.

| Concept | A part of registered verbs |
|---|---|
| DFLOW, CFLOW | pass, move, transfer, receive, input |
| ANDSUB, ORSUB | subpart, part, construct |
| GEN | generate, produce, make |
| RET | retrieve |
| INS | insert, add |
| UPDATE | update |
| DEL | delete |

tences. The X-JRDL analyzer automatically assigns a concrete word into a pronoun or a lacked case.

Conjunctions are used to write down compound sentences and complex sentence. The analyzer divides such a sentence into a set of simple sentences.

The X-JRDL analyzer has a dictionary of nouns, verbs and adjectives. When a requirements definer uses a word which is not appeared in the dictionary, the analyzer guesses a type of new noun and a concept of new verb and adjective with the Requirements Frame. Table 4 shows registered verbs. Actually these verbs are Japanese verbs, but translated in English for English readers. The analyzer can treat inflection of these verbs.

After all the X-JRDL sentences are analyzed and transformed into CRD, the analyzer checks the whole CRD whether indispensable functions are absent with the Function Frame. Thus, the analyzer improves the correctness and the consistency of the SRS. X-JRDL analyzer was developed with GNU Common Lisp.

## 3. Spectrum Analysis for Quality Requirements

Some quality requirements are usually scattered over a requirements specification. Therefore, it is not easy to confirm whether quality requirements are defined enough. We thus proposed a technique called spectrum analysis for software quality requirements to help us to confirm them [3], [12]. As shown in the name of this technique, it is inspired by spectrum analysis in optics where all waves can consist of several different canonical waves (sine waves). We will explain the idea of this technique by using an example in Fig. 2. A software engineering artifact such as a requirements specification contains several different quality requirements, and the importance of each quality requirement differs from others. We thus regard each quality requirement as a canonical wave, and such an artifact as a kind of a complex wave. By using a spectrum analyzer for quality requirements, we can identify canonical waves in the complex wave, and powers of each wave. This technique enables us to analyze the following issues.

- To validate quality requirements definition in a specification of a system by comparing a spectrum of the specification to one of similar specification. We assume specifications of similar systems have similar spectrum of quality requirements, and specifications of different kinds of systems have different spectra. We have confirmed this assumption through our experiments [3], [12]. Assume similar Web based shopping systems. Such systems have to similarly take care of quality requirements such as security, performance, usability and so on. If a specification of a system has quite different spectrum from specifications of its similar systems, there could be some missing and/or incorrect quality requirements definition in the specification. Of course, some system may intentionally take care of quality requirements differently from its similar systems. For example, a system sometimes gives high priority to usability and makes light of security for some reasons. This technique also contributes to confirming this kind of intention systematically.
- To trace the inheritance of quality requirements through software development phases. Quality requirements definitions should be inherited though requirements, architecture, design, coding and release phases. However, programmers in the coding phase sometimes meet large difficulties to satisfy such quality requirements due to the wrong decision in architecture and/or design phases. For functional requirements, traditional stepwise refinement techniques can be helpful for this kind of traceability, but they cannot be applied to quality requirements. Our spectrum analysis technique can analyze each software engineering artifact such as a requirements specification, its design document and its codes respectively. By comparing a requirements spectrum to another spectrum, we can check this traceability, and find symptoms of bad decision about quality requirements in an intermediate phase. We have confirmed part of this issue through our experiment [4].

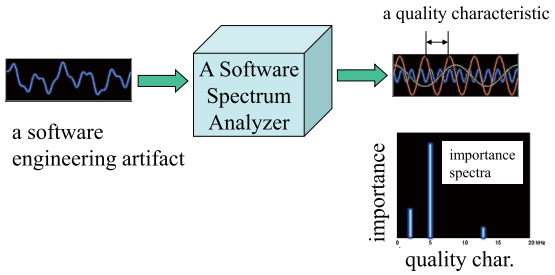



**Fig. 2** Basic idea of spectrum analysis for quality requirements.

As mentioned above, we focus on comparison among spectra of similar systems. Therefore, identifying similar systems is one of the critical issues. We identify such similarity based on application types, but we have to take other features such as running platforms into account for such identification. In [13], such similarity is identified based on the structure of a use case diagram for each system. Because a use case diagram specifies the context of the system as well as externally observable functions of the system, such context and functions are main information for identifying system similarity. According to [14], systems can be categorized based on eight dimensions (dimensions related to state, object, goal and so on), and 13 concrete categories of the object dimension (e.g., resource returning, resource

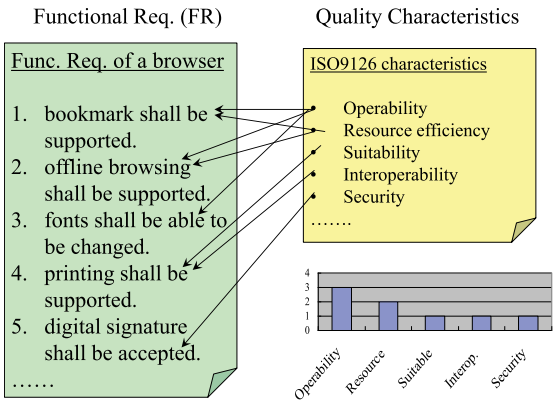



**Fig. 3** Spectrum analysis for quality requirements.

supplying, object sensing, domain simulation and so on) are proposed in the literature. By using such dimensions and concrete categories in [14], we can also identify similarity among systems. Practically, we have already had web sites of software applications[†] and such sites have already categorized a lot of applications. We may choose one of the categories out of the list on such sites according to the system to be analyzed, and use several existing systems in the chosen category. We have also taken critical level, data volume, size and user environment into account for identifying similarity. For example, document readers for PC require different usability and efficiency from those for smart phones. In this case, we have to apply this spectrum analysis to document readers for either PC or smart phones.

By using Fig. 3, we will explain how to implement spectrum analyzer for quality requirements. The inputs of the analyzer are the list of requirements sentences (a requirements list) and a catalog of quality requirements. We regard the list contains a complex wave and the wave consists of several different canonical waves, each of which has different cycle and power. In the figure, quality subcharacteristics for software defined in ISO9126 [15] is used as the catalog of quality requirements. Different cycles of canonical waves specify the differences of quality requirements in our analogy. The output is a spectrum of the requirements list, which is a kind of vector of powers of each quality requirement. In the figure, the spectrum is a five dimensional vector, where each dimension shows the number of requirements in the list qualified by a quality requirement. To derive this output from the inputs, a requirement analyst has to make relationships between each requirement and each quality requirement as shown in the figure. For example in this figure, an analyst decided requirements #1 and #2 are related to resource efficiency.

Because this kind of decision largely depends on the ability of an analyst, we have provided a supporting facility for this decision by using light weight natural language processing technique [16]. However, it is still difficult to perform spectrum analysis for quality requirements accuracy and automatically because of the freedom of natural language representation.

In this technique, the importance of each quality requirement is counted by related requirements sentences. We now explain why we may regard such a count as the extent of the importance. In the section about "ranked for importance and/or stability" of IEEE 830 standard [1], we can find the following sentences.

> Typically, all of the requirements that related to a software product are not equally important. Some requirements may be essential, especially for life-critical applications, while others may be desirable.

We may thus regard the importance of requirements is the degree whether each of them is essential or desirable. The standard also mentions such importance should be specified clearly and explicitly. Therefore, the importance of quality requirements seems to be specified clearly and explicitly as well as the importance of functional ones.

However, the importance of quality requirements depends on the importance of functional ones because "quality requirements describe how well functions are accomplished" [2]. For example, a sentence "usability is more important than security" is meaningless if we do not check whether each functional requirement is qualified by usability and/or security. Therefore, we may regard the importance of a quality requirement depends on the importance of functional requirements qualified by the quality requirement. If most of all functional requirements are qualified by usability, a few of them are qualified by security and all requirements are equally important (i.e., no importance is specified), we have to regard usability is more important than security. In such a case, functional requirements specified by security should be postponed if we do not have enough budgets and/or the time. We thus assume that a count of functional requirements qualified by a quality requirement shows the importance of the quality requirement if all functional requirements are equally important. We have validated this assumption through five published case studies [3] and two unpublished case studies[††]. In the case of an example of Fig. 3, we assume all functional requirements are equally important. The spectrum in the figure then tells us the most important quality requirement is operability in this requirements list, and suitability, interoperability and security have the same importance.

If functional requirements are not equally important, we have to give some weights to each of them according to each importance. The importance of each quality requirement should be then counted based on the weighted functional requirements. How to weight functional requirements based on their importance is one of our future issues, and existing prioritization techniques [17] will contribute to defining the way of such weighting rules. If the importance for quality requirements is explicitly specified, we have to check the consistency between such specified importance

[†]for example, http://www.vector.co.jp/magazine/softnews/, http://sourceforge.net/index.php

[††]These two appeared in a master thesis.

and the importance based on the number of functional requirements qualified by each quality requirement. Inconsistency should be of course resolved if it is found. In our case study in this paper, we regard all functional requirements are equally important because there is no explicit specification about the importance of both functional and quality requirements in their original document [11].

## 4. Methods for Measure and Recommendation

### 4.1 General Idea

We assume quality requirements defined in a functional requirement can be measured based on each part in the requirement. For example, interoperability is defined in a requirement when the requirement is about DFLOW and one of its cases has external system type. We also assume quality requirements required in a functional requirement can be recommended based on parts in the requirement. For example, usability can be required in a requirement if one of its cases has platform type such as OS or hardware.

As mentioned in Sect. 2, case frame in our requirements frame model is suitable for measuring and recommending quality requirements in a requirement. The reasons are as follows. First, each requirement can be represented in a case structure, and a noun (or a term) in each case in the structure should have specific type as shown in Fig. 1. Second, concept may be regarded as one of cases, and its type is one of DFLOW, CFLOW, ANDSUB and so on in Table 2. Third, we do not have to apply additional natural language processing because semantic elements such as concepts, cases and types are already identified in our requirements frame model.

For simplicity, we call the instance of case structure about a specific functional requirement as a conceptual requirements sentence (CRS). For example, Table 5 shows a CRS of a functional requirement in Table 3. Although DFLOW can have additional case goal in general, goal case does not exist in this example due to the missing elements of original sentence.

### 4.2 Method of Measurement

Here we explain how to identify quality requirements in a functional requirement representation, i.e., measuring quality requirements in a functional requirement. We perform the following steps for this purpose.

1. We convert each functional requirement into one or more CRSs. How to convert them is out of scope of this paper because we have already mentioned in our previous works [7], [8]. We also add information of quality requirements to each CRS if the quality requirements qualify the functional requirement. We have to manually identify such quality requirements and functional requirements qualified by them now.
2. We develop if-then rules for measuring each type of quality requirements. The rules are applied to each CRS. If-part of each rule consists of logical conjunctions whether some case has specific types or not. Then-part tells us whether a quality requirement is mentioned in the CRS or not. Then-part also tells us the certainty of its decision. We will show examples below.
3. We apply all rules to each CRS. If more than one rule decides a CRS contains a quality requirement, a rule with the larger/largest certainty is chosen as its result.
4. We may visualize all the results by using our spectrum analysis mentioned in Sect. 3.

We then show the premises and the limitations of this method.

- The input of this method is a requirements document written in natural language, and the document is converted into a set of CRSs in step 1 above. The document should be at least unambiguous and consistent mentioned in IEEE standard 830 [1]. No ambiguity is very important because we have to identify functional requirements qualified by each quality requirement in step 1. We regard all functional requirements are equally important if ranked for importance is not explicitly specified.
- Cases and their types for CRS should be extended when rules for quality requirements cannot be specified based on current cases and their types. As mentioned in Sect. 2, cases and types in Tables 1, 2 and 3 are just typical ones, and we may extend them.
- Because several quality requirements are similar to each other, rules sometimes decide whether some of similar quality requirements are mentioned in a CRS. However, such rules cannot decide whether a quality requirement is exactly mentioned in the CRS. We usually use about 30 quality sub-characteristics in ISO9126 [15] as a catalog of quality requirements, and each sub-characteristic belongs to one of six (main-)characteristics. Our rules sometimes cannot decide a quality requirement (i.e., sub-characteristics) is mentioned in a CRS, but only decide a set of quality requirements (e.g., a main-characteristic) is mentioned in the CRS.

By using examples of three CRSs in Fig. 4 and two rules in Fig. 5, we will explain above steps in detail especially how to represent and to apply the rules. In step 1, we obtain CRS such as Fig. 4. In our method, we only focus on types of each case. As shown in Fig. 5, each rule consists of three parts. In the first part, a quality requirement type measured by a rule is specified. Currently, we

**Table 5** An example of a requirements frame about the sentence "A user enters a retrieval command with a terminal." in Table 3.

| Case | Type |
|---|---|
| Concept | DFLOW |
| Object | data |
| Source | human |
| Instrument | device |

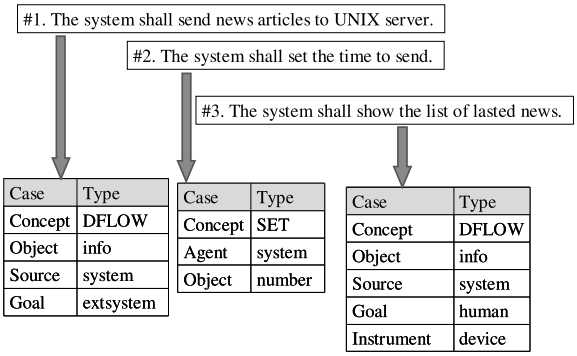



**Fig. 4** Examples of original requirements sentences and CRSs.

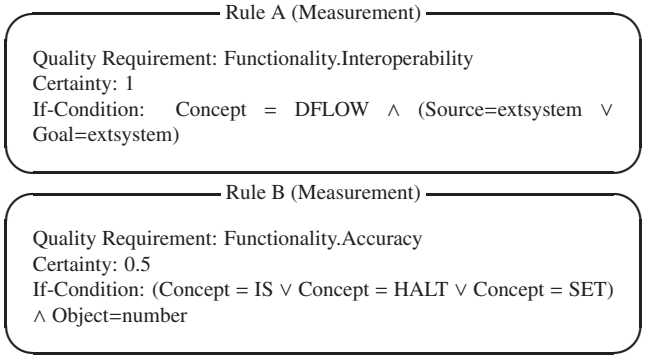



**Fig. 5** Examples of rules of measurement.

**Table 6** the results of applying rules in Fig. 5 to CRSs in Fig. 4 ("-" means not applicable).

| Frame ID | Rule A (Interoperability) | Rule B (Accuracy) |
|---|---|---|
| #1 | 1 | - |
| #2 | - | 0.5 |
| #3 | - | - |

use quality sub-characteristics in ISO9126 [15] as a catalog of quality requirements. However, any kinds of quality requirements may be specified if you can specify its condition. The second part shows the level of certainty of the result of a rule. Certainty 1 means the result is completely true, and 0.5 means the result is in the middle. Currently, we only use the values of 1 and 0.5. If the certainty is zero, you should not write the rule of course. We do not introduce negative values of certainty now, which can specify counterexamples of our measurement. The last part in a rule shows the condition whether a CRS frame meets the rule or not. As shown in Fig. 5, a condition consists of logical conjunctions of types in a CRS.

In Table 6, we show the results of applying these two rules in Fig. 5 to each of three CRSs in Fig. 4. In Fig. 5, Rule A specifies that software system receiving the same data from an external system or passing the same data to an external system should be interoperable. Rule B specifies that a requirement related to numerical data definition or halt condition with numerical data may be related to accuracy. As shown in the results, at least these rules meet our intuitive

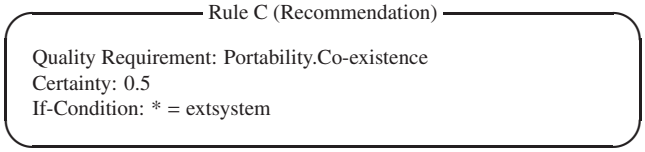



**Fig. 6** Example of a rule of recommendation.

decision which kinds of quality requirements are mentioned in a CRS.

### 4.3 Method of Recommendation

In the case of recommending which quality requirements should be defined in a CRS, the steps and the syntax of rules are exactly the same as the measured method above. The only difference is the contents of the rules. Figure 6 shows an example of a rule of recommendation. Note that "*" in the rule means any cases. We use several syntax sugars like this to represent rules efficiently, and the other sugars will be explained in appendixes. The rule thus tells us that a CRS with one or more cases with the type "extsystem" may be related to co-existence quality requirements. In other words, the rule specifies that requirements related to external system should be recommended co-existence.

### 4.4 Comparison of the Results of Both Measurement and Recommendation

If measured quality requirements in a CRS are the same as recommended ones in the CRS, we may regard the CRS is complete and correct with respect to the quality requirements. However, measured and recommended quality requirements are different in general. We thus summarizes how to suspect incompleteness and incorrectness in a CRS in Table 7. If results by rules of measurement are the same as the results by rules of recommendation, we do not have to suspect incompleteness and incorrectness in a CRS. If rules of measurement measure some quality characteristics but rules of recommendation do not recommend them, we may suspect there is some incorrectness about the characteristics in the CRS. If rules of recommendation recommend some quality characteristics but rules of measurement do not measure them, we may suspect there is some incompleteness about the characteristics in the CRS.

As mentioned in Sect. 4.2, cases and their types for specifying CRS are gradually extended. Therefore, we cannot guarantee rules for all quality requirements can be specified before extending such cases and their types. To improve the recall for detecting incorrectness and incompleteness, i.e., to minimize the oversight of incorrect or incomplete CRS, we introduce the following heuristics in addition to Table 7. If a rule of measurement decides a CRS contains a quality requirement but any rules of recommendation about the quality requirement cannot be specified, we suspect the CRS is incorrect with respect to the quality requirement. If a rule of recommendation decides a CRS contains a quality

**Table 7** Suspecting incompleteness and incorrectness in a CRS.

| each CRS | | rules of recommendation | |
|---|---|---|---|
| | | Yes | No |
| rules of measurement | Yes | OK | Incorrect? |
| | No | Missing? | OK |

requirement but any rules of measurement about the quality requirement cannot be specified, we suspect the CRS is incomplete with respect to the quality requirement.

Visualization by our spectrum analysis enables analysts to identify trends of incompleteness and incorrectness in a specification even if the specification is very large.

## 5. Case Study

As mentioned in introduction, we would like to show the usefulness of our methods through this case study. We set up the following concrete research questions to show its usefulness.

- By using our method of measurement, quality requirements stated in a specification can be measured correctly and completely.
- By comparing measured quality requirements to recommended ones, we can identify missing quality requirements to be actually defined, i.e. we can make a specification more completely with respect to quality requirements definition.
- By comparing measured quality requirements to recommended ones, we can identify incorrect or inappropriate quality requirements, i.e. we can make a specification more correctly with respect to quality requirements definition.

### 5.1 Set up and Steps

We use a requirements specification for a Web based system for browsing latest news summary [11] (simply call it "news system" in this paper). The system is a typical Web based information retrieval system. The specification is one of the samples for requirements specifications provided by Japanese government to standardize the format of specifications. Although the specification is a sample, it is not complete and correct with respect to quality requirements definition. The original specification consists of 13 pages of sentences, figures and tables in A4 size paper. We picked up 45 CRSs as an input of this case study mainly based on the sentences in the specification.

We use sub-characteristics in ISO9126 [15] as a catalog of quality requirements. There are 6 characteristics and 27 sub-characteristics in ISO9126.

Based on the materials above, we performed the following steps.

1. Based on our experiences about Web based systems, we develop rules of measurememnt and rules of recommendation respectively.
2. We apply two kinds of rules above to 45 CRSs.
3. We visualize the results of each kind of rules to identify the following things by using our spectrum analysis.
   - Which kinds of quality requirements are important in the specification? (measured value)
   - Which kinds should be important in it? (recommended value)
4. We focus on quality requirements where recommended value is larger than measured value. We suspect there are some CRSs that are missing quality requirements.
5. We focus on quality requirements where measured value is larger than recommended value. We suspect there are some CRSs that contain incorrect quality requirements.

### 5.2 Results

According to step 1 above, we developed 15 rules of measurement and 14 rules of recommendation respectively. All these rules are shown in appendixes of this paper. Because one rule sometimes relates to more than one quality requirements with the help of syntax sugars for rules as shown in appendixes, the number of rules is not equal to the number of quality requirements. Therefore, the numbers of rules are not important but which quality requirements are measured and recommended is important. Table 8 shows such things. As shown in the table, most quality requirements except for both maintainability and portability can be measured or recommended. Note that we do not care the differences of certainty of each rule (1 or 0.5) here.

According to steps 2 and 3, we applied two kinds of rules to 45 CRSs and visualized the results. Figure 7 shows the visualized results using our spectrum analysis. Note that the importance is normalized based on the number of CRSs, i.e. 45. For example in this figure, our recommended rules decide 40% of CRSs should be mentioned to security, but our measured rules identify none of them are mentioned to security. On the other hand, our rules decide about 9% should be mentioned to operability, and our rules identify about 15% are mentioned to operability.

### 5.3 Discussion

We discuss the results of this case study mainly according to the research questions at the beginning of this section. Before discussing the questions, we discuss whether rules can measure and recommend quality requirements in a CRS. As shown in the Table 8, we cannot define rules about changeability, stability, testability and installability. Most of these quality requirements are about internal quality, thus they are rarely written in a requirements specification. This is one of the limitations of our methods.

We then recall our research questions. First, we confirmed whether quality requirements in 45 CRSs could be actually measured by our rules of measurement. We

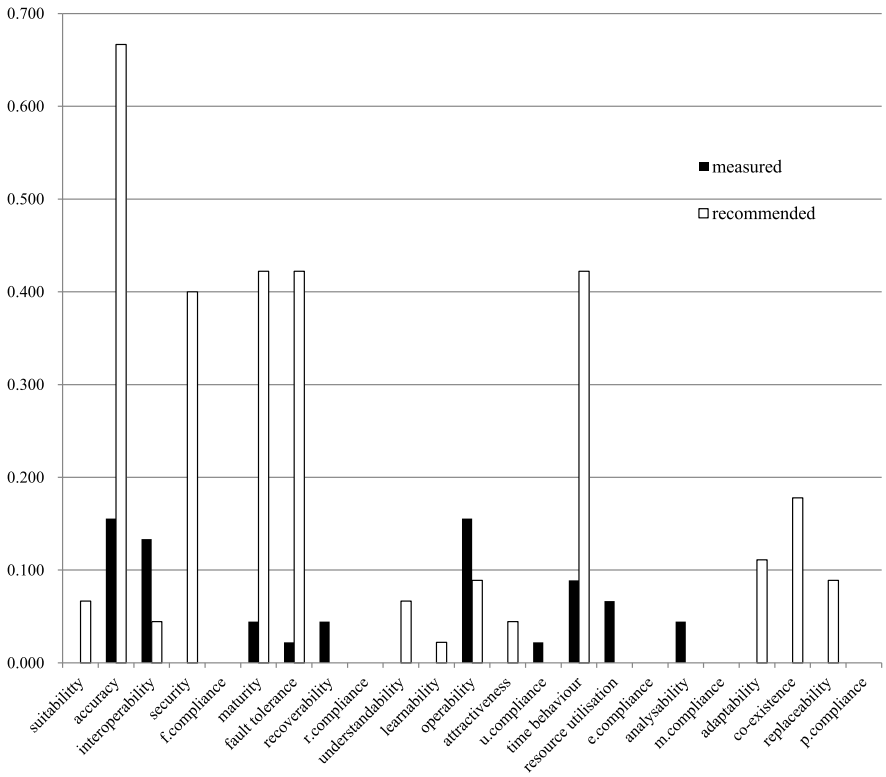



**Fig. 7** Measured spectrum and recommended spectrum.

**Table 8** Quality requirements measured and recommended. Sub characteristics in ISO9126 are used for categorizing quality requirements. The IDs of rules in appendixes such as M1 or R2 are filled in each cell if a requirement can be measured and/or recommended. Otherwise, the cell is empty.

| char. | sub char. | measured | recommended |
|---|---|---|---|
| functionality | suitability | M12 | R1 |
| | accuracy | M4 M8 M13 | R2, R3 R4 |
| | interoperability | M5 M6 | R5 |
| | security | | R7 R8 |
| | compliance | M3 | |
| reliability | maturity | M11 M14 | R9 |
| | fault tolerance | M11 | R10 |
| | recoverability | M11 M2 M7 | |
| | compliance | M11 M3 | |
| usability | understandability | | R2 R3 |
| | learnability | | R2 R3 |
| | operability | M9 | R2 R3 |
| | attractiveness | | R2 R3 |
| | compliance | M3 | |
| efficiency | time behaviour | M4 M1 | R11 |
| | resource utilisation | M4 M15 | |
| | compliance | M4 M3 | |
| maintainability | analysability | M10 | |
| | changeability | | |
| | stability | | |
| | testability | | |
| | compliance | M3 | |
| portability | adaptability | | R14 |
| | installability | | |
| | co-existence | | R12, R13 |
| | replaceability | | R6 |
| | compliance | M3 | |

checked the contents of requirements sentences and their original sentences step by step and confirmed our rules can measure quality requirements stated in the specification. We may thus regard our measure method is useful for measuring quality requirements in a specification. Concrete examples of measured quality requirements in this case study are as follows.

- A CRS (SET, Agent=system, Object=number)[†] is derived from a sentence "The system shall set the timestamp of retrieved data.", and a rule M13 in Appendix A decides the CRS is related to accuracy of functionality.
- A CRS (DFLOW, Object=data, Source=system, Goal=extsystem) is derived from a sentence "Log files shall be sent to an external UNIX server.", and a rule M5 decides the CRS is related to interoperability of functionality.
- A CRS (RESPOND, Agent=system, Object=number) is derived from a sentence "The system shall respond within 5 seconds.", and a rule M1 decides the CRS is related to time behavior of efficiency.

Second, we explored missing quality requirements by comparing measured and recommended spectra. As shown in Fig. 7, recommended value is larger than measured value especially in accuracy, security, maturity, fault tolerance and time behavior. We thus suspected these quality requirements could be missing in the CRSs, and investigated the contents of related CRSs. We then found the following kinds of miss-

[†]We abbreviated a case structure such as Fig. 1 in this way.

ing requirements.

- Accuracy: There are several functions about retrieving or checking data. Although accuracy of such functions should be specified, some of them were not.
- Security: Some functions require interaction between external systems, and such interaction usually requires security concern. However, some of them did not take care of security. In addition, some functions related to logging are simply stated, and such logging can contribute to security such as forensics. However, there were no explicit explanations about the logging functions.
- Fault tolerance and maturity: Some crucial functions such as updating data should take care of their faults and recovery, but some of them never mentioned such issues.
- Time behavior: Some functions are sensitive to its response, thus time behavior seems to be important. However, some of them did not care of it.

We may thus our measured and recommended rules contribute to finding missing quality requirements.

Finally, we explored incorrect requirements in the same way. As shown in Fig. 7, measured value is larger than recommended value especially in interoperability, recoverability, operability and analyzability. There were a few incorrect quality requirements in fact, but measured results with certainty 0.5 seemed to be incorrect or ambiguous. For example, our rule decided a CRS about the average access number per a day is related to resource utilization. However, it could be related to time behavior because such number largely depends on the performance of the system as well as its resource.

### 5.4 Threats to Validity

Because 45 CRSs were developed beforehand in this case study, there is no bad or good effects to the CRSs by this case study. Our rules of measurement and rules of recommendation are developed only suitable for Web based systems. In addition, we applied our rules only to one specification. Therefore, the rules are not general enough. Both spectrum derived from measured rules and spectrum derived from recommended rules seem to be useful to infer completeness and correctness of quality requirements. However, we need some other metrics against right quality requirements definition. We have to make more case studies to eliminate threats to conclusion validity by using statistical tests.

As mentioned above, our rules are specific for Web based systems, and there were developed by us (experts of requirements engineering and Web based systems) in this case study. Because the success of our methods largely depends on the quality of rules, how to maintain rules in high quality is one of the crucial problems. To overcome this problem, we perform PDCA (Plan, Do, Check and Action) cycles for our rule's improvement. In addition, we will categorize rules into the following three types to facilitate its reuse.

- General and Domain-independent rules
- Domain-specific rules
- System or organization specific rules

First two types of rules can be easily reused.

Another problem is an application of our rules. Our rules were designed to apply them without subjective decisions. However, some of them required subjective decision. For example, we had to examine terms in case frames whether a specified number shows time or size when we applied rules for time behavior and resource efficiency. To eliminate such decision, we will use the facility of limited representations in each case frames. Basically, we may freely fill words or terms in each frame in a requirements frame. However, we may limit variations of such words or terms in advance. We can have a mapping between such limited variations and rules to eliminate subjective decision above. Another possibility is to extend our type systems, e.g. introduce time and resource types that are subclass of number type.

Finally, we mention the possibility to generate rules automatically by using machine learning techniques. Even if we can improve our rules step by step through PDCA cycles, it takes a lot of efforts to do that. Because our rules depend on types in each case including concept (concept may be simply regarded as a kind of case frames in our rules), we may generate rules based on negative and positive instances of their application. We preliminary performed this idea to one of quality requirements and several CRSs by using WEKA data mining tool [18], and we could obtain rules with about 95% precision.

## 6. Related Works

There are several studies how to define each quality requirement. In ISO25021 [19], concrete examples how to measure quality requirements are shown, and these examples help analysts to make quality requirements measurable. Donald Firesmith gives some format to specify quality requirements rigorously [20]. In ATAM (Architecture Tradeoff Analysis Method) [21], [22], a template for quality requirements called "quality attribute scenario" is provided to evaluate the validity of architectural decision. Such template may be used to support stakeholders writing quality requirements. In an article by Ozkayad et al. [23], an empirical data of the most common quality attributes was shown based on the ATAM. The idea to analyze this kind of data is similar to quality spectrum analysis [24]. However, quality spectrum analysis cannot be used for architecture evaluation because such evaluation is not its purpose. UML is a most popular semi-formal notation for software development now, and there are some challenges to introduce quality requirements into it [25], [26]. Using ontology, dictionary and/or thesaurus [27] is one of the useful ways to improve the quality of requirements with respect to semantic aspect. Most of them do not focus on comparison between as-is (measured)

and to-be (recommended) quality requirements like our proposal.

## 7. Conclusion

In this paper, we proposed methods to improve quality requirements definition in a requirements specification. We first measure a specification whether what kinds of quality requirements are defined. We then recommend quality requirements that should be defined in the specification. We use a semi-formal representation for requirements based on our requirements frames model because the representation enables us to show semantic elements in a specification explicitly. To measure and recommend quality requirements, we use if-then rules based on the semi-formal representation. We applied our methods to a specification and confirmed our methods could measure quality requirements written in the specification. We also confirmed our methods contribute to improving the completeness and correctness of quality requirements definition, i.e., finding missing and incorrect quality requirements.

In Sect. 3, we explain the importance of each quality requirement is counted by related requirements sentences, and we mentioned we have to take importance of such sentences at the end of the section. We want to improve how to identify such importance by using existing prioritization techniques [17] in the future. As mentioned in the discussion of case study, maintaining rules in high quality is crucial in our methods. To overcome this issue, we want to encourage requirements analysts to reuse rules and improve them through PDCA cycle. Categorizing rules into domain-independent, domain-specific and system-specific ones is one of the useful aspects for this improvement. We also explore the possibility to generate rules automatically by using machine learning techniques. Currently, we perform our methods by using general spread sheets. We want to develop a supporting tool specific for our methods to perform our methods more efficiently and to make rules and results reusable.

### References

[1] "IEEE Recommended Practice for Software Requirements Specifications," 1998. IEEE Std. 830-1998.
[2] J.D. Blaine and J. Cleland-Huang, "Software quality requirements: How to balance competing priorities," IEEE Softw., vol.25, no.2, pp.22–24, March/April 2008.
[3] H. Kaiya, M. Tanigawa, S. Suzuki, T. Sato, A. Osada, and K. Kaijiri, "Improving reliability of spectrum analysis for software quality requirements using TCM," IEICE Trans. Inf. & Syst., vol.E93-D, no.4, pp.702–712, April 2010.
[4] H. Kaiya, K. Amemiya, Y. Shimizu, and K. Kaijiri, "Towards an integrated support for traceability of quality requirements using software spectrum analysis," Proc. 5th International Conference on Software and Data Technologies (ICSOFT), pp.187–194, Athens, Greece, July 2010.
[5] M.D. Cin, "Structured language for specifications of quantitative requirements," IEEE 5th International Symposium on High Assurance Systems Engineering (HASE), pp.221–227, 2000.
[6] R. Beg, Q. Abbas, and A. Joshi, "A method to deal with the type of lexical ambiguity in a software requirement specification document," First International Conference on Emerging Trends in Engineering and Technology (ICETET'08), pp.1212–1215, 2008.
[7] Y. Matsuo, K. Ogasawara, and A. Ohnishi, "Automatic transformation of organization of software requirements specifications," RCIS, pp.269–278, 2010.
[8] A. Ohnishi, "Software requirements specification database based on requirements frame model," ICRE, pp.221–228, 1996.
[9] C.J. Fillmore, The Case for Case, Universals in Linguistic Theory, Rinehart and Winston Publishing, 1968.
[10] R. Shank, "Representation and Understanding of Text," Machine Intelligence, vol.8, pp.575–607, 1977.
[11] http://www.meti.go.jp/feedback/data/i30728aj.html, "Guideline for writing requirements specifications (draft)," 2003. last accessed Dec. 2010. copy of its contents http://kaiya.cs.shinshu-u.ac.jp/i30728aj/
[12] H. Kaiya, M. Tanigawa, S. Suzuki, T. Sato, and K. Kaijiri, "Spectrum Analysis for Quality Requirements by Using a Term-Characteristics," 21st International Conference Advanced Information Systems Engineering (CAiSE 2009), Amsterdam, The Netherlands, pp.546–560, June 2009. LNCS 5565.
[13] H. Kaiya, A. Osada, and K. Kaijiri, "Identifying stakeholders and their preferences about NFR by comparing use case diagrams of several existing systems," IEICE Trans. Inf. & Syst., vol.E91-D, no.4, pp.897–906, April 2008.
[14] N.A.M. Maiden and M. Hare, "Problem domain categories in requirements engineering," Int. J. Hum.-Comput. Stud., vol.49, no.3, pp.281–304, 1998.
[15] International Standard ISO/IEC 9126-1, "Software engineering - Product quality - Part 1: Quality model," 2001.
[16] S. Suzuki, T. Sato, M. Tanigawa, A. Osada, H. Kaiya, and K. Kaijiri, "A Systematic Method for Generating Quality Requirements Spectrum," Proc. 24th Annual ACM Symposium on Applied Computing 2009, vol.1 of 3, pp.399–400, Honolulu, Hawaii, March 2009. Track on Requirements Engineering.
[17] A.A. Eds., P. Berander, and A. Andrews, Engineering And Managing Software Requirements, ch. Requirements Prioritization, pp.69–94, Springer, 2005.
[18] I.H. Witten, Data Mining, Second Edition: Practical Machine Learning Tools and Techniques, second ed., The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, 2005.
[19] International Standard ISO/IEC 25021, "Software engineering - Software product quality requirements and evaluation (SQuaRE) - Quality measure elements," Oct. 2007.
[20] D. Firesmith, "Quality requirements checklist," J. Object Technology, vol.4, no.9, pp.31–38, Nov.-Dec. 2005.
[21] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), pp.68–78, 1998.
[22] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed., Addison-Wesley, 2003.
[23] I. Ozkayad, L. Bass, R.S. Sangwan, and R.L. Nord, "Making practical use of quality attribute information," IEEE Softw., vol.25, no.2, pp.25–33, March/April 2008.
[24] H. Kaiya, T. Sato, A. Osada, N. Kitazawa, and K. Kaijiri, "Toward quality requirements analysis based on domain specific quality spectrum," Proc. 23rd Annual ACM Symposium on Applied Computing 2008, Volume 1 of 3, pp.596–601, Fortaleza, Ceara, Brazil, March 2008. Track on Requirements Engineering.
[25] Y. Zhang, Y. Liu, L. Zhang, Z. Ma, and H. Mei, "Modeling and Checking for Non-Functional Attributes in Extended UML Class Diagram," Annual IEEE International Computer Software and Applications Conference (COMPSAC2008), pp.100–107, 2008.
[26] Z.M. Yi Liu and W. Shao, "Integrating non-functional requirement modeling into model driven development method," 17th Asia-Pacific Software Engineering Conference (APSEC 2010), pp.98–107, Dec. 2010.

[27] D.V. Dzung and A. Ohnishi, "Improvement of quality of software requirements with requirements ontology," QSIC, pp.284–289, 2009.

## Appendix A: Rules of measurement in Sect. 5

Following 15 rules were used in the case study in Sect. 5.

**Rule M1**

Quality Requirement: Efficiency.Time behaviour
Certainty: 1
If-Condition: Concept = RESPOND

**Rule M2**

Quality Requirement: Reliability.Recoverability
Certainty: 1
If-Condition: Concept = RESTORE

**Rule M3**

Quality Requirement: *.Compliance
Certainty: 1
If-Condition: Concept = OBEY

Several quality requirements are sometimes similar to each other. For example, each quality characteristic has its sub characteristic about "compliance". In addition, sub characteristics in a characteristic are inherently similar to each other. For example, "understandability", "learnability", "operability" and "attractiveness" are all about the quality about how to use a system. It is thus sometimes difficult to specify only one quality characteristic stated in a CRS based on its cases and types. We use a notation (a syntax sugar) like "*.Compliance" or "Usability.*" to represent a set of similar quality sub-characteristics. An analyst has to choose one or more quality sub-characteristics that are really stated in a CRS if the CRS satisfies the If-Condition part of a rule with the notation. For example, an analyst decides that Usability.Compliance is specified in a CRS with a concept "OBEY" based on a rule M3 above.

Note that we exclude a sub-characteristic "compliance" when we specify all sub-characteristics in a characteristic by using "*". For example, we exclude "Efficiency.compliance" when we use "Efficiency.*".

**Rule M4**

Quality Requirement: Functionality.Accuracy ∧ Efficiency.*
Certainty: 0.5
If-Condition: (Concept = IS ∨ Concept = HALT ∨ Concept = SET) ∧ (Object = number)

A conjunction of quality requirements may be put in a field of "Quality Requirement" when each requirement has the same certainty and the same condition. In a rule M4, we may represent two rules for "Functionality.Accuracy" and "Efficiency.*" respectively, but we represent one rule for simplicity.

**Rule M5**

Quality Requirement: Functionality.Interoperability
Certainty: 1
If-Condition: Concept = DFLOW ∧ (Source = extsystem ∨ Goal = extsystem)

**Rule M6**

Quality Requirement: Functionality.Interoperability
Certainty: 0.5
If-Condition: Concept = COMM ∧ Agent = extsystem

**Rule M7**

Quality Requirement: Reliability.Recoverability
Certainty: 1
If-Condition: Concept = RETURN

**Rule M8**

Quality Requirement: Functionality.Accuracy
Certainty: 1
If-Condition: Concept = CHECK

**Rule M9**

Quality Requirement: Usability.Operability
Certainty: 1
If-Condition: Concept = USE ∧ Agent = human

**Rule M10**

Quality Requirement: Maintainability.Analysability
Certainty: 0.5
If-Condition: Concept = DFLOW ∧ Goal = human

**Rule M11**

Quality Requirement: Reliability.*
Certainty: 1
If-Condition: Concept = HALT ∧ Object = number

**Rule M12**

Quality Requirement: Functionality.Suitabilitty
Certainty: 0.5
If-Condition: Concept = ANDSUB ∧ ((Agent = func ∧ Object = func) ∨ (Agent = data ∧ Object = data))

**Rule M13**

Quality Requirement: Functionality.Accuracy
Certainty: 1
If-Condition: Concept ≠ OBEY ∧ Concpet ≠ HAVE ∧ Concept ≠ SET ∧ Agent = system ∧ Object = number

**Rule M14**

Quality Requirement: Reliability.Maturity
Certainty: 0.5
If-Condition: Concept = RETURN

Rule M15

Quality Requirement: Efficiency.Resource utilisation
Certainty: 0.5
If-Condition: Concept = USE ∧ * = number

## Appendix B: Rules of recommendation in Sect. 5

Following 14 rules were used in the case study in Sect. 5.

Rule R1

Quality Requirement: Functionality.Suitabilitty
Certainty: 0.5
If-Condition: Concept = ANDSUB ∧ ((Agent = info ∧ Object = info) ∨ (Agent = human ∧ Object = human) ∨ (Agent = device ∧ Object = device))

Rule R2

Quality Requirement: Functionality.Accuracy ∧ Usability.*
Certainty: 1
If-Condition: Concept = DFLOW ∧ (Source = human ∨ Goal = human)

Rule R3

Quality Requirement: Functionality.Accuracy ∧ Usability.*
Certainty: 1
If-Condition: Agent = human

Rule R4

Quality Requirement: Functionality.Accuracy
Certainty: 1
If-Condition: Concept ≠ OBEY ∧ Concept ≠ HAVE ∧ Concept ≠ SET ∧ Agent = system

Rule R5

Quality Requirement: Functionality.Interoperability
Certainty: 1
If-Condition: Concept = RESPOND ∧ Agent = system ∧ Object ≠ human

Rule R6

Quality Requirement: Portability.Replaceability
Certainty: 0.5
If-Condition: Object = extsystem

Rule R7

Quality Requirement: Functionality.Security
Certainty: 0.5
If-Condition: * = extsystem ∧ (* = data ∨ * = info)

Rule R8

Quality Requirement: Functionality.Security
Certainty: 0.5
If-Condition: Concept = DEL ∨ Concept = GEN ∨ Concept = LINK ∨ Concept = MANIP ∨ Concept = RERUN ∨ Concept = RESTORE ∨ Concept = SET ∨ Concept = UPDATE

Rule R9

Quality Requirement: Reliability.Maturity
Certainty: 0.5
If-Condition: Concept ≠ OBEY ∧ Concept ≠ HAVE ∧ Concept ≠ SET ∧ Agent = system

Rule R10

Quality Requirement: Reliability.Fault tolerance
Certainty: 0.5
If-Condition: Concept ≠ OBEY ∧ Concept ≠ HAVE ∧ Concept ≠ SET ∧ Agent = system

Rule R11

Quality Requirement: Efficiency.Time behaviour
Certainty: 0.5
If-Condition: Concept ≠ OBEY ∧ Concept ≠ HAVE ∧ Concept ≠ SET ∧ Agent = system

Rule R12

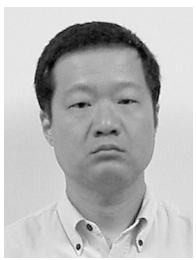
Quality Requirement: Portability.Co-existence
Certainty: 0.5
If-Condition: * = extsystem

Rule R13

Quality Requirement: Portability.Co-existence
Certainty: 1
If-Condition: * = platform

Rule R14

Quality Requirement: Portability.Adaptability
Certainty: 0.5
If-Condition: * = proto



**Haruhiko Kaiya** is an associate professor of Software Engineering in Shinshu University, Japan. He is also a visiting associate professor in National Institute of Informatics (NII), Japan. http://www.cs.shinshu-u.ac.jp/~kaiya/

**Atsushi Ohnishi** is a professor at Department of Computer Science in Ritsumeikan University, Japan.