Predicting Quality Requirements Necessary for a Functional Requirement Based on Machine Learning

Ken Tanaka
Department of Information Sciences
Kanagawa University
Kanagawa, 259-1293, Japan
Email: ktanaka@info.kanagawa-u.ac.jp

Haruhiko Kaiya
Department of Computer Science
Shinshu University
Nagano, 380-8553, Japan
Email: kaiya@shinshu-u.ac.jp

Atsushi Ohnishi
Department of Computer Science
Ritsumeikan University
Shiga, 525-8577, Japan
Email: ohnishi@cs.ritsumei.ac.jp

***Abstract*—In the early stage of the software development, quality requirements should be explicitly specified as well as functional requirements. Software architecture and/or design decision should be largely reconsidered if some quality requirement is overlooked in the early stage. We thus propose a technique for predicting quality requirements necessary for each functional requirement. A functional requirement is represented with a semi-formal language called eXtended Japanese Requirements Description Language (X-JRDL), which is based on the case grammar. In our previous work, the results of the prediction largely depended on human such as domain experts and requirements analysts because prediction rules were manually written by them. We thus introduce machine learning to avoid this problem. To predict quality requirements necessary for any kinds of functional requirements, training data should be appropriately chosen. We choose the training data so that we can predict necessary quality requirements for all types of functional requirements. Since semantically impartial data are suitable for such training data and one of the cases called concept is semantically dominant in an X-JRDL sentence, we choose the training data set in which any of the concepts evenly occurs. Through the experiments, we confirm our technique works well for predicting necessary quality requirements.**

Keywords*-*requirements analysis; quality requirements; machine learning; case grammar.

I. INTRODUCTION

In order to write a requirements specification of high quality, we must take the following characteristics into account; correctness, consistency, unambiguity, completeness, rank of importance, stability, verifiability and traceability [1]. For functional requirements, these characteristics are taken into account well, but taking them into account is still a research challenge in non-functional or quality requirements. Quality requirements specify how well functions are accomplished [2], and they are very important. Some systems such as the online computer aided instruction (CAI) systems or the 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.

To resolve these problems, detecting quality requirements necessary for each functional requirement is crucial because such detection is a basis of discussing stakeholders' understanding, their trade-offs and tacking. We have already proposed a technique for detecting such quality requirements [3]. The technique uses a semi-formal notation for a functional requirement called eXtended Japanese Requirements Description Language (X-JRDL) [4] because the notation explicitly represents the semantic structure of a functional requirement and the structure directly gives influences on quality requirements necessary for the functional requirement. The rules for the detection thus can be written in the if-then rules. Although the results of applying the technique were useful for defining quality requirements [3], it took a lot of effort for preparing the rules for detection. In addition, the quality of rules largely depended on the expertise of people (normally domain experts and requirements analysts) who wrote the rules.

The main contribution of the new technique proposed in this paper is to avoid these problems. In our new technique, a machine learning technology is used to detect the quality requirements necessary for each functional requirement. By using a machine learning technology, rules for detection are automatically generated based on the existing results of detection (we called such results training data). Necessary quality requirements are thus automatically detected based on the rules. The detection results become more accurate than ever when the appropriate training data increase. We still use the semi-formal notation for a functional requirement X-JRDL because the semantic information is explicitly represented in an X-JRDL sentence, and such information is convenient for machine learning. In addition, converting a natural language sentence to a sentence in X-JRDL is already studied [5]. If a sentence is characterized in more

 ISBN: 978-1-61208-230-1

than hundreds components, it is important to choose limited number of components for the efficient machine learning. Because an X-JRDL sentence (called a requirements frame) consists of less than ten components (cases), we do not have to do it. It is rather important to choose appropriate training data because the appropriate choice of training data enables us to detect the quality requirements necessary for all types of functional requirements. In general, we choose the training data so that the data is evenly chosen with respect to the cases in a requirements frame. Because a requirements frame is regarded as a vector of cases, we can use the cosine similarity to choose such training data. However, we have to focus on specific cases to choose such training data if such cases are more dominant than others. Through the experiments, we found one of the cases called concept is more dominant than other cases. In addition, the training data in which any of the concepts evenly occurs enabled us to predict necessary quality requirements successfully.

The rest of this paper is organized as follows. In the next section, we briefly introduce our previous rule-based technique for predicting quality requirements necessary for each functional requirement. Because the technique requires rules written by experts, it is not easy to use it in practice. In Section III, we introduce our new technique for such prediction using machine learning. In our technique, training data are carefully chosen for better prediction. We made an experiment to evaluate our proposed technique. The experiment, the results and its discussion are also reported. We then review existing researches about both the quality requirements analysis and the application of machine learning to the software engineering field. We, finally, summarize our current results and show future issues.

II. Rule-based Technique for predicting Quality Requirements

In this section, we explain our previous rule-based technique [6] [3] for predicting quality requirements necessary for each functional requirement because the inputs and the outputs of the technique are the same as those of our new technique presented in the next section.

A. Overview of the rule-based technique

The goal of our rule-based technique is to predict quality requirements types such as usability, reliability and accuracy for each functional requirement. The steps of the technique for predicting quality requirements are as follows.

1) We have to prepare the rules for each problem domain such as web based information systems, drawing software and so on. Each rule decides whether specific quality requirements types are necessary for a functional requirement. An example of a rule is shown in Figure 2.

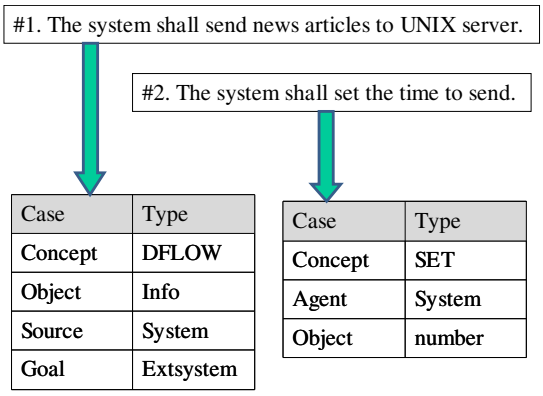

Figure 1. Examples of converting a requirement sentence to a requirements frame.

2) Requirements specifications are usually written in natural language such as English. We have to convert each sentence in a specification into a requirements frame, which is a sentence represented in a semi-formal language called X-JRDL [4] based on the case grammar [7].
3) For each requirements frame, all rules are applied and candidates of necessary quality requirements types are detected. Examples of quality requirements types are "usability", "accuracy", "time behavior" and so on. We can use the quality sub-characteristics in ISO9126 [8] and/or NFR framework [9] as the catalog of quality requirements types.
4) Based on the results of the detection, requirements analysts adds quality requirements descriptions (usually represented as adverbs) each functional requirement. Examples of quality requirements descriptions are "without specific training for its operation", "more than 80% correct results for its query", "within 3 seconds for its response" and so on.

B. Requirements Frames for Functional Requirements

Semantics of a functional requirement largely gives influences on the decision which types of quality requirements are necessary for the functional requirement. We thus convert a functional requirements sentence (usually written in natural language) into a requirements frame. A requirements frame is based on the case grammar [7], and consists of several cases and its types. We thus represent a requirements frame as a tabular form. The examples of the requirements frames represented in a tabular form are shown at the bottom of the Figure 1.

The mandatory case is called "Concept" in a requirements frame, and it corresponds to the verb in an original requirements sentence. We have prepared a list of typical types of a concept as shown in Table I. For each concept type, complementary cases may be specified for each functional

 ISBN: 978-1-61208-230-1

Table I
TYPICAL TYPES OF A CONCEPT

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 II
TYPICAL TYPES OF COMPLEMENTARY CASES

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

requirements sentence. Such complementary cases correspond to a subject, objects and a complement in a sentence. Each of such complementary cases also takes a type as shown in Table II. How to convert each sentence into a requirements frame is out of scope of this paper because we have already studied the issue in our previous works [4] [5].

We show examples of requirements frames and their corresponding original sentences in Figure 1. At the top of the figure, two typical functional requirements sentences are represented. Because the sentence #1 is about data flow from the system to be developed to an external system (UNIX server), DFLOW is chosen as a type of its concept. According to the definition of our requirements frames, DFLOW requires complementary cases such as Object, Source and Goal. Object corresponds to the data to flow. Source and goal correspond to be the source and the destination of the data flow. In the sentence #1, the information flows from the system to the external system. We thus assign Info, System, Extsystem types to each case as shown in the figure. A requirement #2 is also converted in the same way.

C. Rules

Because a requirements frame explicitly represents the semantic information about the original requirements sentence, we may simply check the types of cases to detect necessary quality requirements for each functional requirement. We thus simply construct if-then rules for deciding whether a specific quality requirement is necessary for a requirement frame. At the top in Figure 2, we show an example of such a rule (Rule A). The rule decides whether Interoperability is necessary for a requirement frame. The if-then part of the rule focuses on the types of cases about Concept, Source and Goal. Because a requirement frame #1 in the figure satisfies this condition, the rule decides Interoperability is necessary for the requirement frame #1. On the other hand, the rule does not decide Interoperability is necessary for a requirements frame #2 because the condition is not satisfied in it.

D. Discussion about the rule-based technique

We have written 14 rules for web-based information system [3], and the rules are applied in a case study [10]. One of the big problems of this technique is the effort for writing such rules. It takes about a few weeks for two experts of requirements engineering for writing such 14 rules. Even if the rules can be reused in the same domain and they can be improved during their usage, such expectations largely depend on the expertise of requirements analysts. Another problem is about their application. According to the result of the case study, an expert simply referred the results of rule application and he basically subjectively updated the original requirements. One of the reasons was that the expert considered the rules to be still immature and the rules should be manually improved during more applications. If such rules can be improved automatically along the progress of their usage, the effort for the rule users (normally, requirements analysts) largely decreases.

III. QUALITY REQUIREMENTS PREDICTION USING MACHINE LEARNING

The problem of obtaining quality requirements from a requirements frame can be formalized as a classification problem of obtaining classification rules from a finite data set. Let the set of input vectors be denoted by $I = B^n$ and the set of labels by $L = \{l_1, l_2, \cdots, l_m\}$, where $B = \{0, 1\}$. Training data D denotes a finite set $D \subseteq I \times L$.

Definition 3.1:

$$D = \{(d^{(1)}, l^{(1)}), (d^{(2)}, l^{(2)}), \cdots (d^{(n)}, l^{(n)})\}$$

Here, $d^{(1)}, d^{(2)}, \cdots, d^n$ are called instances, and $l^{(1)}, l^{(2),\cdots,l^{|D|}}$ are called classes to which the individual instances belong. $n = |D|$ denotes the number of training data samples, and m denotes the number of labels. Classification rules are represented by the function

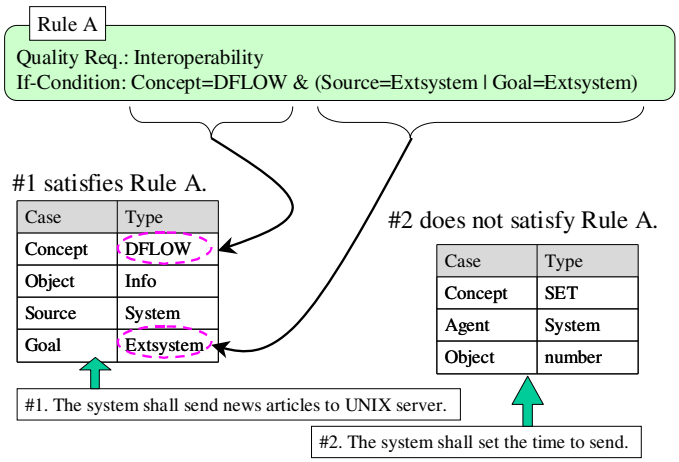

Figure 2. Examples of applying a rule to requirements frames.

δ: $I \Rightarrow L$. A classification problem is the problem of obtaining optimal δ for given D based on certain criteria.

Various learning algorithms are used in classification problems, such as support vector machines, decision trees, maximum entropy models, and naive Bayes classifiers. In choosing the learning algorithm, it is necessary to choose an algorithm that can suitably capture the structures included in the data set of the target classification problem [11].

In the case where a support vector machine is used, it is necessary to determine a kernel function that is suitable for the problem. A kernel function is a function that gives the similarity between instances. However, it is not easy to obtain a function that appropriately represents the semantic similarity between instances represented by requirements frames. As for a decision tree, although it is advantageous in that classification rules obtained can be readily understood, in the case of the type of problem being considered in this research, which requires manual preparation of training data, since the number of instances is limited, excessive segmentation might occur, resulting in degraded generalization ability. Furthermore, since the training data samples investigated in this research are sparse binary vectors, and it is assumed that the same instance might be classified into different classes, deterministic learning algorithms are not suitable. Therefore, in order to obtain appropriate results, it will be a better approach here to acquire corresponding relationships between inputs and outputs by directly using word co-occurrences rather than capturing complex contextual structures. Accordingly, a naive Bayes classifier is adopted here as a learning machine in view of the simplicity of the model and the ease of computation.

Representations of requirements frames of the specifications prepared by experts are converted into binary vectors. For example, the concepts of the case frame are associated with a 17-dimensional vector since the number of types is 17. This vector changes in accordance with the number of concepts defined in accordance with the relevant domain. Structures included in the concepts, such as objects, sources, and goals, are also represented as binary vectors of predetermined orders. Similarly, quality characteristics required for individual requirements frames are also converted into binary vectors.

A. Instance selection using cosine values

Binarized specifications and quality characteristics are considered as instances and classes of the instances, and the set of these will be denoted by R. A portion of the set R is used as the training data D. Here, half of the instances prepared are used as the training data D. The set of instances correctly classified by the learned function δ will be distinguished by attaching the subscript $_c$. The correct answer rate representing the ratio of correctly classified instances among the other half instances $D' = R - D$ will be referred to as the successful learning rate E_{suc}, and the correct classification rate for all instances will be referred to as the learning accuracy E_{acc}.

$$E_{suc} = \frac{|D'_c|}{|D'|} \tag{1}$$

$$E_{acc} = \frac{|D_c \cup D'_c|}{|R|} \tag{2}$$

In selecting instances, it is necessary to select training data samples uniformly in some sense from the set of instances. As a criterion of the similarity between instances, here, the sum of cosine values between learning data samples will first be used. The sum S_i of cosine values of an instance i will be defined as follows.

Definition 3.2:

$$S_i = \sum_{j(\neq i)}^{n} \frac{d^{(i)} d^{(j)}}{|d^i||d^j|} \tag{3}$$

By choosing instances with small values of S_i, orthogonal instances will be preferentially used for learning from the data set, so that unbiased instance selection can be expected.

In this research, four requirements specifications were picked up from the specifications of procurement examples at the Information Systems and Welfare Division of the Ministry of Economy, Trade and Industry of Japan and were converted into requirements frames, which were used as training data. The numbers of instances of the training data used in the experiment are given in the table below. The instances corresponding to the four requirements specifications were sorted in ascending order according to equation (3), and $\lfloor \frac{|R|}{2} \rfloor$ instances were picked up as the training data D.

Table III
NUMBERS OF INSTANCES OF REQUIREMENTS SPECIFICATIONS USED IN LEARNING EXPERIMENT.

Spec. 1	Spec. 2	Spec. 3	Spec. 4
37	41	52	58

In the learning experiment, the naive Bayes classifier scheme of WEKA [12], which is a machine learning platform, was used. In order to confirm the effect of cosine-based selection, a learning experiment in which instances were chosen at random was also conducted. Learning was performed for each quality characteristic of the individual specifications, and E_{suc} and E_{acc} values for each quality characteristic, as well as average E_{suc} and average E_{acc} for all quality characteristics, were obtained. The average E_{suc} values are shown in Figure 3, and the average E_{acc} values are shown in Figure 4, in which the horizontal axis represents the number of instances and the vertical axis represents the average E.

The cosine-based instance selection was effective when $|D|$ was small; however, there was a tendency for both E_{suc} and E_{acc} to decrease as $|D|$ increased. One reason for this tendency was that the cosine values of requirements frames with the same representation were added up into the sum S_i of cosine values for each instance, used in equation (3), so that frequently used requirements frames were excluded. In instance selection, some measure is needed to avoid adding up the cosine values of requirements frames with the same representation. Therefore, the sum defined by equation (4) below will be used hereafter.

Definition 3.3:

$$S_i = \sum_{j(d^j \neq d^i)}^{n} \frac{d^{(i)} d^{(j)}}{|d^i||d^j|} \tag{4}$$

B. Requirements frames representations and features

In order to adapt to the learning scheme, here, it is assumed that training data samples are represented by simple binary vectors. However, individual cases included in

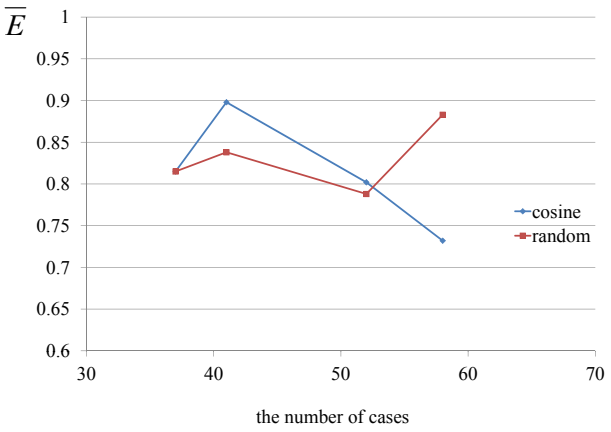

Figure 3. Relationship between the number of instances and E_{suc}.

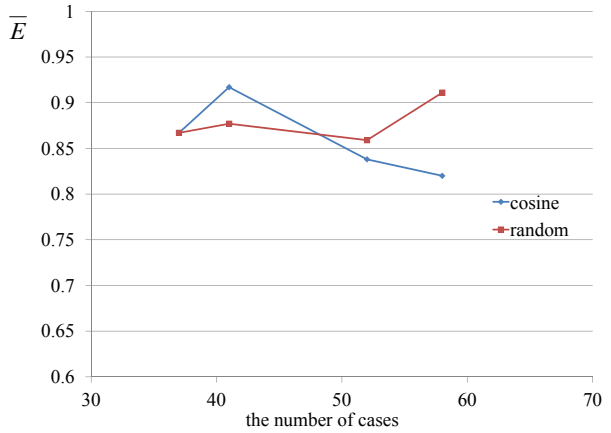

Figure 4. Relationship between the number of instances and E_{acc}.

requirements frames representations include elements that are necessary for quality characteristics identification and those that are not necessary. Here, of the elements of the case structure, such as concept, function, and reliability, elements that are necessary for quality characteristics identification were revealed experimentally. The necessary elements correspond to features used in learning, which can be utilized for appropriate selection of example problems. Note that, in this paper, we place the term "feature" in the context of machine learning; so, the meaning of it is different from that in software engineering.

Features necessary for quality characteristics identification were estimated for Specification 4, which was found to have the lowest level of E in the results described in the preceding section. Table IV shows the individual requirements frames representations included in the instances of Specification 4.

 ISBN: 978-1-61208-230-1

Table V shows the results of learning in which half of the instances were selected as training data based on cosine values in view of five frames representations.

Table IV
REQUIREMENTS FRAMES REPRESENTATIONS OF SPECIFICATION 4.

Concept	DFLOW, CFLOW, SET, RET, UPDATE, DEL, INS, MANIP
Agent	System, Human
Goal	System, Human, Extsystem
Object	Data, Info, Function
Source	System, Human, Info

Table V
RESULTS OF LEARNING IN WHICH INSTANCES WERE SELECTED FOR EACH REQUIREMENTS FRAME REPRESENTATION.

	Concept	Agent	Goal	Object	Source
E_{acc}	0.90	0.83	0.70	0.86	0.70
E_{ttl}	0.93	0.87	0.81	0.89	0.81

The highest E_{suc} and E_{acc} were obtained when instances were selected based on concept among the requirements frames representations. Concept has the highest order among the requirements frames representations, effectively serving for instance separation, so that it is suitable feature that can be used for learning.

C. Instance selection based on concept

In order to confirm the estimation in Section III-B, a learning experiment in which instances were selected based on concept was conducted. Regarding the four requirements specifications used in Section III-A, instances were selected based on concept among the requirements frames representations of the individual specifications. When the number of instances was less than half, orthogonality of the remaining instances was evaluated based on equation (4), and instances were selected accordingly until the number reached half. E_{suc} and E_{acc} for each quality characteristic, as well as the average E_{suc} and E_{acc}, were obtained. The E_{suc} average values are shown in Figure 5, and the E_{acc} average is shown in Figure 6. For the purpose of comparison, the results of learning in the case where instances were chosen at random and the results of learning in the case where instances were chosen at random from concept and are also plotted. The experimental results demonstrate that the success rate in the case where instances were selected from concept monotonically increased as the number of instances increased, suggesting its effectiveness in learning. In the case where instances were selected from concept, a maximum success rate of about 0.91 was achieved for unknown data.

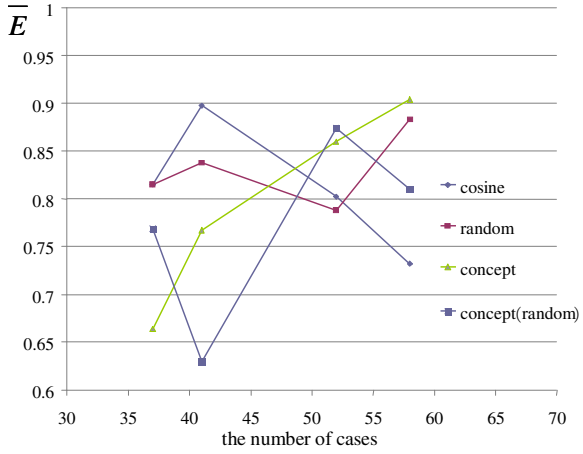

Figure 5. E_{suc} in the case where instances were selected from concept.

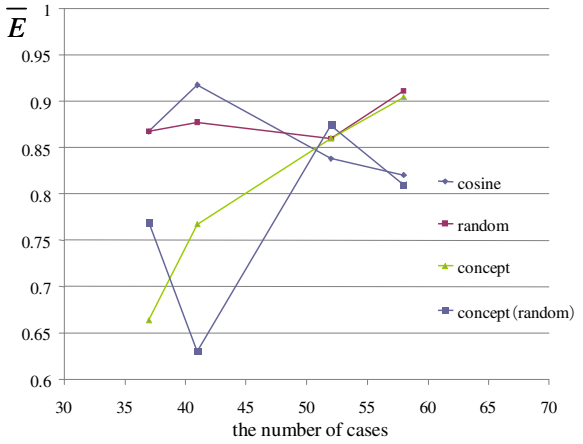

Figure 6. E_{acc} in the case where instances were selected from concept.

In the case where instances were chosen at random, the success rate varied considerably depending on the number of instances. From what has been described above, it is estimated that, when obtaining the quality characteristics of a large-scale specification, it will be effective to select instances in such a manner that overall quality characteristics are determined based on concept in requirements frames representations and that further classification is performed based on cases as needed.

IV. RELATED WORK

There are several studies how to define each quality requirement, but most of them requires the huge amount of human effort. In ISO 25021 [13], 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 [14]. In Architecture Tradeoff Analysis Method (ATAM) [15] [16], a template for quality requirements called “quality attribute scenario” is provided

 ISBN: 978-1-61208-230-1

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. [17], an empirical data of the most common quality attributes was shown based on the ATAM. This kind of empirical data help requirements analysts to specify quality requirements. For similar systems, similar kinds of quality requirements are normally required. For example, most functions in typical Web browsers require security and usability, but do not so require accuracy and fault tolerance. This kind of analysis helps analysts to validate their quality requirements definition in a specification by comparing to specifications of other similar systems [18]. However, such analysis does not directly point out missing quality requirements in each functional requirement. The analysis just suggests that the specification is unbalanced with respect to quality requirements definition. UML is the most popular semi-formal notation for software development now, and there are some challenges to introduce quality requirements into it [19] [20]. However, how to specify such introduced information is normally out of scope in each research. Using ontology, dictionary and/or thesaurus [21] is one of the useful ways to improve the quality of requirements with respect to semantic aspect. However, it is a little bit weak because simple words/terms matching cannot completely represent the semantic information in a requirement.

There are a lot of software engineering researches using machine-learning techniques, such as cost estimation [22], defect prediction [23] and design pattern mining [24]. Most researches focus on variable selection rather than training data selection because plenty of variables exist in such application area. For requirements engineering researches, machine-learning techniques are rarely used. One of the exceptions is a method for classifying non-functional requirements (NFR) automatically using a machine learning technique [25]. In this method, usual natural language documents are used for the classification, but semi-formal notation is used in our research. Training data sets are chosen empirically so as to effectively classifying NFRs in this research, but training data sets are systematically chosen based on the theory of machine learning in our research.

V. Conclusion and Future Work

In this paper, we proposed and evaluated a new technique for predicting quality requirements necessary for a functional requirement based on the machine learning. The main contribution of this research is to avoid human effort for preparation of detection, i.e., writing and improving detection rules manually. In our technique, each functional requirement is represented in X-JRDL, which is a semi-formal language based on case grammar. Because X-JRDL explicitly represents the semantic structure of a functional requirement, we can easily decide which kinds of quality requirements are necessary for a functional requirement.

In our previous work [3] [6], we proposed a rule-based technique for predicting necessary quality requirements. The results of the previous work were not bad, but it took a lot of effort to write rules for prediction. In addition, the quality of prediction largely depended on the expertise of the people who wrote the rules. Our new technique avoids such problems because detection rules are automatically generated by machine learning, and the rules can be also automatically improved.

In this research, requirements frames representations that were suitable as features were revealed experimentally through selection of training data based on cosine values. In the case of a large-scale requirements specification, uniform selection of instances from concept among requirements frames representations gave the most appropriate quality characteristics. Although duplicates increased as the data volume increased in the case of instance selection based on simple similarity of cosine values, a high success rate was achieved by selectively choosing instances from suitable features.

The results shown here are average values of E_{suc} and E_{acc}. In view of the individual quality characteristics, the results for reliability were lower than those for the other quality characteristics. Although independence of individual attributes is assumed when classes are given by naive Bayes classifiers, presumably, some dependencies exist in reality.

In future research, it is necessary to examine instance representations and selection methods that are free of such dependencies. We would like to also develop a supporting tool (a CASE tool) to support a requirements analyst to write a requirements specification based on our proposed technique.

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 Software*, vol. 25, no. 2, pp. 22–24, Mar./Apr. 2008.

[3] H. Kaiya and A. Ohnishi, "Finding incorrect and missing quality requirements definitions using requirements frame," *IEICE Transactions*, vol. 95-D, no. 4, pp. 1031–1043, 2012.

[4] A. Ohnishi, "Software requirements specification database based on requirements frame model," in *ICRE*, 1996, pp. 221–228.

[5] Y. Matsuo, K. Ogasawara, and A. Ohnishi, "Automatic transformation of organization of software requirements specifications," in *RCIS*, 2010, pp. 269–278.

[6] H. Kaiya and A. Ohnishi, "Quality requirements analysis using requirements frames," in *QSIC*, 2011, pp. 198–207.

[7] R. Shank, "Representation and Understanding of Text," *Machine Intelligence*, vol. 8, pp. 575–607, 1977.

 ISBN: 978-1-61208-230-1

[8] International Standard ISO/IEC 9126-1, "Software engineering - Product quality - Part 1: Quality model," 2001.

[9] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Academic Publishers, 1999.

[10] H. Kaiya and A. Ohnishi, "Improving software quality requirements specifications using spectrum analysis," in *COMPSAC Workshops*, 2012, pp. 379–384.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning*, new ed. Springer-Verlag, 2008.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[13] International Standard ISO/IEC 25021, "Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Quality measure elements," Oct. 2007.

[14] D. Firesmith, "Quality Requirements Checklist," *Journal of Object Technology*, vol. 4, no. 9, pp. 31–38, Nov.-Dec. 2005.

[15] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method," in *IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 1998, pp. 68–.

[16] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003.

[17] I. Ozkayad, L. Bass, R. S. Sangwan, and R. L. Nord, "Making Practical Use of Quality Attribute Information," *IEEE Software*, vol. 25, no. 2, pp. 25–33, Mar./Apr. 2008.

[18] H. Kaiya, M. Tanigawa, S. Suzuki, T. Sato, and K. Kaijiri, "Spectrum analysis for quality requirements by using a term-characteristics map," in *CAiSE*, 2009, pp. 546–560.

[19] Y. Zhang, Y. Liu, L. Zhang, Z. Ma, and H. Mei, "Modeling and Checking for Non-Functional Attributes in Extended UML Class Diagram," in *Annual IEEE International Computer Software and Applications Conference (COMPSAC2008)*, 2008, pp. 100–107.

[20] Z. M. Yi Liu and W. Shao, "Integrating Non-Functional Requirement Modeling into Model Driven Development Method," in *17th Asia-Pacific Software Engineering Conference (APSEC 2010)*, Dec. 2010, pp. 98–107.

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

[22] D. G. e Silva, M. Jino, and B. T. de Abreu, "Machine learning methods and asymmetric cost function to estimate execution effort of software testing," *Software Testing, Verification, and Validation, 2008 International Conference on*, vol. 0, pp. 275–284, 2010.

[23] E. Ceylan, F. O. Kutlubay, and A. B. Bener, "Software defect identification using machine learning techniques," *EUROMICRO Conference*, vol. 0, pp. 240–247, 2006.

[24] R. Ferenc, Á. Beszédes, L. J. Fülöp, and J. Lele, "Design pattern mining enhanced by machine learning," in *ICSM*, 2005, pp. 295–304.

[25] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requir. Eng.*, vol. 12, no. 2, pp. 103–120, 2007.

 ISBN: 978-1-61208-230-1