Quality Requirements Analysis with Machine Learning

Tetsuo Tamai[1] and Taichi Anzai[2]
[1]*Faculty of Science and Engineering, Hosei University, Tokyo, Japan*
[2]*NTT Com Solutions Corp., Tokyo, Japan*

Keywords: Quality Requirements, Machine Learning, Requirements Classification, Natural Language Processing.

Abstract: The importance of software quality requirements (QR) is being widely recognized, which motivates studies that investigate software requirements specifications (SRS) in practice and collect data on how much QR are written vs. functional requirements (FR) and what kind of QR are specified. It is useful to develop a tool that automates the process of filtering out QR statements from an SRS and classifying them into the quality characteristic attributes such as defined in the ISO/IEC 25000 quality model. We propose an approach that uses a machine learning technique to mechanize the process. With this mechanism, we can identify how each QR characteristic scatters over the document, i.e. how much in volume and in what way. A tool *QRMiner* is developed to support the process and case studies were conducted, taking thirteen SRS documents that were written for real use. We report our findings from these cases.

1 INTRODUCTION

The importance of software quality requirements is being widely recognized, although the term is somewhat confusing. Historically, the term *non-functional requirements (NFR)* has been used longer (Mylopoulos et al., 1992; Glinz, 2007; Chung and do Prado Leite, 2009) and *quality requirements (QR)* was concocted to replace it, avoiding ambiguity that derives from the negative definition with the prefix "non" (Blaine and Cleland-Huang, 2008). But "quality requirements" is sometimes confused with "requirements quality," which denotes quality of requirements represented as specifications. Throughout this paper, we use the term quality requirements (QR) but in most places it can be interchanged with NFR without confusion.

In the introduction of the IEEE Software special issue on quality requirements (Blaine and Cleland-Huang, 2008), three challenges to the quality requirements research are pointed out.

1. How to elicit quality requirements.
2. How to resolve trade-offs between quality requirements.
3. How to measure quality requirements.

Another crucial issue is "how to implement quality requirements into products." Most of quality requirements research focus on one or multiple of the above four issues, e.g. QR elicitation (Chung and do Prado Leite, 2009), trade-off resolution (Regnell et al., 2008; Svensson et al., 2009), QR measurement (Glinz, 2008; Knauss and Boustani, 2008) and QR implementation (Wei et al., 2012).

However, before addressing these problems, it would be worthwhile to grasp how quality requirements are dealt with in practice and analyze them (Svensson et al., 2013). It will make a good base to explore ways for eliciting, representing and implementing QR. For that purpose, we need a method to identify where in a software requirements specification (SRS) quality requirements are stated and which characteristic class each requirement belongs to. If such detection is easily implemented, we can have a good view on how much amount of quality requirements are defined compared to functional requirements, how they are distributed over the characteristic categories, i.e. performance, compatibility, usability, reliability, security, etc., and how they are structured in the whole specification document.

In this paper, we propose an approach for analyzing QR found in an SRS in terms of their volume, balance and structure. The SRS are assumed to be written in natural language, Japanese in our case. Natural language processing and machine learning techniques, particularly deep learning, are employed to detect and classify quality requirement sentences.

The paper is organized as follows. Section 2 dis-

Tamai, T. and Anzai, T.
Quality Requirements Analysis with Machine Learning.
DOI: 10.5220/0006694502410248
In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 241-248
ISBN: 978-989-758-300-1

cusses how QR are described in practice and what benefit can be obtained by grasping the QR amount and structure in the SRS. Related work are surveyed in Section 3. A method for mining QR in SRS is introduced in Section 4. Section 5 presents the results of our experiments. The paper ends with discussions in Section 6 and conclusions in Section 7.

2 HOW ARE QUALITY REQUIREMENTS DESCRIBED IN PRACTICE?

It is often argued that the importance of requirements engineering is not well recognized in industry as it should be (Kaindl et al., 2002). Yet, software requirements specifications are actually written extensively in practice, although no advanced RE techniques may be used and SRS's are mostly written in natural language. In those requirements documents, the writers may not be much conscious of the distinction between FR and QR but still both types of requirements are described.

Our research objective is to investigate an SRS in practice and collect data on how much QR (or NFR) are written vs. FR and what kind of QR are actually written, e.g. how they are distributed over quality characteristics as categorized in the ISO/IEC 25000 standard (ISO/IEC, 2014). The major objective of developing such a QR classifier is to support a requirements engineer in writing requirements specifications. In the middle of the specification writing process or after it, she uses the classifier to examine how the written requirements are classified. Then, she may check the following points.

1. Check the distribution balance between FR and QR, to see if they are within the expectation. In general, as the amount of QR tends to get lower than necessary, it had better be watched carefully and if that is the case, it may imply more QRs should be added.
2. Check the distribution balance among the property characteristics. In our approach, we discern eight categories for QR and four for FR. Some may be written densely and others may be written scarcely or none. It does not mean that uniformly distributed descriptions over the quality characteristics is ideal. As M. Glinz pointed out, "if we have an implicit shared understanding among stakeholders and developers about a quality requirement, there's no need to specify it." (Glinz, 2008). At the same time, sensitivity to a particular quality characteristic is naturally dependent on the target domain and the system property. But still, if descriptions of some quality characteristic are null or much less than expectation, it is worth noting it and examining the reason. It is informative just to expose tacit understanding explicitly. On the other hand, if a characteristic is densely described but that quality does not look so crucial to the target system, the description may be redundant or there is some imbalance in the SRS. Thus, looking at the distribution of QR can be a good cue to start analyzing the requirements document. For example, if the amount of security requirements is relatively small for a system that is supposed be security-sensitive, it may imply specification rewriting is necessary.
3. It is often recommended to write FR and QR separately (Glinz, 2007). For example, FR and QR are to be described in separate chapters in the Volere template (Robertson and Robertson, 1999). IEEE Std. 830-1998 (IEEE, 1998) also recommends to write them separately in most of the template examples. Wiegers's book (Wiegers and Beatty, 2013) introduces another template adapted from the IEEE 830, which also separates FR and NFR. As not so many authors of SRS use the Volere or Wieger template or are conscious of IEEE 830, it is not expected that SRS found in industry has such a clear structure, nor does the separate description always imply a good quality of the SRS. But still, it is often instructive to grasp how QR's are distributed over the document and how they are mixed with FR or not. Particularly, visualization of the distribution is helpful.
4. In case some requirement instance is categorized into a different class from the expected class, it may simply imply poor performance of the classifier but in some case it may imply the way of expressing the requirement is not appropriate.

Requirements specification readers such as users or implementers can also use the tool to grasp the general view of the system to be used or to be implemented. It will also benefit maintenance engineers who need to locate specific requirements such as performance requirements in the requirements specification.

3 RELATED WORK

There are already some work that attempt to discern and classify QR in SRS. For example, the work by H. Kaiya et al. (Kaiya et al., 2008) calls the distribution of requirements sentences across QR characteris-

tics as *spectrum* and propose a method for detecting spectra. However, their policy of distinguishing QR from FR is not clear. Actually, they take functional requirements statements like "printing shall be supported" and sort it into quality like "interoperability." To execute the sorting, they first classify requirements quality characteristics by hand (Kaiya et al., 2008). Then, they partially mechanized the process using a keyword-to-quality matrix (Kaiya and Ohnishi, 2012) but the matrix is constructed for each system by a human, although there is a possibility of reusing the existing one if the application domain is the same.

Svensson et al. (Svensson et al., 2009) did much larger-scale work of analyzing QR in practice. They took more than 2000 requirements for systems of a company, elaborately coded each requirement with types and characteristics and accumulated and analyzed them.

These are based on manual work but an automated approach to detecting and classifying QR was pioneered by the work presented by the Cleland-Huang et al. at the Requirements Engineering Conference, 2006 (RE06) (Cleland-Huang et al., 2006). They used a weighted term frequency measure, basically the same as TF-IDF (term frequency-inverse document frequency) to calculate similarity between documents and classify the requirements with that measure. Fifteen requirements specifications developed as term projects by MS students were used to evaluate their proposed approach and also a large requirements document from industry was used for a case study.

Similar work followed. One was conducted by the same group, applied to healthcare systems (Rahimi et al., 2014). Another example is the work by Slankas & Williams, also applied to the healthcare domain (Slankas and Williams, 2013). They used eleven documents related to electronic health records (EHR), including two request for proposals (RFP) and two manuals for open-source projects. They classify the requirements by the k-nearest neighbor method. They also evaluated the naive Bayes and the Support Vector Machine (SVM) learning methods.

Casmayor et al. proposed what they called semi-supervised learning approach (Casamayor et al., 2010). It starts with a labeled set of data but then the EM (expectation maximization) method is used to iteratively enhance learning. They used the same data as that of (Cleland-Huang et al., 2006), which was made available at the PROMISE software engineering repository.

Vias & Robinson used various types of documents found in open-source software projects to discover and classify requirements. Data of thirty projects were obtained from SourceForge and on an average 4,962 sentences were collected for each project. For the QR classification, it used McCall et al.'s quality model but as the authors admit, the model is old (published in 1977) and does not fit well to modern software systems.

Some work in natural language processing (NLP) is worth noting. The natural language processing techniques have been widely used for requirements engineering but most of them are concerned with SRS evaluation and improvement. One successful area is applying NLP to detecting ambiguity in words and phrases in the SRS, e.g. (Chantree et al., 2006). There is also an example of applying NLP to use cases for detecting ambiguity by Fantechi et al. (Fantechi et al., 2002).

Text mining employing document similarity detection is also widely used, particularly in various recommendation systems. A typical area is code recommendation, e.g. (Watanabe and Masuhara, 2011). In the RE community, it is often used for the traceability analysis (Cleland-Huang et al., 2010).

4 QUALITY REQUIREMENTS MINING AND CLASSIFICATION PROCESS

4.1 QR Mining Process Flow

We propose a method for mining QR in an SRS. The workflow of our approach is as illustrated in Figure 1. The explanation of the work in the flow will be given in the following subsections.

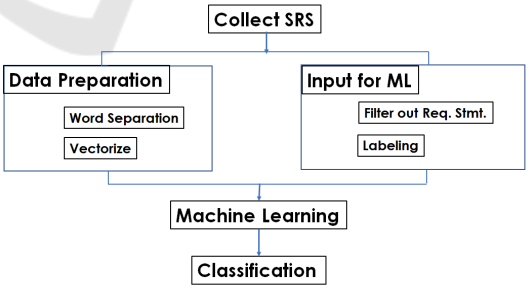

Figure 1: QR Mining Workflow.

4.2 SRS Collection

We collected thirteen SRSs available on the web, issued from local governments or other public institutions in Japan. They are listed in Table 1. Most of them are Requests for Proposal (RFP) for information systems but there are some others such as RFP

for a medical system. Moreover, two documents for standard QR writing, "Grade table of non-functional requirements" issued from Information Technology Promotion Agency, Japan (IPA) and "Non-functional requirements indicators" issued from Japan Users Association of Information Systems (JUAS) were added, because relatively fewer QRs are found in those collected SRSs. In total, 11,538 requirements sentences were collected as shown in Table 1. All of them are written in Japanese.

Use of RFP has the following advantage.

- They are publicly available.
- As they are used as the base for contracts, the descriptions are mostly unambiguous.

Most of the past related work used student project data or open source project data. In some cases, data from industry are used but they are proprietary and not easily available in public. Students projects are small in size and requirements are not written extensively. Open source projects in general do not give much weight on requirements writing as many of them adopt the agile process. Compared to them RFP's are given much efforts and time for creation. Among the past related work, Slankas & Williams (Slankas and Williams, 2013) is rather exceptional as they included two request for proposals (RFP), although the domain is restricted to the healthcare systems.

4.3 Labeling

For the classification categories of QR, we used the standard ISO/IEC 25030:2007 (ISO/IEC, 2014). It provides requirements and recommendations for the specification of software quality requirements. It is an extended version of ISO/IEC 9126 quality model. Eight quality property characteristics are defined: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. Each of the eight characteristics category is further composed of sub-characteristics as shown in Fig. 2.

To train and verify a machine learning system, we have to label the requirements data with a gold standard, i.e. with "correct" categories assigned to all the sentences. The authors did it manually as well as checked the results within us. We extracted requirements sentences from the SRS but still some non-requirement sentences remained. So, they were sorted to non-R, FR or QR at the top level. QRs were labeled with the eight categories following the ISO/EIC 25000. Table 2 shows the distribution of the labels.

We also find FRs and they are classified into the following four categories.

1. requirements on user interface
2. requirements on system functions
3. requirements on database
4. requirements on external interface

4.4 Japanese Morphological Analysis

In Japanese writing, words are not separated by spaces and so for the preparation of natural language processing, a morphological analyzer has to be used to separate words and determine part of speech. We used Kuromoji (kuromoji@atilika.com, 2017) and MeCab (Kudo, 2013) for that purpose. The two have similar capabilities but we used Kuromoji for preparing input to machine learning and MeCab for analyzing the relations of the document and term spaces.

As we used ANN (artificial neural network) for machine learning, we had to vectorize the input text. For that purpose, the tools Word2Vec (Mikolov et al., 2013a) and Doc2Vec (Mikolov et al., 2013b) are available. Both tools themselves exploit deep learning, and are so powerful that they are currently used almost as the standard. We used them, particularly Doc2Vec, for our purpose and found it can be used without any problem for Japanese language processing.

4.5 Deep Learning

Starting from the work by Cleland-Huang et al., most of the related work used information retrieval (IR) techniques for detecting and classifying QRs. The use of traditional machine learning methods such as k-nearest neighborhood and naive Bayesian is mentioned in some literature but the results are not extensively reported. However, in the past decade, machine learning, particularly deep learning has made a big advance as well as natural language processing like Word2Vec and Doc2Vec. We have made use of those novel technologies as much as possible.

To select an appropriate machine learning method, we conducted some preliminary experiments, including comparison of the conventional multi-layered perceptrons and Convolutional Neural Network (CNN). As the result, CNN significantly outperformed the multi-layered perceptrons. Considering other factors as well, we adopted one of the deep learning technique, CNN. It was originally invented by K. Fukushima as *Neocognitron* (Fukushima and Miyake, 1982).

For implementation, we used an existing tool *Chainer*, combined with related tools packaged in the

Table 1: SRS List.

No.	Issued from	Systems	Size (lines)
1	Moriyama City	Sewerage accounting system	330
2	NIRS	Medical information system	7083
3	Okayama City	Attendance management system	168
4	Nara Prefecture	House construction registration system	50
5	Hayama Town	Public service company management system	88
6	Kanda Town	Public health management sytem	744
7	Kudarimatsu City	School meal management system	126
8	Yokohama City	Library information system	1458
9	JUAS	Non-functional requirements indicators	288
10	Kyoto Prefecture	Total information system	377
11	Kyoto Prefecture	Library information system	552
12	IPA	Grade table of non-functional requirements	201
13	Ashikaga City	Sewerage accounting system	73
		total	11,538

NIRS: National Institute of Radiological Sciences
JUAS: Japan Users Association of Information Systems
IPA: Information Technology Promotion Agency, Japan

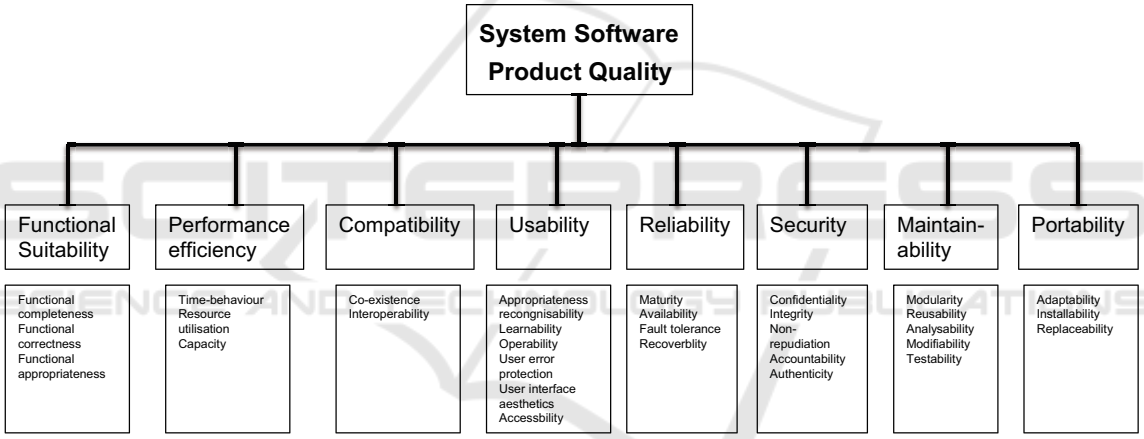

Figure 2: System/software product quality.

Table 2: Label distribution.

No.	Categories	Number
0	Non-requirements	1530
1	Req. on user interface	2870
2	Req. on system functions	4756
3	Req. on database	140
4	Req. on external interface	168
5	Functional suitability	227
6	Performance efficiency	148
7	Compatibility	152
8	Usability	265
9	Reliability	145
10	Security	283
11	Maintainability	304
12	Portability	60
	total	11,538

Python machine learning library. As well known, parameter tuning is important for this kind of work. We tuned parameters such as the number of units in the hidden layers, the number of epochs, the number of minibatches and so on. They were decided by repeated experiments.

The data were randomly divided into two sets with the size ratio 9:1. The former is for training and the latter is for testing. The partition was repeated a number of times.

5 RESULTS

We named our a tool *QRMiner*. The top-level classification is between non-R, FR and QR. The results are as shown in Table 3

Table 3: Results of QRMiner: FR and QR.

	precision	recall	F value
non-R	0.86	0.84	0.85
FR	0.89	0.94	0.91
QR	0.70	0.54	0.61

The results of QR category classification are as shown in Table 4.

Table 4: Results of QRMiner: QR characteristics.

QR	precision	recall	F value
Functional suitability	0.44	0.42	0.43
Performance efficiency	0.67	0.62	0.64
Compatibility	0.38	0.27	0.32
Usability	0.28	0.31	0.29
Reliability	0.45	0.39	0.42
Security	0.61	0.63	0.62
Maintainability	0.45	0.54	0.49
Portability	0.25	0.20	0.22

One of our original results is the detection and classification of FRs. In the literature, there is almost no work on classifying FRs. Our result shows that the same technique is applicable to classify FRs. As even the way of categorizing FRs has been given very little attention in the past (Sharma and Biswas, 2015), this can be a starting point for exploring research on the detection and classification of FRs.

6 DISCUSSIONS

6.1 Usage in the RE Process

As we discussed in the introduction, the QR mining can be used in different phases of the RE process cycle.

Elicitation Phase. In the elicitation phase, requirements may not be duly documented but as our QR mining is applicable even to rough descriptions of requirements, it can be used to check elicited QR in the middle of the elicitation process.

Documentation Phase. Similarly, as unfinished requirements documents can be analyzed with this tool, findings obtained from QR mining can be used to give a feedback when writing an SRS.

Evaluation Phase. Probably, the most natural usage of QR mining is in the phase of evaluation when the SRS is completed and passed to a quality assurance team for reviewing. The results of the QR mining will be informative in analyzing and improving the SRS.

The cycle of analysis and improvement may be repeated and the mining can be employed in every cycle.

6.2 Threats to Validity

We studied thirteen systems, which are large enough compared to the past related work. But still, it is desirable to collect much more SRS from a variety of areas.

The gold standard defined by the authors may be a factor of threat to validity. We plan to ask other experts to validate our decisions.

All the documents dealt with are written in Japanese and so the lessons learned may not be applicable to other languages, e.g. English. However, the techniques used were not language dependent.

6.3 Findings

We explain and discuss our findings through this case study.

QR Ratio. One of our questions was how much QR are written in SRS or what is the ratio of QR versus FR. In total, we dealt with 9518 requirements and 1584 among them were QR, which occupies 16.6% (this is based on human counting but counting by QRMiner is about the same). This ratio is lower than the data reported by Olsson et al.(Olsson et al., 2007) but we need more data to interpret this result.

Usefulness of QRMiner. Precision and recall for FR are 0.89 and 0.94 and those for QR are 0.70 and 0.54, which are satisfactorily high. Of course, we have to make an effort to obtain higher performance, which is our near future work.

Scarce Quality Characteristics. Among the eight QR categories, the number of instances of portability was 60, which is relatively low. It may indicate the nature of the target systems.

QR Structure. Some SRS show well-structured descriptions, separating QR and FR clearly but some do not show such a structure clearly. It shows QRMiner is useful in illustrating the structure of SRS.

7 CONCLUSIONS

We proposed a QR mining approach and developed a tool *QRMiner* that supports it. With the case studies, it was shown that the tool produces informative output to analyze SRS.

The evaluation of the tool in terms of precision and recall is satisfactory but there is a good room for improvement.

Original points of our work can be summarized as follows..

- We used the up-to-date machine learning technology, deep learning and Doc2Vec, which have enhanced performance of QRMiner considerably.
- We used the real practical requirements documents open to the public, rather than data from student projects or open-source projects.
- The scale of the collected requirements is large enough.
- We showed that our approach can be applied not only to QR but also to FR.
- We showed that using non-English SRS is not restriction but indicates wide applicability of the current machine learning and natural language processing technology.

REFERENCES

Blaine, J. D. and Cleland-Huang, J. (2008). Software quality requirements: How to balance competing priorities. *IEEE Software*, 25(2):22–24.

Casamayor, A., Godoy, D., and Campo, M. (2010). Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology*, 52(4):436–445.

Chantree, F., Nuseibeh, B., de Roeck, A., and Willis, A. (2006). Identifying nocuous ambiguities in natural language requirements. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 59–68.

Chung, L. and do Prado Leite, J. C. S. (2009). *On Non-Functional Requirements in Software Engineering*, volume 2072 of *LNCS*, pages 363–379. Springer.

Cleland-Huang, J., Czauderna, A., Emenecker, J., and Gibiec, M. (2010). A machine learning approach for tracing regulatory codes to product specific requirements. In *International Conference on Software Engineering (ICSE'10)*, pages 155–164.

Cleland-Huang, J., Settimi, R., Zou, X., and Solc, P. (2006). The detection and classification of non-functional requirements with application to early aspects. In *Requirements Engineering, 14th IEEE International Conference*, pages 39–48. IEEE.

Fantechi, A., Gnesi, S., Lami, G., and Maccari, A. (2002). Application of linguistic techniques for use case analysis. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02)*, pages 157–164.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.

Glinz, M. (2007). On non-functional requirements. In *15th International Conference on Requirements Engineering (RE'07)*, pages 21–26, Delhi, India.

Glinz, M. (2008). A risk-based, value-oriented approach to quality requirements. *IEEE Software*, 25(2):34–41.

IEEE (1998). Recommended practice for software requirements specifications. Technical report, IEEE. IEEE Std 830-1998.

ISO/IEC (2014). Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)ISO/IEC 25000:2014). ISO/IEC, ISO/IEC.

Kaindl, H., Brinkkemper, S., Bubenko Jr, J. A., Farbey, B., Greenspan, S. J., Heitmeyer, C. L., do Prado Leite, J. C. S., Mead, N. R., Mylopoulos, J., and Siddiqi, J. (2002). Requirements engineering and technology transfer: Obstacles, incentives and improvement agenda. *Requirements Engineering*, 7:113–123.

Kaiya, H. and Ohnishi, A. (2012). Improving software quality requirements specifications using spectrum analysis. In *Computer Software and Applications Conference Workshops (COMPSACW)*, pages 379–384.

Kaiya, H., Sato, T., Osada, A., Kitazawa, N., and Kaijiri, K. (2008). Toward quality requirements analysis based on domain specific quality spectrum. In *SAC '08 Proceedings of the 2008 ACM symposium on Applied computing*, pages 596–601.

Knauss, E. and Boustani, C. E. (2008). Assessing the quality of software requirements specifications. In *16th IEEE International Requirements Engineering Conference*.

Kudo, T. (2013). MeCab: Yet another Japanese morphological analyzer. https://github.com/taku910/mecab.

kuromoji@atilika.com (2017). Kuromoji. http://www.atilika.org/.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mylopoulos, J., Chung, L., and Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6).

Olsson, T., Svensson, R. B., and Regnell, B. (2007). Nonfunctional requirements metrics in practice — an empirical document analysis. In *Workshop on Measuring Requirements for Project and Product Success*.

Rahimi, M., Mirakhorli, M., and Cleland-Huang, J. (2014). Automated extraction and visualization of quality concerns from requirements specifications. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 253–262. IEEE.

Regnell, B., Svensson, R., and Olsson, T. (2008). Supporting roadmapping of quality requirements. *IEEE Software*, 25(2).

Robertson, S. and Robertson, J. (1999). *Mastering the Requirements Process*. Addison-Wesley.

Sharma, R. and Biswas, K. K. (2015). Functional requirements categorization: Grounded theory approach. In *Proceedings of 2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 301–307, Barcelona, Spain. IEEE CS Press.

Slankas, J. and Williams, L. (2013). Automated extraction of non-functional requirements in available documentation. In *Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on*, pages 9–16. IEEE.

Svensson, R. B., Gorschek, T., and Regnell, B. (2009). Quality requirements in practice: An interview study in requirements engineering for embedded systems. In Glinz, M. and Heymans, P., editors, *REFSQ 2009*, volume 5512 of *LNCS*, pages 218–232.

Svensson, R. B., Olsson, T., and Regnell, B. (2013). An investigation of how quality requirements are specified in industrial practice. *Information and Software Technology*, 55(7):1224–1236.

Watanabe, T. and Masuhara, H. (2011). A spontaneous code recommendation tool based on associative search. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation (SUITE'11)*, pages 17–20.

Wei, B., Jin, Z., Zowghi, D., and Yin, B. (2012). Automated reasoning with goal tree models for software quality requirements. In *Computer Software and Applications Conference Workshops (COMPSACW)*, pages 373–378.

Wiegers, K. and Beatty, J. (2013). *Software Requirements, 3rd edition*. Microsoft Press.