A Semi-automated Approach to Generate an Adaptive Quality Attribute Relationship Matrix

Unnati Shah[1](✉), Sankita Patel[2], and Devesh Jinwala[3]

[1] C. K. Pithawala College of Engineering and Technology, Surat 395007, India
unnati.shah25@gmail.com

[2] National Institute of Technology, Surat 395007, India

[3] Indian Institute of Technology, Jammu, Jammu 181221, Jammu and Kashmir, India

Abstract. **[Context and Motivation]** A critical success factor in Requirements Engineering (RE) involves recognizing conflicts in Quality Requirements (QRs). Nowadays, Quality Attributes Relationship Matrix (QARM) is utilized to identify the conflicts in QRs. The static QARM represents how one Quality Attribute (QA) undermines or supports to achieve other QAs. **[Question/Problem]** However, emerging technology discovers new QAs. Requirements analysts need to invest significant time and non-trivial human effort to acquire knowledge for the newly discovered QAs and influence among them. This process involves searching and analyzing a large set of quality documents from literature and industries. In addition, the use of static QARMs, without knowing the purpose of the QRs in the system may lead to false conflict identification. Rather than taking all QAs, domain-specific QAs are of great concern for the system being developed. **[Principal ideas/results]** In this paper, we propose an approach which is aimed to build an adaptive QARM semi-automatically. We empirically evaluate the approach and report an analysis of the generated QARM. We achieve 85.67% recall, 59.07% precision and 69.14% F-measure to acquire knowledge for QAs. **[Contributions]** We provide an algorithm to acquire knowledge for domain-specific QAs and construct an adaptive QARM from available unconstrained natural language documents and web search engines.

Keywords: Requirements Engineering · Quality ontology · Quality Attribute Relationship Matrix

1 Introduction

In Requirements Engineering (RE), Quality Requirements (QRs or Non-Functional Requirements) describe the overall behavior of the system [1]. Requirements analysts have to acquire knowledge of Quality Attributes (QAs) and influences among them, when they specify QRs for the system. In literature, there exist various approaches to specify QRs [2–12]. We divide them into two categories such as the NFR Framework-based approach and QARM (Quality Attribute Relationship Matrix) based approach.

The NFR Framework [4] is a systematic approach to define QRs for the system. It provides visibility to relevant QAs and their interdependencies[1]. The NFR Framework is based on the Soft-goal Interdependency Graph (SIG). It is a graph of interconnected soft-goals (means goals without clear cut criteria or hard to define) where each soft-goal represents a QR for the software under development. However, the NFR Framework approach suffers from limitations such as (i) It is mostly reliant upon drawing SIGs manually, which is time-consuming and error-prone; (ii) SIG provides an informal description of the goals. Even though graphical representation is suitable for interaction between requirements analysts and end-users, it does not support machine readability. Hence, for large scale software, it is impractical to use such graphical representations.

On the other hand, in the QARM based approach, requirements analysts specify QRs using QARM that represents a pair-wise relationship between QAs. Various QARM based work has been carried out in [5–8]. These QARMs are generic to any system. In addition, as discussed in [12], the advent of emerging technologies introduces new QAs such as *mobility* [13], *context-awareness* [14], *ubiquity* [15] and *invisibility* [16]. However, the limitations in [13–16] are as follows:

1. Usage of the manual process to define QAs and their relationship is tedious and their outcomes rely upon the participant's skills. Also, for each newly discovered QAs, the process needs repetition. Hence, it is difficult to reuse and refine.
2. Lack of information concerning the relationship among newly discovered QAs to the traditional ones such as *security*, *performance*, *usability* among others.

The goal of our research is to build an adaptive QARM semi-automatically, acquiring knowledge from available natural language quality standards, industry documents, and web search engines.

The paper is organized as follows: In Sect. 2, we review the related work. We present the proposed approach in Sect. 3. Then we describe our research methodology in Sect. 4. In Sect. 5, we present and discuss the results of our approach. In Sect. 6, we present threats to validity. Finally, we conclude the paper with the future directions in Sect. 7.

2 Related Work

A large number of quality models have been proposed to make the abstract concept of software quality more tangible [17]. The main purpose of the quality models [18–21] is to decompose quality concepts down to a level where one can measure and evaluates the quality. The quality model is *"the set of quality characteristics and the relationships between them (i.e. QARM) that provides the basis for specifying QRs and evaluation"* [19]. Various QARM based work has been carried out in [5–12]. We broadly classify them into two categories viz. *QA to QA relationship matrix* [5–8] and *QA to functionality relationship matrix* [9–12].

QA to QA relationship matrix represents how one QA undermines (−) or supports (+) achieving other types of QAs. From the literature, we observe four such matrices based

[1] In this work, the term "Interdependency" indicates the relationship among QAs (i.e. how QAs support or limit one or more QAs).

on: (i) Potential conflicts and cooperation relationship among QRs [5]; (ii) Standard quality model ISO/IEC9126 [6]; (iii) Positive and negative relationships among QRs [7]; and (iv) Relative, absolute and never conflicts relationship among QRs [8]. On the other hand, *QA to Functionality relationship matrix* represents how one technique to achieve QA undermines (−) or supports (+) achieving other types of QAs. This matrix was first proposed by the authors in [4]. The authors state that each QA should be specified by the level of operationalizing soft-goals. The soft-goals are the possible design techniques that will help to implement the QA. They can be operations, functions, data, and constraints which may affect other QAs. In Table 1, we provide a comparative study of the existing approaches to generate QARM.

Table 1. Comparative study of the existing approaches to generate QARM

QARM characteristics	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	Our approach
False conflict identification	Y	Y	N	Y	N	N	N	Y	N
Adaptive	N	N	N	N	N	N	N	N	Y
Domain-specific	N	N	N	N	Y	Y	Y	Y	Y
QARM generation process	M	M	M	M	M	M	M	M	SA
QA to QA relationship	Y	Y	Y	Y	N	N	N	Y	Y
QA to functionality relationship	N	N	N	N	Y	Y	Y	N	Y
QA level	H	H	H	H	L	L	H	H	L
QARM representation	Mx	Mx	Mx	Mx	Mx	Mx	Mx	SIG	Ontology

The table contains QARM characteristics such as false conflict identification (**Y**es/**N**o), adaptive matrix (**Y**es/**N**o), domain-specific (**Y**es/**N**o), QARM generation process (**M**anual/Semi-automated (**SA**)), QA to QA relationship (**Y**es/**N**o), QA to functionality relationship (**Y**es/**N**o), QA level (**H**igh/**L**ow), QARM representation (**SIG**/Matrix (**Mx**)/**Ontology**). Our comparative study shows that the relationship between QAs present in [6, 8] is inconsistent. In [6], security and usability have no relationship, while in [8], security and usability have a relative relationship (i.e. These QAs are sometimes in conflict and sometimes not, depending on the application domain and in which context they are used) that leads to the false conflict identification and analysis. In addition, more than 50% of QAs (such as autonomy, productivity, satisfaction) listed in the literature [8] do not have clear definitions and their relationship. Furthermore, non-adaptive QARMs are based on the past literature and industry experience. Hence, recently discovered QAs such as *recoverability, context awareness, mobility, transparency,* and their relationships need to be considered which are not available [12].

3 Proposed Approach

The objective of our proposed approach is to help the requirement analysts to semi-automatically acquire knowledge of the domain-specific QAs (from available quality standards, industry documents and web search engine) and generate QARM (Fig. 1). The approach consists of two modules viz. Acquiring knowledge and QARM generation. Here, we first provide a vector representation of the QAs and QA related documents: Let *QA* be the set of QAs *QA* = *{QA1, QA2....QAn}*. Let *QAD* be the set of QA documents *QAD* = *{D1, D2...Dn}*, where a document *Dn* contains information for the *QAn*. Let *T* be the set of terms extracted from the *Dn* represented by *{t1, t2...tm}*. We discuss these modules in the subsequent subsections.

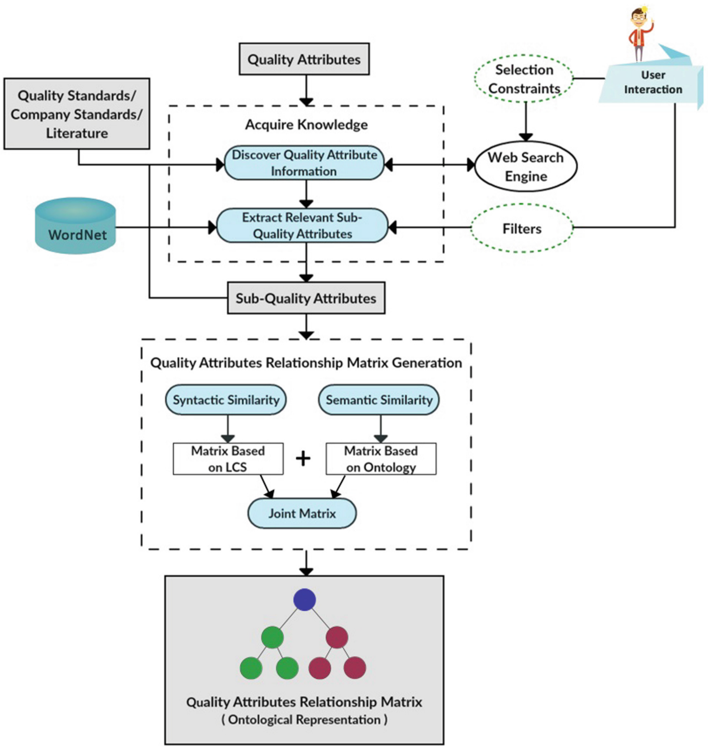
Quality Attributes
Quality Standards/ Company Standards/ Literature
Selection Constraints
User Interaction
Acquire Knowledge
Discover Quality Attribute Information
Web Search Engine
WordNet
Extract Relevant Sub-Quality Attributes
Filters
Sub-Quality Attributes
Quality Attributes Relationship Matrix Generation
Syntactic Similarity
Semantic Similarity
Matrix Based on LCS
+
Matrix Based on Ontology
Joint Matrix
Quality Attributes Relationship Matrix
(Ontological Representation)

Fig. 1. Proposed approach

3.1 Acquire Knowledge

To acquire knowledge, the entire process is divided into two iterative steps. Firstly, we discover QA information from web search engines and available quality standards/documents for a given *QAn* and store it in the *Dn*. Secondly, we extract relevant sub-QAs from the *Dn*. We present the details of each step in the following sub-sections.

3.1.1 Discover QA Information

Definitions and/or descriptions of the QA are the basis that helps us to learn the relevant concepts of the QAs. We analyze a large number of quality documents and websites in order to find the relevant concepts for the QA by searching the QA's definitions. We notice that these definitions on the web differ from each other in some aspects and have similarities in others. Based on our study, we formulate the following questions and search queries in order to discover the information for a given *QAn*:

1. What are the *"Quality_Attribute"* definitions? This question aims to collect definitions available on the web/documents about a *QAn*;

Query 1: *Define* OR *definition* OR *about* OR *what is + Quality_Attribute => Dn*
Query 2: *Quality_Attribute +is/are => Dn*
Query 3: *Quality_Attribute + is/are + called* OR *defined as* OR *known as* OR *refer (s)* OR *described as* OR *formalized as => Dn*
Query 4: *is/are + called* OR *defined as OR known as formalized as + Quality_Attribute => Dn*

2. What are the *"Quality_Attribute"* definitions in *"Specific_Domain"*? This question aims to collect definitions available on web/documents about *QAn* in a specific domain;

Query 5: *Define* OR *definition* OR *about* OR *what is + Quality_Attribute +[in] + Domain_Name => Dn*
Query 6: *Quality_Attribute +[in] + Domain_Name + is/are => Dn*
Query 7: *Quality_Attribute + [in] + Domain_Name + is/are + called* OR *defined as* OR *known as* OR *refer (s)* OR *described as* OR *formalized as => Dn*
Query 8: *is/are + called* OR *defined as* OR *known as formalized as + Quality_Attribute + [in] + Domain_Name => Dn*

3. How is *"Quality_Attribute"* characterized? This question aims to collect existing sub-characteristics for *QAn*;

Query 9: *Feature* OR *Attributes* OR *Characteristics* OR *Matrix + [of] + Quality_Attribute + [in] + Domain_Name) => Dn*

4. What are the details available in *"Software Requirements Specification (SRS)"* to implement *"Quality_Attribute"*? This question aims to identify any kind of solution available in SRS to implement the *QAn*;

Query 10: *Implement* OR *Development [method/scheme]* OR *Execution [details]* OR *Technique* OR *Operationalization* OR *Achieve [of] + Quality_Attribute => Dn*
Query 11: *Function* OR *Procedure* OR *Algorithm* OR *Standard + [for] + Quality_Attribute => Dn*

5. How is the *"Quality_Attribute"* implemented in *"Specific_Domain"*? This question aims to identify any kind of solution used to implement the *QAn* in a specific domain;

Query 12: *Implement* OR *Development [method/scheme]* OR *Execution [details]* OR *Technique* OR *Operationalization* OR *Achieve [of] + Quality_Attribute + [in] + Domain_Name => Dn*

Query 13: *Function* OR *Procedure* OR *Algorithm* OR *Standard + [for] + Quality_Attribute + [in] + Domain_Name => Dn*

3.1.2 Extract Relevant sub-QAs

A detailed analysis of ambiguity in RE [22] shows that dealing with the ambiguity issue at an early stage makes the information accurate. To identify and resolve anaphora ambiguity in *Dn,* we follow the procedure present in [23]. After resolving ambiguity, we extract *Terms* such as adjectives, ending words with -bility, -bilities, -ness, -able, noun and verb phrases from *Dn*. We retain stop-words within noun and verb phrases as they are important for finding hierarchical relationships. To extract sub-QAs, we apply the following filters on *T*:

Filter1: Generic terms such as *data, system, computer, network, system, etc.* from *Dn*
Filter2: Terms that are present in 95% of the QA documents.
Filter3: Terms that violate the minimum size constraints.
Filter4: A stemming algorithm for the English language is used to reject plurals, verbal forms, etc.

For each sub-QA in *SubQAn*, we perform the following analysis to select the relevant sub-QAs for a *QAn*:

(i) Total number of appearances on all the web sites: this is the measure of the importance of the QA concept's to the domain and allows eliminating very specific ones;
(ii) A number of different websites that contain the QA concept at least once: this provides a measure of the generality of the terms for the domain (e.g. interface is quite common, but invisibility is not).

After analyzing the *SubQAn*, for each sub-QA, a new search string is constructed joining the sub-QA with the QA (e.g. *"Transparency (sub-QA) + Invisibility (QA)"*), and the entire procedure executes again. Quality experts will assess and confirm the final SubQAn. The experts can add/update/delete the relevant sub-QAs in *SubQAn*. Each relevant sub-QA is represented in a hierarchy by the searching pattern *"Noun Phrase 1 + [connector] + Noun Phrase 2"*; where the connector can be any combination of *verb phrase/preposition/adjective*. If *Noun Phrase 1 + connector + Noun Phrase 2* then *Noun Phrase 2* is likely to be the subclass of *Noun Phrase 1*. For example, *"Electronic signature (Noun Phrase 1) as (connector) authentication (Noun Phrase 2)"*, *"Biometrics (Noun Phrase 1) as (connector) authentication (Noun Phrase 2)"* indicates that *electronic signatures* and *biometrics* are the subclasses of the *authentication* and both are at the same level in the hierarchy. However, we observe that sometimes the order of the noun phrase may differ. For example, *"Authentication (Noun Phrase 1) via (connector) password (Noun Phrase 2)"*; *"Electronic signature (Noun Phrase 1) as (connector) authentication (Noun Phrase 2)"*. In this case, we check both noun phrases and if we find the exact match in any of the noun phrases, another is considered as sub-classes. We store the hierarchy with a standard representation language: Web Ontology Language (OWL) as follows:

```
<Declaration><Class IRI="#Security"/></Declaration>
<SubClassOf><ClassIRI="#Authentication"/><ClassIRI="#Security"/></SubClassOf>
<SubClassOf><ClassIRI="#Op_Biometric"/><ClassIRI="#Authentication"/></SubClassOf>
<SubClassOf><ClassIRI="#Op_Password"/><ClassIRI="#Authentication"/></SubClassOf>
```

The OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. Moreover, OWL is supported by many ontology visualizers and editors, like Protégé 2.0, allowing the user to explore, understand, analyze or even modify the resulting ontology[2] easily. We present the algorithm to acquire knowledge for a QA as follows:

```
Algorithm 1: Acquire Knowledge
Input: QAn, Quality Standards/Documents, Selection Constraints
Output: SubQAn: Set of relevant Sub-QAs for a QAn
1. for all Qulaity_Doc do
2.    Dn: = Search_Concept (QAn, Quality_Doc)
3. end for
4. List_URLs : = Search_Engine(QAn)
5. for all URL in List_URLs do
6.    Page: = Download_Doc (URL)
7.    Dn: = Search_Concept(QAn, URL.Description)
8.    for all Link in Page do
9.    if Concept_match(QAn, Link.Description) then
10.    List_URLs: = Link
11.    end if
12.   end for
13. end for
14. for each statement (i) in Dn do
15. T: = Noun(Tagword(Dn[i]))UAdjective(Tagword(Dn[i]))U
        Verbs(Tagword(Dn[i])) U Adverb(Tagword(Dn[i)) U Wordsenwith (bility,
        -bilities, -ness) U NounPhrase(Dn[i]) U VerbPhrase(Dn[i])
16. end for
17. SubQAn = Filters(T)
18. Return SubQAn
```

[2] Ontology is a formal description of the concept of sharing, stressing the link between real entities [24]. The ontology helps domain users- to suggest their NFRs effectively and requirements analysts- to understand and model the NFRs accurately. Building ontology based on domain knowledge gives a formal and explicit specification of a shared conceptualization.

3.2 QARM Generation

In this module, we aim to define the relationship between sub-QAs. This procedure takes the input as a *SubQA* from the previous section and produces output as an ontological representation of QARM. The term-based weighting ignores the semantic relation between the SubQA. In order to overcome the weakness of the term-based weighting, in the proposed work we have combined two similarities- string similarity and ontology-based semantic similarity. For string similarity, we use the summation of normalized longest common subsequence (NLCS) measure, normalized maximum consecutive longest common subsequence starting at character 1 (NMCLCS1) and normalized maximum consecutive longest common subsequence starting at any character n (NMCLCSn) presented in [25].

$$\begin{aligned} v1 &\leftarrow NLCS(SubQAi,\ SubQAj) \\ &= len(LCS(SubQAi, SubQAj))\ 2/len(SubQAi) \times len\ (SubQAj); \end{aligned} \tag{1}$$

$$\begin{aligned} v2 &\leftarrow NMCLCS1\ (SubQAi,\ SubQAj) \\ &= len(MCLCS1(SubQAi, SubQAj))2/len(SubQAi) \times len(SubQAj); \end{aligned} \tag{2}$$

$$\begin{aligned} v3 &\leftarrow NMCLCSn(SubQAi, SubQAj) \\ &= len(MCLCSn(SubQAi, SubQAj))2/len(SubQAi) \times len(SubQA); \end{aligned} \tag{3}$$

We take the weighted sum of these individual values *v1, v2*, and *v3* to determine string similarity weight: *wij* = *v1* + *v2* + *v3*. For ontology-based similarity, we use WordNet relatedness measure viz. Lesk [26] algorithm on our extracted knowledge for QAs. The Lesk algorithm [26] works on the concept of identifying relatedness of two terms based on the overlapping of the context of the two terms. The reason for using Lesk on our extracted knowledge is the unavailability of some of the sub-QAs in the existing ontology. For example, if we consider WordNet ontology, sub-QAs such as *login, operability, agility*, etc. are not defined and hence not able to find the similarity of such terms. We utilize the content of the *Dn* for the Lesk algorithm. The normalized QA relatedness:

$$w'ij = Nlesk(lesk(SubQAi, SubQAj)/100) \tag{4}$$

We present the algorithm as follows:

```
Algorithm 2: QARM Generation
Input  : SubQAn
Output: QARM: Quality Attribute Relationship Matrix
1.Construct a string similarity matrix M1 = NLCS + NMCLCS1 +NMCLCSn
2.Construct an ontology-based relatedness (lesk) matrix M2
3.Construct a joint matrix QARM = M1 + M2
4.Return QARM
```

We represent the QARM in the form of triples such as *SubQA1, SubQA2, Relatedness/Relationship*. To determine how different QAs relate to each other, we compare every pair of QAs. By using such pairwise comparisons, it is possible to provide a matrix where the relation between QAs can be decided. To calculate relationship, we determine the percentage of similarity of each respective *SubQA:*

$$QAR = t + y/x \times 100\% \tag{5}$$

Where Q*AR* = percentage of sub-QA similarity; *t* = QARM weight; *y* = *NSyno* and *x* = *NAnto*. We find a number of synonyms and antonyms for each term using ontology WordNet as *NSyno = CountNum(Syno(Wi, Wj) U Hypo(Wi, Wj) U Poly(Wi, Wj))* (Note that we use the hyponyms to identify the relationship between a generic term and a specific instance of it.) and *NAnto = CountAnto(Anto(Wi, Wj)* respectively. Based on the percentage similarity of sub-QA, we define the relationship- (i) Positive (+): when one QA supports other QAs; (ii) Negative (−): when one QA adversely affects other QAs; (iii) Relative (*): when one QA, either supports or adversely affects other QAs.

4 Research Methodology

In order to assess the impact of our approach to construct an adaptive QARM semi-automatically, we conducted a controlled experiment on 18 QAs. We followed the guidelines provided in [27].

4.1 Research Questions

The following Research Questions (RQs) are established to evaluate our approach.

RQ1: *How can we semi-automatically extract the sub-QAs for a given QA?* The accuracy of our approach is partly driven by the sub-QAs extracted. With RQ1, we examine whether our approach can find the information for the QAs semi-automatically from available quality standards, industry documents and web search engines. Also, we analyze and compare the accuracy of the resulting sub-QAs.
RQ2: *To what extent the generated QARM can be useful to requirements analysts?* The overall goal of our experiment is to construct the QARM. In RQ2, we assess the accuracy of the generated QARM.

Viewing the RQs from an industry perspective, the questions would focus on how the requirements analysts semi-automatically discover the sub-QAs for newly discovered QA and perceive the relations among them. For example, the requirements analysts need to spend considerable time and non-trivial human effort to acquire knowledge for QAs and influence among them during the process of QRs conflict identification. This process involves the manual search and analysis of a large set of quality documents. Our approach parses these documents, extracts sub-QAs and constructs the QARM semi-automatically at requirements engineering phase. However, the requirements analysts should be aware of the QAs that are concerned with the system being developed. In addition, the approach needs an analyst to assess the correctness of the sub-QAs and QARM.

4.2 Experimental Setup

Our proposed approach has been developed in Java, due to the availability of a large number of libraries that facilitate the retrieval and parsing of web pages and the construction of ontologies. We use the following supporting tools/API:

1. Stanford-core NLP: It provides a wide range of algorithms for natural language processing, e.g. Stemming, PosTag, stop-word removal, etc.
2. Java API for WordNet Searching (JAWS): JAWS is a Java API for searching WordNet. It helps retrieve synonyms, hyponyms, etc. very easily.
3. jsoup: This HTML parser helps to fetch results from Google search and parse the html text.
4. Crawler4j: This is a web crawler API, that helps to crawl the websites retrieved from Google search.

To answer the RQs, we select 18 QAs viz. *Security, usability, performance, reliability, understandability, portability, maintainability, flexibility, supportability, testability, suitability, manageability*, *reusability*, *agility, mobility, ubiquity, invisibility,* and *enhanceability*. In our experiment, we classify these QAs into three categories:

(i) *Traditional QAs:* QAs that are well defined in the literature and their relationship details are available [5–8] such as *security, usability, performance, reliability, understandability,* and *portability*. With *traditional QAs*, we aim to validate the accuracy of the proposed approach with respect to the available literature.
(ii) *Traditional QAs but Lack of Relationship Details*: QAs that are well defined in the literature, but their relationship details are missing such as *maintainability, flexibility, supportability, testability, suitability, manageability,* and *reusability*. With this category of QAs, we aim to discover and analyze the relationship details that are missing in the literature [8];
(iii) *Emergent QAs*: QAs that are not yet systematically defined and their relationship details are missing [11, 12] such as *agility, mobility, ubiquity, invisibility,* and *enhanceability.* With *emergent QAs*, we aim to discover the sub-QAs for the recently discovered QAs and their relationships with the traditional QAs.

For experimental analysis, we use the following evaluation parameters:

(i) Recall = TP/(TP + FN) *100;
(ii) Precision = TP/(TP + FP) *100;
(iii) F-measure = 2 *((precision * recall)/(precision + recall));

Where, TP = True Positive (Number of correctly identified sub-QAs), FP = False Positive (Number of incorrectly identified sub-QAs), and FN = False Negative (Number of sub-QAs incorrectly not identified).

4.3 Data Gathering and Analysis

To answer **RQ1**, we analyze quality standards and company documents that provide the definitions, characteristics, sub-QAs, and its implementation details. Moreover, we collect relevant concepts for the QAs, from the Google search. During our experimental study, we observed that some data are misleading, uninterpretable and required additional processing. We manually analyze the received data and define the following constraints:

(i) According to our findings, the web search engine's initial pages are more comprehensive and important. Therefore, the maximum number of websites per search is limited to the first hundred results returned by the Google for our experimentation purpose.
(ii) We considered the maximum depth level (redirections) for finding the relevant terms.
(iii) We found from our experimental results that the minimum number of characters for a sub-QA should be at least four characters. Therefore, we have considered Sub-QAs with a minimum of four characters. The partial results of our experiments are available online[3]
(iv) In order to avoid processing irrelevant terms, the maximum number of results returned by the Google for each new sub-QA is set up to 1200 terms.
(v) Different types of non-HTML document formats (.pdf and.doc) are processed by obtaining the HTML version from the Google's cache. Furthermore, through analyzing each sub-frame, we considered frame-based sites to obtain the complete set of texts.

We process the document in a text format. We extract adjective, noun, adverb, verb, noun phrase and verb phrase from the document. We retain stop-words within noun and verb phrases as they are important to find hierarchical relationships. From our experiment, we apply four filters as stated in Sect. 3.1. Our approach is based on the textual information available on the search engine and cannot deal with the information contained in the images and tables. To answer **RQ2**, we experimentally analyzed the semantic similarity of the collected sub-QAs. We define the range of positive, negative and relative relationships among QAs as shown in Table 2.

Table 2. Degree of relatedness based on semantic similarity

Percentage of similarity	Relationship
$45\% \leq \text{QAR} \leq 100\%$	Positive (+)
$31\% \leq \text{QAR} \leq 44\%$	Relative (*)
$\text{QAR} \leq 30\%$	Negative (−)

[3] https://github.com/UnnatiS/QARM-Generation/tesoutputfile1.txt.

5 Results and Discussion

In this section, we present and discuss the RQs based on the results of our approach.

5.1 Results

RQ1: In Table 3, we provide the results of the sub-QAs extracted semi-automatically for the 18 selected QAs. To validate the proposed approach, we first evaluate the accuracy of the sub-QAs collected (Table 3) for the *traditional QAs* by comparing them with the existing literature. We discover a total of 15 sub-QAs for the QA *performance*, where 12 sub-QAs viz. *Responsiveness, Latency, Throughput, Space, Capability, Execution rate, Delay, Loss, Consequence effect, Usage, Activeness, Resourcefulness* are matched with [8] and 3 sub-QAs viz. *Serviceability, Measurability* and *Reaction time* are new-found. We observed that in the literature [5–12], some sub-QAs are incorrect or redundant. For instance, in [8], the authors state 19 QAs viz. *Response time, Space, Capacity, Latency, Throughput, Computation, Execution speed, Transit delay, Workload, Resource utilization, Memory usage, Accuracy, Efficiency, Compliance, Modes, Delay, Miss rates, Data loss, Concurrent transaction Processing* for the QA *performance*.

However, QAs such as *"resource utilization and memory usage"*; *"transit delay and delay"*; *"computation, Execution speed and concurrent transaction processing"*;

Table 3. Experimental results of extracted sub-QAs

Category for Evaluation	QAs	Sub-Qas
Traditional QAs	Security **(S)**	Confidentiality, Integrity, Availability, Authentication, Access control, Privacy, Protection, Prevention, Reliability, Safety
	Performance **(P)**	Responsiveness, Latency, Throughput, Serviceability, Measurability, Capability, Resourcefulness, Space, Execution, rate, Reaction time, Scalability, Activeness
	Usability **(U)**	Simplicity, Understandability, Comfort, Informality, Operability, Ease of use, User service, Potentiality, Attractiveness, Likeliness, Accessibility, Familiarity
	Reliability **(R)**	Portability, Performance, Availability, Maturity, Accuracy, Precision, Recoverability, Operability, Maintainability, Consistency, Correctness, Satisfactorily, Dependability, Completeness, Robustness, Integrity
	Portability **(Po)**	Transferability, Changeability, Channelize, Transpose, Adjustability, Reusability, Interoperability, Extensibility, Dependability
	Understandability **(Un)**	Readability, Predictability, Organization of the document, Essential, Complexity, Verifiability, Visibility, Clarity, Controllability, Measurability, Reviewability

(*continued*)

Table 3. (*continued*)

Category for Evaluation	QAs	Sub-Qas
Traditional QAs but Lack of Relationship Details	Maintainability **(M)**	Operability, Upgrade, Performance, Evolution, Cost, Changeability, Analysability
	Suitability **(Su)**	Appropriateness, Usefulness, Readiness, Interrelate, Correctness
	Testability **(T)**	Ease, Validity, Examination
	Flexibility **(F)**	Ease, Interaction, Interface, Modifiability, Changeability, Maintainability, Adaptable, Simplicity
	Manageability **(Ma)**	Performance, Monitoring, Flexibility, Tractability
	Supportability **(Sp)**	Sustenance, Accompaniment, Reinforcement, Substantiate
	Reusability **(Ru)**	Probability, Changeability, Efficiency, Interchange, Evolution
Emergent QAs	Invisibility **(In)**	Transparency, Obviousness, Clearness, Diffusion, Interaction
	Enhancability **(En)**	Portability, Scalability, Flexibility, Evolution
	Ubiquity **(Ub)**	Attention, Safety, Privacy, Robust, Mobility, Efficiency, Inexpensive, Complexity, Context-aware, Transparency, Invisibility, Calmness, Availability
	Mobility **(Mo)**	Adaptability, Flexibility, Ubiquity, Portability, Multispace, support, Connectivity, Integrity, Availability, Motility, Movability, Manipulability
	Agility **(Ag)**	Changeability, Flexibility, Quickly, Operability, Easiness, Simplicity, Comfortness, Informality

"Response time and processing" are redundant. Furthermore, an attribute *"mode"* is not a valid sub-QA for the QA *performance*. To evaluate the extracted sub-QAs, we consider TP as sub-QAs that are matched with the literature (after removing redundant sub-QAs from literature); FN as sub-QAs that are newly discovered by our approach; FP as sub-QAs that are not identified by our approach. For instance, in [8], the authors stated a total of 19 sub-QAs for QA *performance*. From which 2 sub-QAs are redundant and 1 sub-QA is falsely considered. In our experiment, we identified a total of 15 sub-QAs for QA *performance*. From which 12 sub-QAs are matched with [8], 1 sub-QA is falsely considered and 3 sub-QAs are new-found. Based on these details, we calculate recall and precision for QA *performance* as **Recall** = Sub-QAs matched with the literature/(Sub-QAs matched with the literature + Sub-QAs newly discover by our approach) *100; Recall = 12/(12 + 3) = 80%; **Precision** = Sub-QAs matched with the literature)/(Sub-QAs matched with the literature + Sub-QAs our approach considered falsely))*100;

Precision = 12/(12 + 4) = 75%. In Fig. 2, we provide a detailed analysis of the discovered sub-QAs with respect to the existing literature.

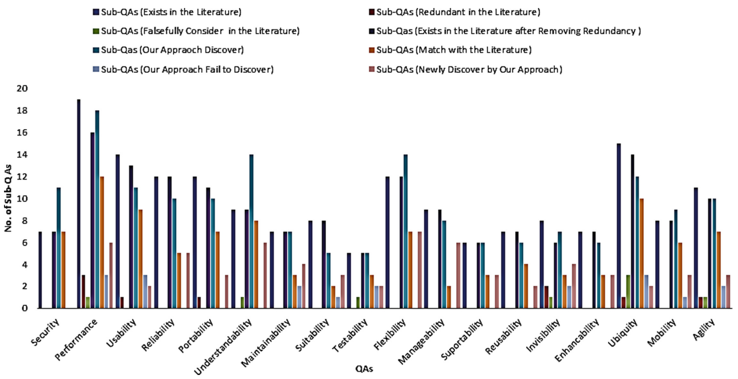

Fig. 2. Detailed analysis of the discovered sub-QAs

RQ2: In Table 4, we present the adaptive QARM generated semi-automatically after analyzing the QAs's taxonomy. The ontological representation of the constructed QARM is available online[4]. The constructed matrix (Table 4) extends and complements previously published QARM [8–11].

5.2 Discussion

The strength of the approach lies in its ability to quickly trawl through quality documents and web search engines to discover sub-QAs and to construct QARM. Even though the approach still requires an analyst to evaluate the correctness of the sub-QAs and QARM, it requires less manual effort than the approach discussed in [8–12].

An interesting observation of the results of our approach is how many sub-QAs we discovered semi-automatically compared to manual analysis. We discovered that the proposed approach achieves an average 85.67% recall, 59.07% precision and 69.14% F-measure to semi-automatically discover sub-QAs. We also observed that 38.88% (7/18) of the cases, the top 10–20 pages contain the relevant information and the top 20–50 pages contain the relevant information in 61.11% (11/18) of the cases. Furthermore, we found that 2.86% sub-QAs are falsely considered and 4.31% sub-QAs are redundant in the literature [5–8] on average. We discovered an average of 4% new sub-QAs for a given QA.

We constructed the QARM with positive, negative and relative relationships. According to existing literature [8–11], the resulting QARM is accurate. This QARM can be

[4] https://github.com/UnnatiS/QARM-Generation/.

Table 4. An adaptive quality attribute relationship matrix

QAs	S	P	U	R	Po	Un	M	Su	T	F	Ma	Sp	Ru	In	En	Web	Mo	Ag
S		*	*	+	–	*	–	*	*	–	+	*	–	–	*	*	–	–
P	*		*	–	+	+	–	+	–	–	–	–	*	–	*	–	–	–
U	*	*		*	+	+	*	+	*	+	–	–	+	*	–	+	*	*
R	+	–	*		–	*	–	+	+	+	*	*	–		+			
Po	–	+	+	–		*	–	*	+	+	–	+	+	–	+	+	+	–
Un	*	+	+	*	*		*	*	+	+	*	*	*	+	+	+	*	+
M	–	–	*	–	–	*		+	*	+	+	+	–	–	*	–	–	*
Su	*	+	+	+	*	*	+		+	–	*	*	*	+	+	+	*	–
T	*	–	*	+	+	+	*	+		–	–	*	*	–	*	–	–	–
F	–	–	+	+	+	+	+	–	–		–	*	–	+	–	+	+	+
Ma	+	–	–	*	–	*	+	*	–	–		*	*	–	+	*	–	–
Sp	*	–	–	*	+	*	+	*	*	*	*		*	+	+	*	+	+
Ru	–	*	+	–	+	*	–	*	*	–	*	*		–	+	–	+	+
In	–	–	*	*	–	+	–	+	–	+	–	+	–		*	+	+	*
En	*	*	–	+	+	+	*	+	*	–	+	+	+	*	*	+	+	+
Ub	*	–	+	–	+	+	–	+	–	+	*	*	–	+	+		+	*
Mo	–	–	*	–	+	*	–	*	–	+	–	+	+	+	+	+		+
Ag	–	–	*	–	–	+	*	–	–	+	–	+	+	*	+	*	+	

used to identify the conflicts among QRs in various software development phases. For instance, in the requirements engineering phase, during the elicitation process, system analysts would be able to identify the relationship among QRs. This analysis would allow developers to identify the conflicts among QRs early and to discuss this potential conflict with the system's stakeholders before specifying the software requirements. In addition, during the architecture design process, system designers would also be able to use the QARM to analyze the potential conflict among QRs in terms of the architecture decision. The relative relationship among QAs presented in the QARM would allow system designers to investigate the potential architecture strategies to get the best solution based on the type of conflicts among QRs. The proposed approach can be applied to the project management process when the project manager predicts QRs conflicts before implementing the system and then adjusts manpower, time, or cost-effectively and efficiently.

6 Threats to Validity

Even though we successfully construct QARM, we observe the following threats to the validity of our approach. A first possible threat is about the validity of the knowledge discovered from the search engine for the QAs, as the inaccuracy of the information leads to the false QARM generation. To mitigate this threat, we perform the experiments on 18 QAs. The experimental datasets and QARMs generated by the approach are manually analyzed by the authors. However, our evaluation might differ from an industry perspective because we use clear-cut definitions available on quality documents/search engine to identify QAs and influence among them, while industry relies on experience and implementation details. To mitigate this threat, the resultant sub-QAs and QARM are analyzed by three industry experts that are active in the field of software quality assurance. The industry experts observed that the approach identifies an average of 9.57% false relationships among QAs. For instance, we discovered a relative relationship between QAs *usability* and *manageability*. The industry experts argued that *"If the system is user-friendly then managing it will also be easy. Hence, QA usability has a positive relationship with QA manageability*". In addition, our approach is based on textual information available on the search engine, for this reason, relevant information may be left undetected in several cases.

7 Conclusions

In this paper, we present an approach that semi-automatically constructs QARM, discovering knowledge from available quality standards/literature and search engines. This work helps the requirements analyst to visualize the relationship among QAs by means of the ontology. We evaluate the approach on 18 QAs and achieve 69.14% F-measure to extract sub-QAs. Also, we discover 4% of new sub-QAs that are not defined in the existing literature. Furthermore, we achieve 60% precision to identify the relationship between QAs. In the future, we aim to utilize the knowledge regarding operations through which QAs are to be achieved to enhance the QARM. The other direction of work we intend to pursue is to tackle the diverse meaning of the quality attributes such as *confidentiality* or *privacy*, *stability* or *robustness*, *fault-tolerance* or *resilience*, *flexibility* or *adaptability*, *evolution* or *adaptability*, *sustainability* or *durability*, *clarity* or *understandability*, *reasonability* or *predictability* that creates ambiguity using them in practice.

References

1. IEEE Computer Society, Software Engineering Standards Committee, and IEEE-SA Standards Board: IEEE recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers (1998)
2. Shah, U.S., Patel, S., Jinwala, D.: Specification of non-functional requirements: a hybrid approach. In: REFSQ Workshops (2016)
3. Guizzardi, R.S.S., Li, F.-L., Borgida, A., Guizzardi, G., Horkoff, J., Mylopoulos, J.: An ontological interpretation of non-functional requirements. In: FOIS, vol. 14, pp. 344–357 (2014)

4. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering, 5th edn. Springer, Heidelberg (2012)
5. Egyed, A., Grunbacher, P.: Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help. IEEE Softw. **21**(6), 50–58 (2004)
6. ISO/IEC 9126-1:2001 Software engineering product quality-part 1: quality model. International Organization for Standardization (2001)
7. Duque-Ramos, A., Fernández-Breis, J.T., Stevens, R., Aussenac-Gilles, N.: SQuaRE: A SQuaRE-based approach for evaluating the quality of ontologies. J. Res. Pract. Inf. Technol. **43**(2), 159 (2011)
8. Mairiza, D., Zowghi, D., Nurmuliani, N.: Managing conflicts among non-functional requirements. In: Workshop on Requirements Engineering, pp. 11–19. University of Technology, Sydney (2009)
9. Sadana, V., Liu, XF.: Analysis of conflicts among non-functional requirements using integrated analysis of functional and non-functional requirements. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 1, pp. 215–218. IEEE (2007)
10. Abdul, H., Jamil, A., Imran, U.: Conflicts identification among non-functional requirements using matrix maps. World Acad. Sci. Eng. Technol. **44**, 1004–1009 (2010)
11. Mairiza, D., Zowghi, D., Gervasi, V.: Conflict characterization and analysis of non functional requirements: an experimental approach. In: 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT), September 2013, pp. 83–91. IEEE (2013)
12. Carvalho, R., Andrade, R., Oliveira, K., Kolski, C.: Catalogue of invisibility requirements for UbiComp and IoT applications. In: 26th International Requirements Engineering Conference (RE), pp. 88–99. IEEE (2018)
13. Maia, M.E., Rocha, L.S., Andrade, R.: Requirements and challenges for building service-oriented pervasive middleware. In: Proceedings of the 2009 International Conference on Pervasive Services, pp. 93–102. ACM (2009)
14. Carvalho, R.M., de Castro Andrade, R.M., de Oliveira, K.M.: AQUArIUM - a suite of software measures for HCI quality evaluation of ubiquitous mobile applications. J. Syst. Softw. **136**, 101–136 (2018)
15. Serrano, M.: Ubiquitous, pervasive and mobile computing: a reusable-models-based non-functional catalogue objectives of research. In: ER@ BR (2013)
16. Carvalho, R.M., de Castro Andrade, R.M., de Oliveira, K.M., de Sousa Santos, I., Bezerra, C.I.M.: Quality characteristics and measures for human-computer interaction evaluation in ubiquitous systems. Softw. Q. **25**(3), 743–795 (2017). https://doi.org/10.1007/s11219-016-9320-z
17. Miguel, J.P., Mauricio, D., Rodríguez, G.: A review of software quality models for the evaluation of software products. Int. J. Softw. Eng. Appl. **5**(6), 31–53 (2014)
18. Boehm, B.W., Brown, J.R., Kaspar, H.: Characteristics of Software Quality. North Holland, Amsterdam (1978)
19. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality. General Electric Co., Sunnyvale (1977)
20. Grady, R.B., Caswell, D.L.: Software Metrics: Establishing a Company-Wide Program. Prentice Hall, Upper Saddle River (1987)
21. Dromey, R.G.: A model for software product quality. IEEE Trans. Softw. Eng. **21**(2), 146–162 (1995)
22. Shah, U.S., Jinwala, D.C.: Resolving ambiguities in natural language software requirements: a comprehensive survey. ACM SIGSOFT Softw. Eng. Notes **40**(5), 1–7 (2015)

23. Shah, U.S., Jinwala, D.C.: Resolving ambiguity in natural language specification to generate UML diagrams for requirements specification. Int. J. Softw. Eng. Technol. Appl. **1**(2–4), 308–334 (2015)
24. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
25. Islam, A., Inkpen, D.: Semantic text similarity using corpus-based word similarity and string similarity. ACM Trans. Knowl. Discov. Data **2**(2), 1–25 (2008). Article No. 10
26. Banerjee, S., Pedersen, T.: An adapted Lesk algorithm for word sense disambiguation using WordNet. In: Gelbukh, A. (ed.) CICLing 2002. LNCS, vol. 2276, pp. 136–145. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45715-1_11
27. Jedlitschka, A., Ciolkowski, M., Pfahl, D.: Reporting experiments in software engineering. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, pp. 201–228. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_8