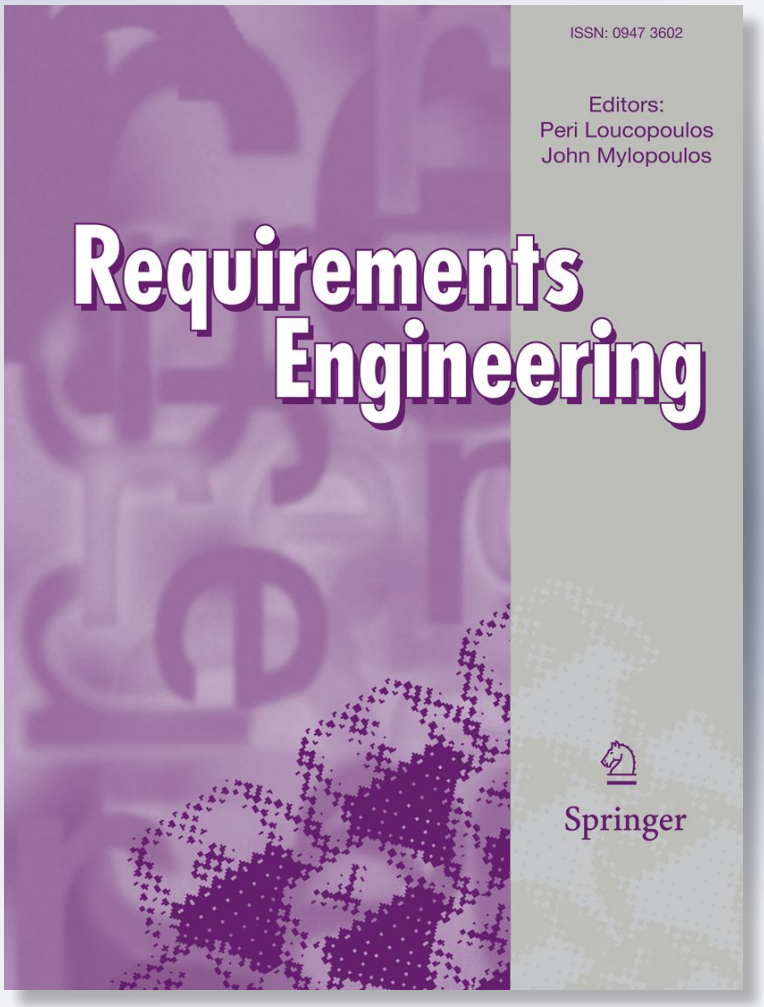
Uncovering quality-attribute concerns in use case specifications via early aspect mining

Alejandro Rago, Claudia Marcos & J. Andrés Diaz-Pace

Requirements Engineering

ISSN 0947-3602

Requirements Eng
DOI 10.1007/s00766-011-0142-z



Springer

Requirements Eng
DOI 10.1007/s00766-011-0142-z

ORIGINAL ARTICLE

Uncovering quality-attribute concerns in use case specifications via early aspect mining

Alejandro Rago · Claudia Marcos · J. Andrés Diaz-Pace

Received: 10 August 2011 / Accepted: 19 November 2011

Abstract Quality-attribute requirements describe constraints on the development and behavior of a software system, and their satisfaction is key for the success of a software project. Detecting and analyzing quality attributes in early development stages provides insights for system design, reduces risks, and ultimately improves the developers' understanding of the system. A common problem, however, is that quality-attribute information tends to be understated in requirements specifications and scattered across several documents. Thus, making the quality attributes first-class citizens becomes usually a time-consuming task for analysts. Recent developments have made it possible to mine concerns semi-automatically from textual documents. Leveraging on these ideas, we present a semi-automated approach to identify latent quality attributes that works in two stages. First, a mining tool extracts early aspects from use cases, and then these aspects are processed to derive candidate quality attributes. This derivation is based on an ontology of quality-attribute scenarios. We have built a prototype tool called QAMiner to implement our approach. The evaluation of this tool in two case studies from the literature has shown interesting results. As main contribution, we argue that our approach can help analysts to skim requirements documents and quickly produce a list of potential quality attributes for the system.

Keywords Quality attribute · Early aspect · Use case specification · Text mining · Tool support

A. Rago (✉) · C. Marcos · J. A. Diaz-Pace
ISISTAN Research Institute, UNICEN University,
Paraje Arroyo Seco, Tandil, Argentina
e-mail: arago@exa.unicen.edu.ar

C. Marcos
e-mail: cmarcos@exa.unicen.edu.ar

J. A. Diaz-Pace
e-mail: adiaz@exa.unicen.edu.ar

A. Rago · J. A. Diaz-Pace
CONICET, Buenos Aires, Argentina

C. Marcos
CIC, Buenos Aires, Argentina

1 Introduction

Requirements engineering (RE) is an important discipline in any software development, as it helps to ensure that the system specification fulfills the stakeholders' needs and, ultimately, that the right software product will be built. Core activities in RE include elicitation, modeling, analysis, communication, and negotiation of requirements, among others [31]. Mistakes or not enough attention to RE activities cause problems and re-work in later development activities such as architectural design, detailed design, and implementation. In particular, identifying *quality attributes* (e.g., modifiability, performance, availability, etc.) relevant to the stakeholders and eliciting quality-attribute requirements (e.g., in the form of quality-attribute scenarios) both play a key role in determining the software architecture of the system [5]. Typical sources of quality-attribute requirements are business goals, domain-specific concerns, and environmental constraints.

Unfortunately, quality-attribute requirements tend to be understated in requirements specifications, because usually the focus is primarily on functionality [10]. At best, quality-attribute concerns are informally captured and often appear scattered through several specifications. For example, a use case can qualify a given piece of functionality with words such as "fast" or "with little delay" to indicate

Published online: 21 December 2011

a performance concern. Related performance comments might be hinted in another use case; but the fragmentation precludes a consistent analysis of the performance concern. Because of these problems, quality-attribute requirements are often discovered late in the development process, and the changes to accommodate those requirements can be costly to make. For example, a key security concern being inadvertently skipped in a use case can lead to a wrong implementation; and adding the necessary security mechanisms to a solution as an afterthought can be difficult. It is often up to the analyst to glean the main quality attributes from different documents and workshops with stakeholders. In this context, a very helpful technique is to have the analyst skim a number of relevant documents and come up with a list of candidate quality attributes, which she can later discuss and refine with the stakeholders. However, requirements tools have not addressed yet this need.

On the other hand, it has been argued [6] that the identification of the so-called *early aspects* (also known as early crosscutting concerns) can significantly help to analyze and plan for design tradeoffs early in the lifecycle. Early aspects are typically detected in requirements specifications. An aspect at the requirements level can be seen a broadly scoped property that affects (i.e., crosscuts) multiple other requirements [1, 36]. Examples of early aspects commonly found in requirements are security, mobility, real-time constraints, etc. Over the last years, several semi-automated techniques for mining early aspects from textual specifications have been developed [23, 37, 39]. Interestingly, some early aspects actually derive into quality-attribute concerns [3, 6] (but not all early aspects will have quality-attribute connotations). For example, the analysis of an early aspect about real-time constraints can reveal a performance concern. Along this line, we argue that the identification of quality attributes in requirements specifications can be informed by the early aspects detected in those specifications. That is, a hidden quality-attribute concern can surface first as an early aspect (e.g., via a mining tool), and then an analyst can check its relevance from a quality-attribute perspective.

In previous work, we developed an aspect mining tool called semantic aspect extractor tool (SAET) [34, 35] that detects potential crosscutting concerns semi-automatically from use case specifications. This tool relies on several techniques such as natural language processing (NLP), word sense disambiguation (WSD), and semantic clustering, among others. In this article, we present a tool approach called *QAMiner (Quality-Attribute Miner)* that helps analysts to extract potential quality attributes from use cases, based on the early aspects resulting from SAET. Basically, QAMiner works in two stages: the first stage takes a set of use cases and asks SAET to generate a list of early aspects with crosscutting relations to the use cases; and the second stage processes the outputs of SAET using a predefined quality-attribute ontology in order to derive a list of candidate quality attributes. QAMiner goes through the early aspects and looks for words that match the concepts of a given quality attribute within the ontology. Those quality attributes that receive more matchings are outputted as candidate quality attributes for the system. The contributions of this work are two-fold. First, we propose a novel use of early aspects as "hints" for latent quality attributes in use cases. Second, we provide semi-automated support for the approach and present an evaluation of QAMiner with two case studies from the literature.

The rest of the article is organized into 5 sections. Section 2 provides background information about quality attributes and early aspects. Section 3 describes the steps of the QAMiner approach, based on a motivating example. Section 4 presents the results of a preliminary evaluation of QAMiner. Section 5 discusses related work. Finally, Sect. 6 gives the conclusions and future work.

2 Background

The development of a software system begins with a specification of the high-level functional requirements as well as any operational constraints that the system should satisfy [31]. These artifacts usually serve to derive a software architecture for the system. In addition to functionality, it has been long recognized that *quality attributes* must be clearly specified as part of requirements specifications [5]. Quality-attribute examples include performance, security, availability, modifiability, testability, etc. Since the definition of a quality attribute is often non-operational, it is customary to characterize a quality-attribute by means of *quality-attribute scenarios*. In some domains, it is also possible to leverage on domain-specific knowledge to define quality-attribute concepts [9, 12]. From an analyst's perspective, the elicitation of quality attributes (and corresponding scenarios) as early as possible is very important, because they will inform subsequent design decisions and help to manage quality-attribute tradeoffs.

Having a good separation of concerns in requirements specifications is also important for an analyst. For instance, if the same concept appears again and again in several use cases, that concept should be factored out and dealt with separately. A recurring concept in requirements is called *early aspect* (or requirement-level aspect), and it is said to *crosscut* specific requirements. For example, let's assume that the system functionality is partitioned into a set of use cases, as illustrated in Fig. 1. The system in this diagram is an excerpt from the health watcher system (HWS) [18]. A *Persistence* aspect crosscuts several use cases dealing with

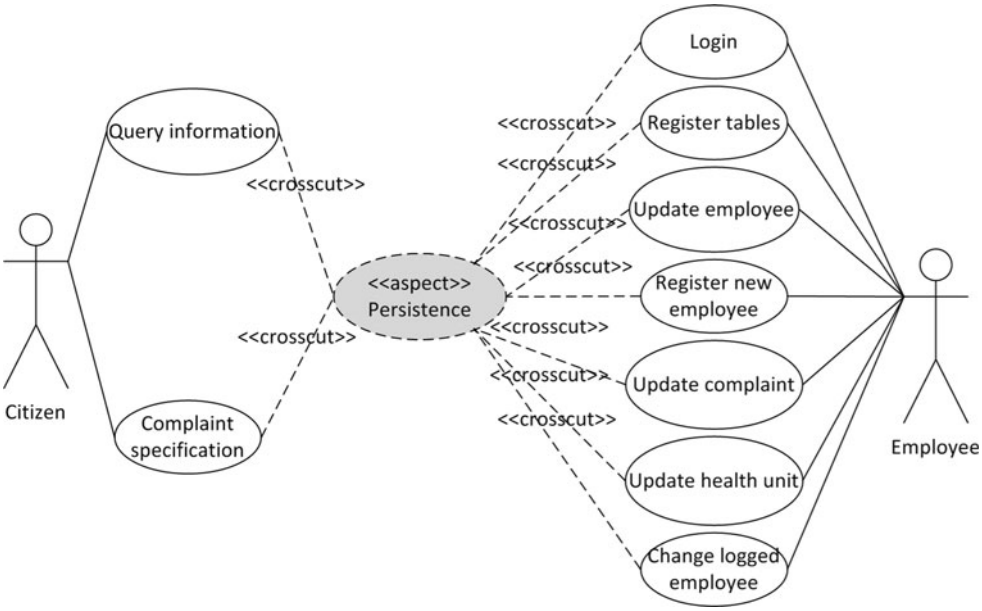


Fig. 1 Example use case diagram

data storage/retrieval, so that this functionality should be segregated to a single artifact [24].

More formally, an early aspect is a broadly scoped property represented by a single requirement or a coherent set of requirements, which affects multiple other requirements in the system [1]. The crosscutting can take two forms as follows: (1) it constrains the specified behavior of the affected requirements; or (2) it influences the affected requirements altering their specified behavior. Coming back to our example, the *Persistence* aspect is actually a requirement that constraints the use cases to a consistent way of handling data. The aspect might also influence the behavior of these use cases by imposing data encryption mechanisms.

When analyzing functional requirements, we believe that quality attributes and early aspects are related to one another, as shown in Fig. 2. On one side, crosscutting concerns are scattered or tangled as functional requirements, making it difficult to understand and design for those requirements. To alleviate this problem, crosscutting concerns can be encapsulated into early aspects. On the other side, functional requirements are often affected by quality-attribute properties. Quality-attribute properties are not always clearly elaborated when being part of functional requirements. For example, saying just that a system "must be secure" does not contribute to the design of a secure architecture. The problem is that "security" is too general and its meaning as a quality attribute cannot be discerned. Bass et al. [5] have proposed predefined quality-attribute scenarios to properly characterize quality attributes. Interestingly, some researchers have discussed how early aspects contribute to the identification of quality attributes [6, 20, 30]. In general, the relationship between early

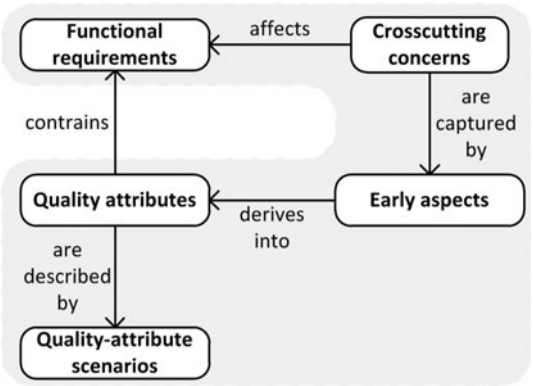


Fig. 2 Relationship among requirements, quality attributes, and early aspects

aspects and quality attributes is one-to-many, but this relationship is not mandatory. In the example of Fig. 1, the *Persistence* aspect is mainly associated with a modifiability quality attribute. Alternatively, the same aspect could point to other quality attributes such as security or performance. Note that the derivation of quality attributes is not straightforward for the analyst, as it depends on the context in which the early aspect is used within the use cases.

2.1 Mining early aspects from textual specifications

Although good requirements specifications are desirable [21],[1] analysts have to deal with the fact that most

[1] Requirements are an important factor in project success or failure, but there are also other factors that come into play (e.g., lack of stakeholder involvement, incorrect environmental assumptions, communications failures within development teams, inadequate conflict management, etc.).

requirements documents are written in textual form, with little or no structure. Some notations such as use cases provide structured templates, but filling out the templates with the right text is still the analyst's responsibility. A survey by Luisa et al. [28] about RE processes in software companies found that approximately 80% of software requirements are written in natural language. In the remaining 20%, nearly 16% were organized in templates but again specified in a non-formal manner. This means that roughly 96% of the requirements are documented in natural language. A clear advantage of using natural language (over sophisticated notations) for capturing requirements is that the communication with stakeholders is improved. On the downside, processing documents written in natural language can be a time-consuming and error-prone activity for the analyst, particularly in the case of large documents. In this context, tools able to extract (or summarize) relevant topics from text are of great help for the analyst's work. Recent advances in NLP and text mining techniques have enabled semi-automated assistance for requirements elicitation and modeling [26]. It is possible to mine various kinds of information, such as domain entities, particular behaviors, potential crosscutting concerns, among others.

2.2 Semantic aspect extractor tool

In previous work [35], we developed a tool called SAET that mines early aspects from use cases. SAET plays an important role in our approach, because the early aspects emitted by this tool provide clues for identifying latent quality-attribute concerns, as we will explain in the next section. At the core of SAET, there are text processing techniques, such as NLP [29] and WSD [33], which look at the textual sections of the use case template and apply a semantic analysis in order to find concerns that crosscut two or more use cases. These concerns constitute candidate early aspects for the use cases. Each early aspect is characterized by a set of *descriptors*, which are pairs of verbs and direct objects. A descriptor essentially represents a particular system behavior, so that a set (of descriptors) contains semantically related behaviors. For instance, the *Persistence* aspect in Fig. 1 was actually suggested by SAET after processing the accompanying use cases. This aspect has descriptors such as: <retrieves, list>, <saves, data>, <search, repository>, <store, information>, <rolled-back, changes>, <updated, complaint>, among others.

Internally, SAET performs several text analysis steps. The first step is a NLP of the textual use cases, for detecting sentences and word boundaries, as well as for recognizing lexical, syntactical, and semantic properties of these words, such as part-of-speech tags. To recognize lexical and syntactical units, the tool uses the OpenNLP[2] toolkit. In addition, the tool uses WordNet[3] as a sense inventory for detecting the semantic meaning of each word. Specifically, we take advantage of semantic similarity measures (e.g., Lesk Overlaps [27, 32] and Extended Lesk Overlaps [32]) and of a WSD algorithm (maximum relatedness disambiguation [33]). Afterward, the second step detects the main verbs and their direct objects in each sentence. These descriptors are arranged in a data structure known as action-oriented identification graph (AOIG) [38], which we have extended to support clustering. Figure 3 schematically depicts the relationships among the use cases, the AOIG, the clusters, and the resulting early aspects. The clustering process is based on the similarity measures and the WSD algorithm above. The resulting clusters are actually the early aspects. At last, the third step transverses the AOIG and ranks the early aspects according to different criteria (e.g., crosscutting rate, cluster statistical weight, among others). Some clusters might be filtered out, if they are considered not relevant. A final ranking is presented to the analyst as a list of candidate early aspects. To facilitate the understanding of its suggestions, the tool also generates traceability links from the aspects to the original use cases. The results of applying SAET to case studies have shown the potential of the semantic analysis techniques.

3 The QAMiner approach

The idea behind our approach, called QAMiner, is to provide support to identify quality attributes from requirements specifications in a semi-automated fashion. QAMiner is intended to assist analysts in dealing with quality-attribute requirements and reasoning about their impact on the product, at early stages of software development. As inputs, the approach takes functional requirements specified by use cases and early aspects previously identified with the SAET tool. Both the use cases and the early aspects come in textual form. Each early aspect is linked to one or more use cases, i.e., an early aspect crosscuts use cases. The core of QAMiner is the analysis of the descriptors representing each early aspect and the words from its related use cases, in order to infer quality attributes that might be involved in the crosscutting concerns.

The QAMiner tool approach works in two stages, as shown in Fig. 4. First, the *token generation* stage processes the textual input and generates tokens with relevant information for further analysis. Second, the *token analysis* stage receives the tokens and, for each crosscutting

[2] http://sourceforge.net/projects/opennlp/.

[3] see http://wordnet.princeton.edu/.

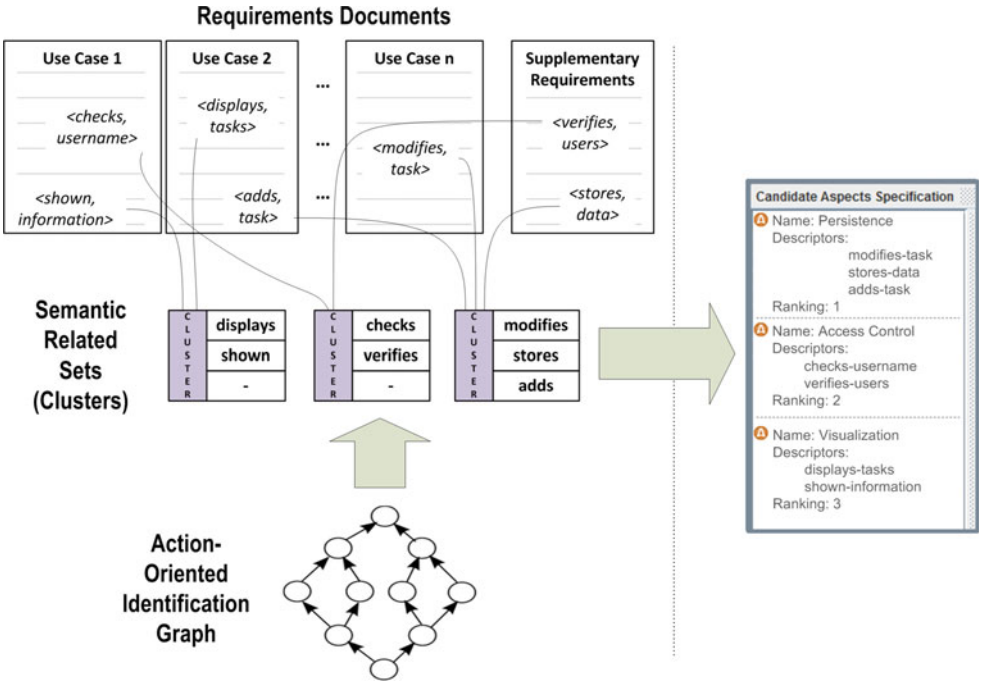


Fig. 3 Determination of candidate early aspects in SAET

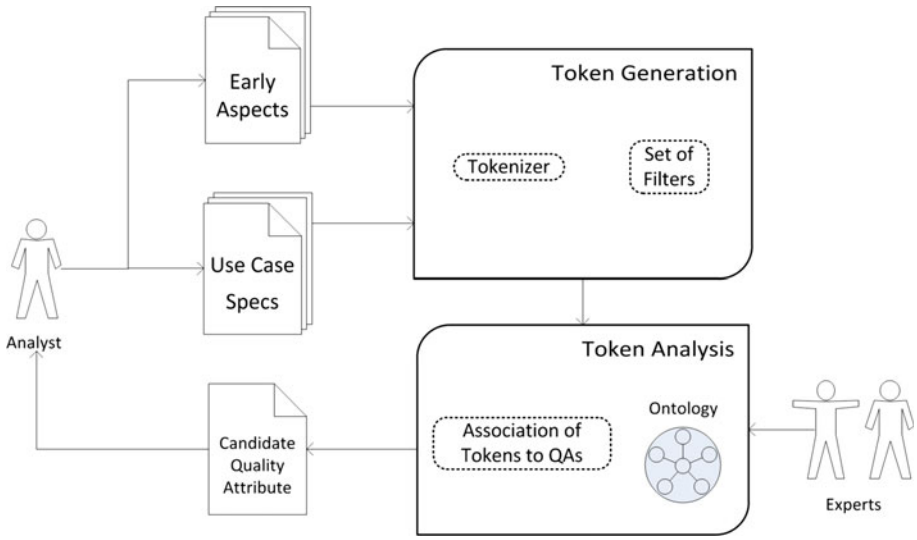


Fig. 4 QAMiner approach

concern, it computes a *score* with respect to predefined quality attributes. The information about these quality attributes is configured in the tool (in advance) using an ontology [19]. The scores from all the crosscutting concerns are combined, and those quality attributes with the highest scores are returned to the analyst as the candidate quality attributes of the system. In the following sub-sections, we describe each of the stages in detail.

3.1 Token generation

This stage involves a lexical and syntactic analysis of the input text (i.e., the use case specifications and early aspects). As mentioned previously, SAET is employed for early aspect identification. There are two sequential activities for token generation. First, input data are preprocessed and transformed into a list of tokens. We define a token as a basic unit of text with several attributes of the form <attribute, value>, for example: <kind, use-case>, <weight, 1>, <occurrences, 4>, etc. Table 1 describes the attributes associated with a token. Thus, the list of tokens contains words selected from each early aspect and use case. Second, a series of filters are applied on the list of tokens in order to remove irrelevant information. Each filter can be seen as a processing component that reads tokens, produces modifications on them, and outputs the

Table 1 Token attributes

Attribute	Description	Value
Id	Unique token identifier	Identifier number
Kind	Originating artifact (use case or early aspect)	Use case or early aspect
Section	Location of the token in the artifact	For use cases, the values are brief-description, normal-flow, etc. or for early aspects, the values are pairs containing verbs and direct objects
Occurrences	Number of token occurrences	Occurrence number
Weight	Weight according to relevance of token	Weight score

modified tokens, so that other filters can work on the tokens. This way, independent transformations can be applied using a pipe-and-filter style [5]. QAMiner implements five filters, each performing one of the following transformations:

- Lower case: The filter puts the words in lower case (e.g., *Persistence* → *persistence*).
- Stop words: The filter deletes non-relevant words, such as prepositions, pronouns, articles, and so on (e.g., *the, of*, and *are* are discarded).
- Stemming: The filter reduces the words to its root (e.g., *query* → *queri, changes* → *chang*).
- Weight: The filter weights each word according to its place in the use case specification (sections are name, description, main flow, precondition, etc).
- Frequency: The filter counts the occurrences of each word within the same section of the token.

To clarify the processing of the filters, let's consider the use cases of Fig. 1, as detailed in Table 2. Additionally, we know that the system has modifiability and availability requirements, as described in Table 3.

Table 2 Example of use case specification

Use case	Textual specification (extract)
Query information	…
	The system retrieves the list of health units stored
	…
	The unique identifier is used by the system to search the repository for the selected health unit
	…
Complaint specification	…
	The system saves the complaint
	…
	The common complaint information is parsed and stored with the OPENED state
	…
Login	…
	The system retrieves the employee details using the login as a unique identifier
	…
Register tables	…
	The system saves the data
	…
Update complaint	…
	The complaint information is updated to store the new information
	…
	The complaint changes are rolled-back
	…
Register new employee	…
	The system saves the new employee's data
	…
…	…

Table 3 Example of quality-attribute requirements

Availability	The system should be available 24 h a day, 7 days a week. Since the systems is not mission-critical, it might stay off until any fault is fixed
Storage medium	The system must be flexible in terms of the storage format, allowing the use of files or different databases (e..g, MySQL, Oracle, etc.)

Table 4 Token processing (first 3 filters)

Token	Attributes <name, value>	Filters		
		Lower-case	Stop-words	Stemming
Query	<id, 1001>, <kind, use case>, <section, name>	*query*	query	*queri*
information	<id, 1002>, <kind, use case>, <section, name>	information	information	*inform*
...	...	...	...	...
The	<id, 1201>, <kind, use case>, <section, basic flow>	*the*	~~the~~	-
system	<id, 1202>, <kind, use case>, <section, basic flow>	system	system	system
retrieves	**<id, 1203>, <kind, use case>, <section, basic flow>**	**retrieves**	**retrieves**	***retriev***
the	<id, 1204>, <kind, use case>, <section, basic flow>	the	~~the~~	-
list	<id, 1205>, <kind, use case>, <section, basic flow>	list	list	list
of	<id, 1206>, <kind, use case>, <section, basic flow>	of	~~of~~	-
health	<id, 1207>, <kind, use case>, <section, basic flow>	health	health	health
units	<id, 1208>, <kind, use case>, <section, basic flow>	units	units	*unit*
stored	<id, 1209>, <kind, use case>, <section, basic flow>	stored	stored	*store*
...	...	...	...	...
Persistence	<id, 2001>, <kind, early aspect>, <section, name>	*persistence*	persistence	*persist*
retrieves	<id, 2002>, <kind, early aspect>, <section, pair>	retrieves	retrieves	*retriev*
list	<id, 2003>, <kind, early aspect>, <section, pair>	list	list	list
search	**<id, 2004>, <kind, early aspect>, <section, pair>**	**search**	**search**	**search**
repository	<id, 2005>, <kind, early aspect>, <section, pair>	repository	repository	*repositori*
saves	<id, 2006>, <kind, early aspect>, <section, pair>	saves	saves	*save*
complaint	<id, 2007>, <kind, early aspect>, <section, pair>	complaint	complaint	complaint
stored	<id, 2008>, <kind, early aspect>, <section, pair>	stored	stored	*store*
state	<id, 2009>, <kind, early aspect>, <section, pair>	state	state	state
retrieves	<id, 2010>, <kind, early aspect>, <section, pair>	retrieves	retrieves	*retriev*
details	<id, 2011>, <kind, early aspect>, <section, pair>	details	details	*detail*
saves	<id, 2012>, <kind, early aspect>, <section, pair>	saves	saves	*save*
data	<id, 2013>, <kind, early aspect>, <section, pair>	data	data	data
updated	<id, 2014>, <kind, early aspect>, <section, pair>	updated	updated	*updat*
information	<id, 2015>, <kind, early aspect>, <section, pair>	information	information	*inform*
changes	<id, 2016>, <kind, early aspect>, <section, pair>	changes	changes	*chang*
rolled-back	<id, 2017>, <kind, early aspect>, <section, pair>	rolled-back	rolled-back	*rolledback*
saves	<id, 2018>, <kind, early aspect>, <section, pair>	saves	saves	*save*
data	<id, 2019>, <kind, early aspect>, <section, pair>	data	data	data
...	...	...	...	...

Table 4 lists the tokens for the "Query information" use case and the *Persistence* early aspect, as produced by the token generation stage. The leftmost column shows the input tokens, while the rightmost column shows the tokens after being processed by three of the filters. For example, token "**retrieves**" belongs to the basic flow of a use case. As another example, token "**search**" comes from an early aspect (actually, the token was extracted from the verb-object pairs for the *Persistence* aspect).

The application of the last two filters annotates the tokens with attributes about frequency and relevance (weight). These attributes will be used when computing the quality-attribute scores, as explained in Sect. 3.2. By default, we use the weights below for each section of a use case specification or early aspect.

- use case sections: name, description, priority, and actor → weight = 1;
- use case section: basic flow → weight = 2;
- use case sections: alternative flow, trigger, special requirement, pre-/post-conditions → weight = 3;
- early aspect: pairs → weight = 3;

If a token appears several times, the individual weights for each occurrence are added. In this schema, a recurrent or highly weighted token will probably be more relevant than other tokens. In Table 5, we show how the two filters process the tokens. The leftmost column are the tokens resulting from the previous filters (see Table 4), while the rightmost column are the final tokens (with all their attributes). For example, token "**inform**" has 3 occurrences, meaning that it is a frequent token in the documentation (both uses cases and early aspects). The same token has a total weight of 6, because it was found in several places, namely: a use case name, a use case basic flow, and an early aspect pair, with weights of 1, 2, and 3, respectively.

3.2 Token analysis

This stage involves the inference of quality attributes from the tokens. To do so, we rely on knowledge from an ontology that represents the main concepts of quality attributes [5]. As output, a set of quality attributes is assigned to every early aspect along with a *score* value per quality attribute. A high score indicates that the quality attribute is likely present in that early aspect. Basically, the predominant quality attribute(s) are identified by looking at subset(s) of tokens, considering that each subset corresponds to an early aspect. The information computed on the tokens (either coming from early aspects or use cases) is stored in data structures referred to as *maps*. More specifically, a *membership function f*: (*Token*) → *Map*

Table 5 Token counting and weighting (last 2 filters)

Token	Attributes <name, value>	Filters	
		Frequency	Weights
queri	<id, 1001>, <kind, use case>, <section, name>	<occurrences, 1>	<weight,1>
inform	**<id, 1002>, <kind, use case>, <section, name> <id, 1203>, <kind, use case>, <section, basic flow> <id, 2015>, <kind, early aspect>, <section, pair>**	**<occurrences, 3>**	**<weight, 6>**
system	<id, 1202>, <kind, use case>, <section, basic flow>	<occurrences, 1>	<weight, 2>
retriev	<id, 1203>, <kind, use case>, <section, basic flow> <id, 2002>, <kind, early aspect>, <section, pair> <id, 2010>, <kind, early aspect>, <section, pair>	<occurrences, 3>	<weight, 8>
list	<id, 1203>, <kind, use case>, <section, basic flow> <id, 2002>, <kind, early aspect>, <section, pair> <id, 2011>, <kind, early aspect>, <section, pair>	<occurrences, 2>	<weight, 5>
health	<id, 1207>, <kind, use case>, <section, basic flow>	<occurrences, 1>	<weight, 2>
unit	<id, 1208>, <kind, use case>, <section, basic flow>	<occurrences, 1>	<weight, 2>
store	<id, 1209>, <kind, use case>, <section, basic flow> <id, 2008>, <kind, early aspect>, <section, pair>	<occurrences, 2>	<weight, 5>
persist	<id, 2001>, <kind, early aspect>, <section, name>	<occurrences, 1>	<weight, 1>
search	<id, 2004>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
repositori	<id, 2004>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
save	<id, 2006>, <kind, early aspect>, <section, pair> <id, 2012>, <kind, early aspect>, <section, pair> <id, 2017>, <kind, early aspect>, <section, pair>	<occurrences, 3>	<weight, 9>
complaint	<id, 2007>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
state	<id, 2009>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
detail	<id, 2011>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
data	<id, 2013>, <kind, early aspect>, <section, pair> <id, 2019>, <kind, early aspect>, <section, pair>	<occurrences, 2>	<weight, 6>
updat	<id, 2014>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
chang	<id, 2016>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>
rolledback	<id, 2017>, <kind, early aspect>, <section, pair>	<occurrences, 1>	<weight, 3>

associates a token with a map, based on a quality-attribute ontology created for this work. Each entry in the map corresponds to a (different) quality attribute matched from the ontology, and the scores come from a weighted sum over the maps, as explained below.

Figure 5 depicts the organization of our ontology. Classes (or concepts) are shown in ovals, filled lines represent object properties, and dashed lines indicate an inheritance relationship between concepts. The ontology models the domain of quality attributes in terms of quality-attribute scenarios, based mainly on information from Bass et al. [5]. A scenario template has 6 parts, namely: STIMULUS, SOURCE, ENVIRONMENT, ARTIFACT, RESPONSE, and RESPONSE MEASURES. The QAMiner ontology was populated with scenario instances (based on the template) by domain experts. The ontology currently covers six quality attributes (modifiability, security, usability, availability, performance, and testability) and contains approximately 50 scenario instances, derived from multiple sources in the literature [4, 5, 41]. Figure 6 shows instances of quality-attribute scenarios for *Availability* and *Modifiability*, Fig. 6a and b respectively. Several properties are attached to these instances. For example, the scenario parts of the two instances have properties such as: "developer" as CONCRETE SOURCE, "response" or "add-delete-modify functionality" as CONCRETE STIMULUS, "build time" as CONCRETE ENVIRONMENT, "persistent storage-database-repository" as CONCRETE ARTIFACT, "be unavailable for a pre-specified interval" as CONCRETE RESPONSE, and "time in days" as CONCRETE RESPONSE MEASURE.

Given a token, the membership function traverses all the types (instances) of quality-attribute scenarios in the ontology, and for each scenario, the function tries to match the token to a specific part of the scenario. Essentially, we attempt here to answer the question: is the token related to a particular CONCRETE SOURCE, STIMULUS, ENVIRONMENT, ARTIFACT, RESPONSE, or RESPONSE MEASURE? If the token matched successfully the "root concept" for the scenario (i.e., the name of its quality attribute), it is recorded in the map. This process is repeated for all possible matchings of the token. Since there can be more than one instance for a given token, the map can include different quality attributes for the same token. The map will register the number of token matches (i.e., the frequency) for each quality attribute (see Fig. 7).

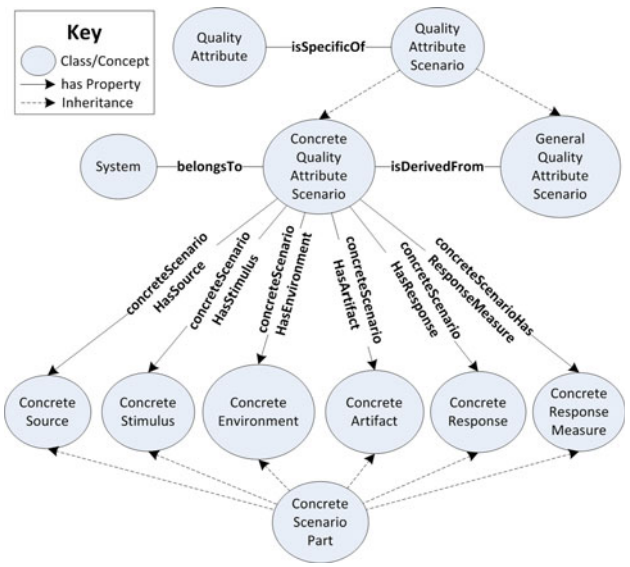


Fig. 5 Organization of the quality-attribute ontology

Let's exemplify how QAMiner determines the association of a list of tokens with quality-attribute scenario instances from the ontology. Let's go back to the preprocessed tokens in Tables 4 and 5 and assume that we would like to match the "repository" token (in Table 5) against the availability instance of Fig. 6a. When applied to the token, the membership function finds that class CONCRETE ARTIFACT has a property value "persistent storage/database/repository" that (partially) matches the "repository" token. As a result, Availability is bound to that token (in the corresponding map). Let's assume now that we would like to match the "repository" token against the modifiability instance of Fig. 6b. If so, the membership function finds another matching of the token with a property value for class CONCRETE ARTIFACT, which ends up adding modifiability to the map. The remaining tokens are processed following the same reasoning, being eventually matched to several parts of the scenario instances and then contributing (new) quality attributes to the map.

Once the maps for all the tokens are ready, the scores for the quality attributes are computed across maps (see schema in Fig. 7). To do so, the tool uses the weights associated with the tokens during the first stage. That is, having an early aspect (or use case) represented by a list of weighted tokens, the score for a quality attribute is the result of a weighted sum of the token frequencies for those tokens that matched the quality attribute. This way, the scores for all quality attributes in the maps are determined. The scores are organized according to the early aspects and presented to the analyst in a graphical manner. For instance, the snapshot of Fig. 8 shows the quality-attribute scores for the *Persistence* early aspect, and the pie chart indicates strong support for the quality attributes of modifiability and availability (in addition to security, which is a false positive).

We should note that the relationships between quality attributes and early aspects provided by the tool should be interpreted by the analyst as hints rather than as definite relationships. The role of QAMiner is to provide a quick overview of latent quality-attribute information in the use cases (related to the early aspects provided by SAET). The

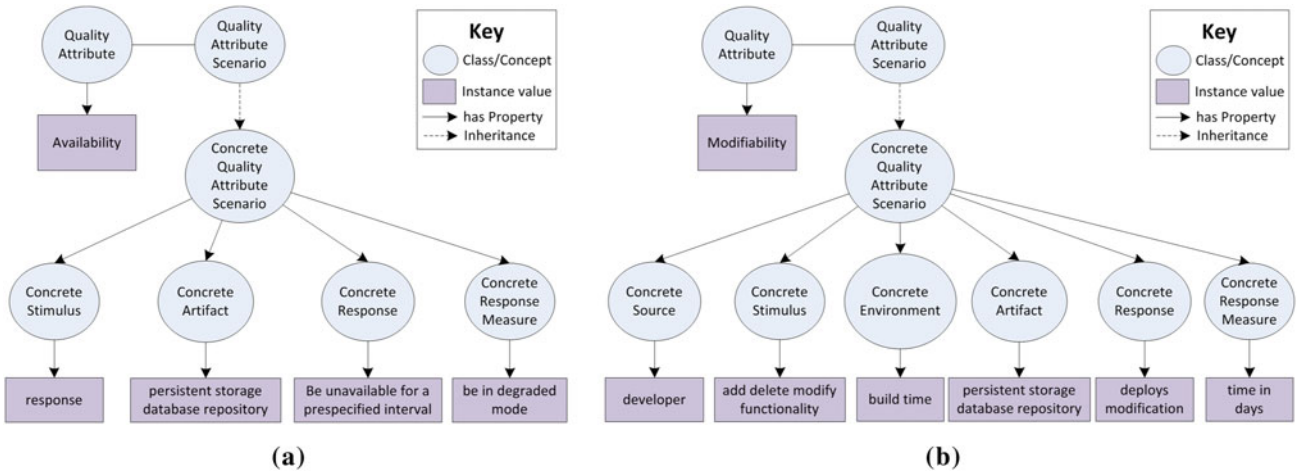


Fig. 6 Examples of ontology instances. **a** Availability instance. **b** Modifiability instance

Fig. 7 Map calculation

map

Token t	Weight W_t	Quality Attribute QA					
		Security	Availability	Testability	Modifiability	Usability	Performance
inform	6	2	0	0	3	0	0
retriev	8	0	0	0	1	0	2
unit	2	0	0	2	0	0	0
store	5	2	0	0	0	0	1
persist	1	0	1	0	2	0	0
repositori	3	0	1	0	1	0	0
state	3	0	0	1	0	2	0
		2	2	0	0	1	1
			0	0	2	0	0
				0	0	0	

score calculation

Score $\sum_t \text{Score}(t, QA) \cdot W_t$	34	22	7	37	12	27
Percentage $\frac{\text{Score}(QA)}{\sum_{QA}\text{Score}(QA)} \cdot 100$	24,46%	15,83%	5,04%	26,62%	8,63%	19,42%

quality attributes outputted by QAMiner are, therefore, "informed guesses", which help the analyst's decisions. The analyst should also analyze other sources of quality attributes (e.g., business mission, stakeholders' workshops) in order to have a complete picture of the relevant quality attributes.

4 Evaluation

To evaluate the advice of the QAMiner approach regarding quality attributes, we applied the tool in two case studies. The first case-study is called the HWS [18, 24], and the second one is a public IBM case-study called course registration system (CRS) [7]. In the experiments, we were interested in the predictive output of the tool, that is, its ability to recover relevant (candidate) quality attributes from a use case specifications, given a set of early aspects and quality attributes identified beforehand. We used the precision and recall metrics from the information retrieval (IR) [17] field in order to assess the results of the experiments. Remember that an early aspect can lead to more than one relevant quality attribute. To deal with this situation, if QAMiner suggests several quality attributes for a particular early aspect, all the quality attributes are accounted for the purpose of our evaluation.

For each case-study, the procedure to perform the experiments involved a pre-processing of the use cases with SAET and two phases. Initially, we fed the use cases of the case-study into SAET to get a list of early aspects. In

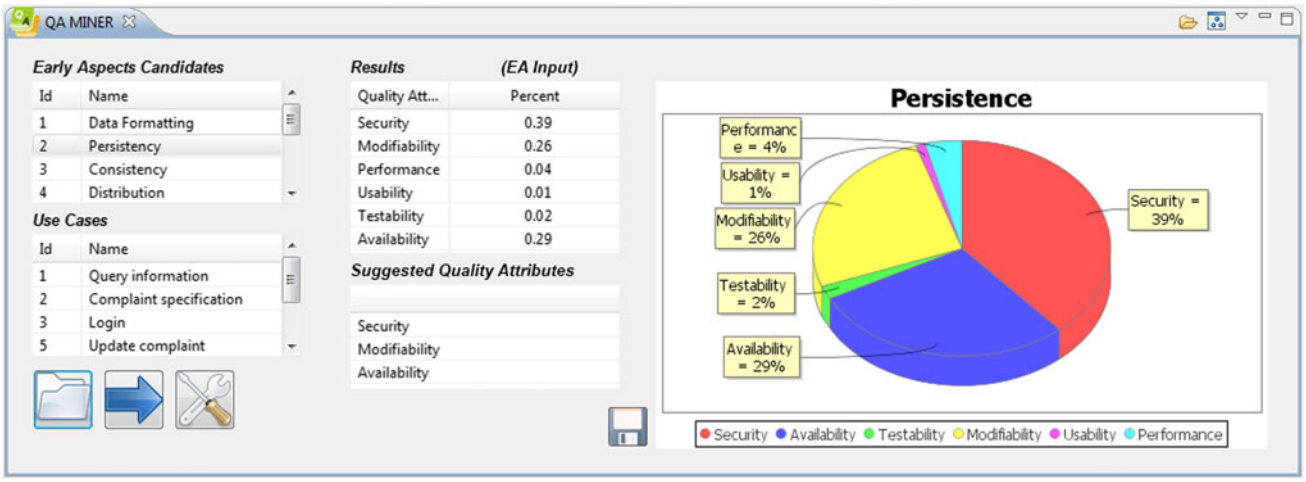


Fig. 8 QAMiner snapshot

the first phase, we processed the (same) use cases with QAMiner and obtained a ranking of quality attributes, based on the knowledge stored in the ontology. Also, we studied the influences of the early aspects on this ranking. The question to answer here was whether the early aspects improve the precision of the resulting quality attributes, when compared to the plain analysis of use cases (i.e., using QAMiner without SAET). In the second phase, we validated the consistency of the ranking of quality attributes produced with QAMiner against the actual quality attributes of the system. The question to answer here was whether the candidate quality attributes have correspondences with the quality attributes considered in solutions to the case studies. In practice, several authors have proposed architectures for the HWS problem [24, 42], so we directly took the quality attributes stated and addressed by those architectures. For the CRS case-study, unfortunately, there is no published architectures, and we had to infer the relevant quality attributes from the problem context.

4.1 Summary of the case studies and pre-processing

The HWS is a typical Web information system and a real-life application. Briefly, the HWS application provides online access for users to register complaints, read health notices, and make queries regarding health issues. There are two relevant actors in this system: employees and citizens (remember Fig. 1). An employee can record, update, delete, print, search, and change the records that are stored in the system repository. All these operations require the employee to be authenticated by the system. A citizen can enter complaints, which are registered also in the repository. In response to each complaint, the system generates a complaint code. The reason for choosing HWS as a case-study is twofold. First, a rich set of development artifacts are available (e.g., requirements specifications, software architecture documents, source code, among others) [18]. Specific to our work, the HWS provides a requirements specification with 9 use cases (around 19 textual pages). Second, other researchers have used HWS as a model problem and analyzed its quality-attribute properties [40, 42]. After running SAET [34, 35] on the use cases of HWS, we detected the early aspects listed in Table 6.

The CRS is a distributed system to be used within a university intranet. It was developed by IBM to demonstrate their CASE tools. The development artifacts publicly available include the requirements specifications, some analysis classes, and a few design documents. Particularly, the CRS specification consists of 8 use cases (around 20 textual pages). The CRS arises as a replacement of an old

Table 6 Early aspects at HWS

Data formatting	Deals with the presentation and formatting of several outputs of the HWS
Persistence	Handles all operations within the HWS that need to be stored and accessed through a data store
Consistency	Is in charge of verifying that inputs of the HWS system are consistent
Distribution	Manages all the interaction between distributed clients and the main server of the system
Access control	Addresses the task of ensuring that only authenticated users (employees) can execute restricted functionality
Error handling	Manages the behavior of the HWS system when an error occurs

Table 7 Early aspects at CRS

Persistence	Encapsulates the access to a data repository used to store information of the CRS
Entitlement	Ensures that users are logged and authenticated before accessing any system functionality, and additionally verifying users have enough permissions to execute certain tasks
Communication	Handles all communications performed with other systems, such as the existing legacy system and the billing system
Access control	Is in charge of ensuring that non-registered users cannot access the CRS
Ease-of-use	It is in charge of providing several facilities to improve the user experience of CRS (such as cancellation, message confirmations, among others)
Data validation	Manages the validation of certain kind of input data (formatting, invalid characters, etc.)
Data processing	Deals with batch processing of information within the CRS

educational institution system, and it has two main actors: students and professors. A student can apply to academic courses and, after taking a course, she can obtain the grade reports. A professor can add new courses and report students' grades. After running SAET on the use cases of CRS, we detected the early aspects listed at Table 7. Using the use case specifications of both case studies and the corresponding early aspects, we proceeded to address the two aforementioned questions, as discussed in the following sub-sections.

4.2 Question 1: can early aspects improve the detection of quality attributes?

We conducted a number of runs with different inputs to QAMiner. The input variations involved (1) using only early aspects (EA), and (2) using only use cases (UCs). Tables 8 and 9 summarize the results of the tool for the two case studies, respectively. Each table is organized as follows. The top row corresponds to the names of the early aspects previously detected with SAET (e.g., *Data Formatting*, *Persistence*, etc.). The uses cases are crosscuted by these early aspects. The leftmost column (and their rows) represents the quality attributes modeled in the ontology (e.g., security, availability, testability, etc.), which actually drove the quality-attribute identification performed by QAMiner. The internal columns, tagged as EA or UC, correspond to the type of input used for the runs. The numbers in the matrix are the scores computed by QAMiner with respect to the quality attributes (from the ontology), depending on the input type. Numbers in bold denote those quality attributes suggested as relevant to the analyst. For instance, security received a score of 0.19 when using the *Data Formatting* early aspect, and the same score was 0.28 when using just the use cases for that early aspect. In the first case, the tool considered security as not relevant (as it was outperformed by Usability), while in the second case, the tool considered both Security and Usability as relevant. The score differences are due to the kind of information provided by the inputs, when their tokens are analyzed in the context of the ontology.

In the experiments with HWS, the quality-attribute scores were affected by the input types. On one hand, we found that using information from the use case specifications (i.e., the full text from the affected use cases) generally added "noise" to the detection of quality attributes. This noise had several effects in the outputs of the tool. A noticeable effect was that the quality attributes marked as relevant received relatively even, low scores. For example, in HWS (see Table 8), we knew in advance that the *Data Formatting* concern has to do with data presentation and

Table 8 QAMiner outputs on HWS

EAs (from SAET)	Data formatting		Persistence		Consistency		Distribution		Access control		Error handling	
QAMiner input type	EA	UCs	EA	UCs	EA	UCs	EA	UCs	EA	UCs	EA	UCs
Ontology QAs												
Security	0.19	**0.28**	**0.39**	**0.28**	**0.59**	**0.27**	0.05	**0.28**	**1.00**	**0.38**	**0.70**	**0.29**
Availability	0.00	0.07	**0.29**	0.07	0.02	0.06	**0.87**	0.06	0.00	0.09	0.00	0.06
Testability	0.08	0.11	0.02	0.11	0.00	0.14	0.00	0.12	0.00	0.03	0.00	0.11
Modifiability	0.00	0.09	**0.26**	0.09	0.04	0.13	0.00	0.11	0.00	0.01	0.00	0.12
Usability	**0.69**	**0.31**	0.01	**0.31**	0.05	**0.26**	0.00	**0.28**	0.00	**0.35**	0.20	**0.26**
Performance	0.04	0.14	0.04	0.14	0.30	0.14	0.08	0.15	0.00	0.15	0.10	0.15

Table 9 QAMiner outputs on CRS

EAs (from SAET)	Persistence		Entitlement		Communication		Access control	
QAMiner input	EA	UCS	EA	UCS	EA	UCS	EA	UCS
Ontology QAs								
Security	0.08	0.11	0.13	0.11	0.00	0.07	**0.40**	0.11
Availability	0.18	0.15	**0.87**	0.15	**0.50**	**0.24**	0.00	0.15
Testability	0.04	0.05	0.00	0.05	0.00	0.02	0.00	0.05
Modifiability	**0.69**	**0.25**	0.00	**0.25**	0.00	0.18	0.10	**0.28**
Usability	0.00	**0.30**	0.00	**0.30**	**0.40**	**0.30**	0.20	**0.28**
Performance	0.01	0.14	0.00	0.14	0.10	0.19	**0.30**	0.12

EAs (from SAET)	Ease-of-use		Data validation		Data processing	
QAMiner input	EA	UCS	EA	UCS	EA	UCS
Ontology QAs						
Security	0.24	0.10	**0.95**	0.11	0.00	0.12
Availability	0.03	0.15	0.00	0.15	0.00	**0.22**
Testability	0.00	0.05	0.00	0.05	0.25	0.05
Modifiability	0.24	**0.27**	0.00	**0.25**	0.00	0.06
Usability	**0.48**	**0.30**	0.00	**0.30**	0.25	**0.31**
Performance	0.01	0.14	0.05	0.14	**0.50**	**0.25**

formatting (that is to say, a usability connotation). However, the tool computed scores of 0.31 for Usability and 0.28 for Security (both as potential quality attributes), when using use cases. Another effect was that, sometimes, the tool suggested disparate quality attributes as relevant ones. In the example, *Data Formatting* has little to do with Security, so the analyst should discard it, even when recommended by QAMiner. We attribute the bias to the wording employed in the use cases, which causes tokens to be likely matched with instances of Security and Usability within the ontology (see the amount of bold scores for these two attributes in Table 8). In some other cases, this effect of use cases was mild, and the resulting quality attributes did have relationships with the early aspects. This happened, for example, with the *Access Control* concern, which can be certainly related to both usability and security. On the other hand, the compact representation of early aspects (based on verb-object pairs as descriptors) helped reducing this noise. In the runs using only early aspects as input, the tool achieved better results in terms of discriminating potential quality attributes. In the example of *Data Formatting* above, the tool appropriately associated the concern just to Usability with a score of 0.69. Nonetheless, note that using early aspects can still point to more than one quality attribute for the same concern. For example, the *Persistence* concern revealed three potential quality attributes, all being apparently relevant in HWS.

The experiments with CRS (see Table 10) reported similar trends as the HWS experiments. For instance, QAMiner related the *Data Processing* early aspect to *Performance* with a score of 0.5, when taking only early aspects as input, while the score dropped to 0.25 when using use cases. In the latter run, the tool detected again additional quality attributes such as Usability and Availability. With inputs based on early aspects, the tool mainly pointed to single quality attributes, with exceptions for the *Communication* and *Access Control* aspects.

Table 10 QAMiner metrics

	Precision	Recall	Accuracy
HWS	0.625	0.666	0.865
CRS	0.555	1.000	0.905

$P(\text{HWS}) = \frac{5}{5+3}$ $R(\text{HWS}) = \frac{4}{4+2}$ $A(\text{HWS}) = \frac{4+28}{4+2+3+28}$

$P(\text{CRS}) = \frac{5}{5+4}$ $R(\text{CRS}) = \frac{5}{5+0}$ $A(\text{CRS}) = \frac{5+33}{5+0+4+33}$

Overall, from the results of the two case studies, we argue that the information contained in the early aspects is a good mechanism to discriminate relevant quality attributes. In other words, the early aspects work as a "term-based summary" of the use cases for the purpose of the quality-attribute analysis performed by QAMiner. The evidence from the experiments, although preliminary, validates the inclusion of SAET for pre-processing the QAMiner inputs. In addition, the early aspects contribute to reduce the search for matchings in the quality-attribute ontology, thus reducing the QAMiner processing time.

4.3 Question 2: are the detected QAs correct?

In the previous section, we analyzed the quality attributes identified as relevant by the tool. An important issue here is whether those quality attributes were actually the right ones for the system. To answer this question, we cross-checked the QAMiner outputs against documentation already provided by the case studies. In HWS, the analysis of the "actual" quality attributes was based on existing architectural sources [40, 42], while in CRS, the actual quality attributes were gleaned from the documented business goals for the system [7]. Since QAMiner performs better when using only early aspects as input, we used the results of that configuration to perform the cross-checking analysis. The rationale of our analyses is sketched in Figs. 9 and 10, for HWS and CRS, respectively.

The leftmost column lists the input early aspects, at the beginning of the process. In the middle column, we have the quality attributes identified by QAMiner, as well as arrows showing the originating early aspects with the corresponding scores (this information is taken from Tables 8 and 9). A filled arrow denotes a correct suggestion of QAMiner, while a dashed arrow denotes an incorrect one. This judgment was performed based on the descriptors (i.e., verb-object pairs) and intent of each early aspect. At last, the rightmost column contains the system quality attributes, that is, the actual quality attributes as interpreted from the case-study documentation [18, 24, 42]. The lines between identified and actual quality attributes highlight the correspondences between them. The fact that the same quality attribute is sometimes identified twice by the tool (e.g., Availability in Fig. 9) is because the early aspects expose different "parts" of the reference quality attribute.

To assess the quality attributes suggested by QAMiner, we adapted IR [2] metrics to a quality-attribute terminology as follows. The set of actual quality attributes (AQAs) for the system is known in advance. A quality true positive (QTP) is a quality attribute identified correctly by QAMiner, which also belongs to AQAs. A quality false positive

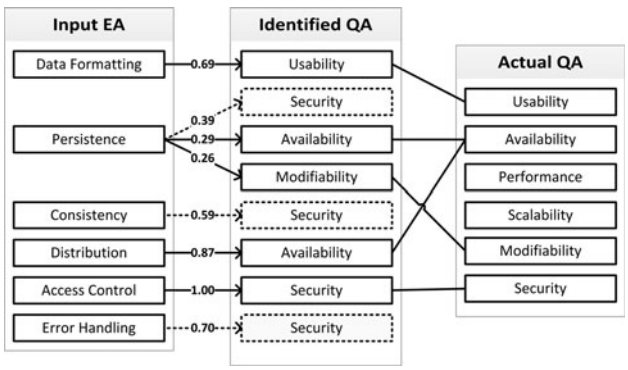


Fig. 9 QAMiner output analysis for HWS

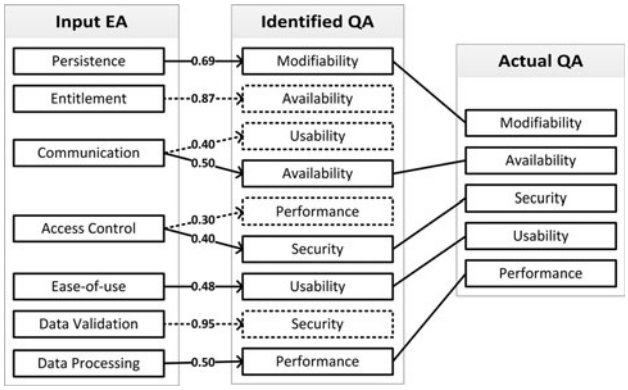


Fig. 10 QAMiner output analysis for CRS

(QFP) is a quality attribute outputted by QAMiner that is not part of AQAs. A quality false negative (QFN) is a quality attribute that is not identified by QAMiner, but it belongs to AQAs. A quality true negative (QTN)[4] is a quality attribute that is neither detected by QAMiner nor included in AQAs. Note that AQAs is the union of the sets QTPs and QTNs. Along this line, the formula for precision, recall, and accuracy are the following:

$$P = \frac{\text{QTP}}{\text{QTP} + \text{QFP}}$$

$$R = \frac{\text{QTP}}{\text{QTP} + \text{QFN}}$$

$$A = \frac{\text{QTP} + \text{QTN}}{\text{QTP} + \text{QFP} + \text{QFN} + \text{QTN}}$$

Given these criteria, we have the following values for HWS (see Fig. 9):

- 6 AQAs (actual quality attributes) → Usability, availability, performance, scalability, modifiability, and security
- 5 QTPs (quality true positive) → Usability (from *Data Formatting* early aspect), availability and modifiability (from *Persistence* early aspect), availability (from *Distribution* early aspect), and security (from *Access Control* early aspect)
- 3 QFPs (quality false positive) → Security (from *Persistence*, *Consistency*, and *Error Handling* early aspects)
- 2 QFNs (quality false negative) → Performance and scalability
- 28 QTNs (quality true negative)

Analogously, we have the following values for CRS (see Fig. 10):

[4] To calculate QFN, we count those quality attributes not related to an early aspect which were not suggested by QAMiner. That is, for each quality attribute the maximum value of QFN is 6, when no quality attribute is associated with that early aspect.

- 5 AQAs (actual quality attributes) → Modifiability, availability, security, usability and performance
- 5 QTPs (quality true positive) → Modifiability (from *Persistence* early aspect), availability (from *Communication* early aspect), security (from *Access Control* early aspect), usability (from *Ease-of-use* early aspect), and performance (from *Data Processing* early aspect)
- 4 QFPs (quality false positive) → Availability (from *Entitlement* early aspect), usability (from *Communication* early aspect), performance (from *Access Control* early aspect) and security (from *Data Validation* early aspect)
- no QFNs (quality false negative)
- 33 QTNs (quality true negative)

Table 10 summarizes the metrics for both case studies. Notice that the calculation of recall for HWS uses 4 QTPs instead of 5. Such a change is explained by the redundancy of Availability, which was hinted correctly by two early aspects. In general, our approach obtained high accuracy and recall when predicting quality attributes. The only quality attribute that was mistakenly classified in a few cases was security. This issue seems related to the way of alluding to (or spelling) security concerns in the textual specification, which causes many tokens to be matched with incorrect scenario instances in the ontology. Note that a similar bias was observed in Sect. 4.2. Further analysis work is needed to determine a mitigation strategy for this kind of problems.

In contrast, the precision was not as good as expected in both case studies, due to incorrect quality-attribute predictions from certain early aspects. A number of factors contributed to a relatively low precision of QAMiner. First, some early aspects were not sufficiently accurate, in the sense that the information provided by their descriptors was not always representative of the aspect semantics. For example, a *Consistency* early aspect (HWS) was composed of verbs like "ensure", which does not have a direct relationship with neither an availability nor a security quality attribute. This factor impacted negatively on the ontology matching process. Second, even when some aspects were properly characterized by their descriptors, the tokens failed to match because the ontology lacked quality-attribute instances for the things addressed by the aspects. That was the case of the *Entitlement* early aspect (CRS), in which words like "logged" were good hints but the matching with security scenarios from the ontology failed. Another limitation had to do with quality attributes that were included in the ontology but were never suggested by QAMiner, such as the testability quality attribute. This is so because it is rather unusual to have textual references to testing-related concerns in use cases. During the experiments, we actually searched through the requirements documents and could not find clues for testability in any of the case studies. Regarding the performance of QAMiner (without considering SAET), the results were computed in few seconds for the textual specifications of the two case studies. We speculate that both the token generation and token analysis stages can handle large specifications with a good performance.

4.4 Threats to validity

QAMiner detected, in a semi-automatic fashion, a number of quality attributes from a set of use case specifications and early aspects. However, there were issues that might compromise the results of the experiments, regarding the two questions we intended to answer. Three threats to internal validity were identified. First, the quality-attribute identification is dependent on the early aspects provided as input to the tool, in our case produced by SAET. Although we validated these inputs with field experts, the workings of SAET are based on heuristics whose effects can ripple to QAMiner. Second, the domain knowledge codified in our quality-attribute ontology is incomplete, despite the various instances of quality-attribute scenarios loaded in the ontology. Furthermore, these scenarios might favor certain types of quality-attribute matchings over others. Since it is easy to add new concepts, relationships and scenarios to the ontology, we plan to address this limitation in future work. And third, determining the "actual" quality attributes for a system involves subjective interpretations either from the people that created the case-study documents or from the authors. The analysis of the outputs of QAMiner in Sect. 4.3 is affected by this assessment. The authors' prior knowledge of both case studies could have also influenced the results. A way to alleviate these issues is to perform a controlled experiment with different people, collecting their responses and then computing consensus and reliability indexes for the study [8]. This kind of study is beyond the scope of the QAMiner work.

When it comes to external validity, QAMiner was tested only with medium-size case studies, but we have reasons to believe that the approach is applicable to other systems. The performance of the tool was very reasonable. In addition, the usage of an ontology contributed to keep this performance low. On the downside, the mining of early aspects (with SAET) is not as fast as QAMiner. Another threat to external validity refers to the quality-attribute support currently provided by the ontology, which might not suffice if other systems are considered, because quality-attribute definitions are non-operational and depend on the system context. As the ontology gets populated with more knowledge about quality attributes, we expect to minimize this problem.

5 Related work

In general, the problem of identifying and classifying quality-attribute requirements can be solved in two ways [14]. First, quality-attribute concerns can be elicited via interviews and negotiation with the stakeholders. Second, the concerns can be extracted semi-automatically from functional specification documents. Existing approaches to mine quality-attribute information often rely on techniques such as IR or NLP [11]. In addition, there are hybrid approaches that combine eliciting techniques with mining tools.

Dörr et al. [15] present an approach whose goal is to achieve a minimal, complete, and focused set of non-functional requirements. Minimal means that only necessary non-functional requirements are stated. Complete means that all non-functional requirements of the stakeholders are identified. Focused means that the impact of the non-functional requirements on the solution is clear. More interestingly, the set of requirements is expected to be measurable and traceable. Measurable refers to the provision of metrics to verify that the system satisfies the non-functional requirements, while traceable refers to the rationale explaining why the non-functional requirements are necessary. The authors define a meta-model in order to support the concepts of the approach, allowing instantiations of different quality models.

A model to identify and specify quality attributes that crosscut requirements is presented in [30]. It includes a systematic integration of those quality attributes with the functional specification. The model comes with a process that consists of three activities: identification, specification, and integration of requirements. During the first activity, the functional requirements are specified using use cases. In the second one, quality attributes are described by means of a special template. Finally, the third activity proposes a set of models to capture the integration of crosscutting quality attributes and functional requirements.

Both approaches above [15, 30] emphasize the need for models to identify, specify, and compose quality-attribute requirements during early development stages. By doing so, several issues derived from incomplete, ambiguous, or non-traceable quality concerns can be avoided. Nonetheless, we argue that Dörr's approach is still complex to be applied in practical development projects. Also, it does not handle all crosscutting requirements properly, and some of them can remain scattered through multiple documents. Moreira's model, in turn, deals successfully with crosscutting quality concerns in requirement stages, but it does not discriminate between quality attributes and early aspects. We think that being able to trace early aspects to quality attributes, as QAMiner does, is beneficial because it facilitates the understanding of requirements specifications and their further usage in software design.

In [14], non-functional requirements are detected and classified using a supervised classification approach. IR techniques are used to process quality attributes scattered across both structured and unstructured documents. It is assumed that a quality-attribute type (e.g., security, performance, etc.) is characterized by keywords called "indicator terms". When a quality attribute has enough indicator terms, it is possible to identify requirements, sentences, or phrases related to that quality attribute. The same technique is also employed to detect and classify early aspects [13].

Another requirements classification technique is presented in [25], which can automatically produce keyword lists and classify requirements, as part of an analysis system for Internet-based requirements. The approach classifies the collected requirements into several categories, or topics. These categories can be cost, priority, development time, quality attributes, and so on. The implementation is based on NLP techniques.

A semi-supervised approach for detection and classification of quality attributes from text is presented in [11]. A classifier automatically recognizes the different types of a predefined category or class within a set of documents and then presents the results to the analyst for inspection. The classifier is built from a set of categorized and non-categorized documents, using a semi-supervised learning algorithm. Once a given classifier is trained, it can be used to categorize unlabeled requirements.

In contrast to the three approaches above [11, 13, 25], which use machine learning techniques (e.g., classification) to uncover quality attributes, QAMiner uses domain knowledge about quality attributes and early aspects for achieving the same purpose. Machine learning schemes, based on keywords extracted from the text, present some disadvantages regarding knowledge updates. In a tool using classification algorithms, customizing or increasing the knowledge of the tool for a particular domain can be difficult. The main reason is that the underlying classifications model often needs to be re-trained (and possible re-evaluated) in order to adapt to changes. In the concrete case of quality attributes, the training would involve either finding already-labeled datasets or having a human to manually label requirements. On the other hand, ontologies (like the one used by QAMiner) are more flexible regarding the incorporation of domain knowledge. Another disadvantage of the existing approaches is that they work on the full extent of textual requirements, without distinguishing between relevant and non-relevant ones (i.e., quality-related requirements versus unrelated requirements). Processing unnecessary information might add noise to the detection, degrading its effectiveness or performance. QAMiner benefits from early aspect descriptors to alleviate this issue, because early aspects are frequently related to quality-attribute requirements.

Related to the representation of information with ontologies, Kayed et al. [22] describe an ontology for quality measurements in web or desktop applications. They have studied and analyzed various documents, reports, and proposals concerned with software measures, attributes, and quality, in order to extract concepts, definitions, and terminologies from them. The goal of the ontology is to achieve a common understanding of quality attributes for web applications.

Overall, the ultimate goal of these approaches (including QAMiner) is to aid analysts in the identification and specification of quality attributes, which are present (but often not explicitly stated) in functional requirements documents. We believe there is still room for improvements regarding precision and performance of these tools.

6 Conclusion

In this article, we present a novel approach, called QAMiner, to detect quality-attribute concerns in use case specifications. We take advantage of early aspects previously identified in the use cases in order to improve that detection. The QAMiner tool relies on a schema of filters to process input data and on a quality-attribute ontology to suggest candidate quality attributes to the analyst. Based on a standard format of quality-attribute scenarios, we have constructed an ontology that contains domain concepts and relationships for predefined quality attributes. The main contributions of QAMiner are the role of early aspects as drivers for identifying quality attributes and the usage of an extensible ontology for representing quality information. Furthermore, although not directly supported by the current tool, we believe that QAMiner helps analysts to clarify traceability relationships between requirements and further design activities.

Our evaluation of QAMiner with two publicly available systems has shown promising results, particularly in terms of accuracy and recall of the quality attributes suggested by the tool. On the downside, the approach has some limitations related to the current knowledge stored in the ontology, which causes relatively low precision results. As future work, there are several lines of research. First, we plan to evaluate QAMiner with other case studies and within controlled environments involving multiple quality-attribute experts and developers. As part of these experiments, we will also analyze the scalability of the approach to handle large use case specifications. We will enhance the current ontology, both by adding new instances of (existing) quality-attribute scenarios and by incorporating new quality attributes. These additions are expected to improve the detection of quality-attribute requirements. Finally, we would like to investigate extensions of QAMiner, based on latent semantic analysis [16], so that the tool can recommend traceability links between requirements and design artifacts.

Acknowledgments We want to thank Sebastian Villanueva and Francisco Bertoni, who implemented the QAMiner prototype and helped us to evaluate the tool with the case studies. We are also grateful to the anonymous reviewers for their feedback to improve the quality of the manuscript.

References

1. Araujo J, Baniassad E, Clements P, Moreira A, Rashid A, Tekinerdogan B (2005) Early aspects: the current landscape. Technical Notes, CMU/SEI and Lancaster University
2. Baeza-Yates R, Ribeiro-Neto B et al (1999) Modern information retrieval, vol 463. ACM press, New York
3. Barbacci M, Klein M, Longstaff T, Weinstock C (1995) Quality attributes. Technical Report
4. Bass L, Klein M, Moreno G (2001) Applicability of general scenarios to the architecture tradeoff analysis method. Technical Report CMU/SEI-2001-TR-014, Software Engineering Institute (SEI), Carnegie Mellon University (CMU)
5. Bass L, Clements P, Kazman R (2003) Software architecture in practice. SEI series in software engineering. Addison-Wesley Professional, Boston, Massachusetts
6. Bass L, Klein M, Northrop L (2004) Identifying aspects using architectural reasoning. In: Tekinerdogan B, Moreira A, Araújo J, Clements P (eds) Early aspects: aspect-oriented requirements engineering and architecture design. Lancaster University, Lancaster, p 51
7. Bell R (2011) Course registration system. http://sce.uhcl.edu/helm/RUP_course_example/courseregistrationproject/indexcourse.htm
8. Ben-David A (2008) Comparison of classification accuracy using cohen's weighted kappa. Expert Syst Appl 34(2):825–832
9. Brito I, Moreira A (2004) Integrating the nfr framework in a re model. In: Tekinerdogan B, Moreira A, Araújo J, Clements P (eds) Early aspects: aspect-oriented requirements engineering and architecture design. Lancaster University, Lancaster, p 28
10. Burge JE, Brown DC (2002) Nfr's: fact or fiction? Technical Report
11. Casamayor A, Godoy D, Campo M (2010) Identification of non-functional requirements in textual specifications: a semi-supervised learning approach. Inf Softw Technol 52(4):436–445
12. Chung L, Prado Leite J (2009) On non-functional requirements in software engineering. In: Borgida AT, Chaudhri VK, Giorgini P (eds) Conceptual modeling: foundations and applications. Springer, Berlin, pp 363–379
13. Cleland-Huang J, Settimi R, Zou X, Solc P (2006) The detection and classification of non-functional requirements with application to early aspects. In: Requirements engineering, 14th IEEE international conference. IEEE Computer Society, pp 39–48
14. Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requirements Engineering 12:103–120. doi:10.1007/s00766-007-0045-1, http://portal.acm.org/citation.cfm?id=1269901.1269904
15. Dörr J, Kerkow D, Von Knethen A, Paech B (2003) Eliciting efficiency requirements with use cases. In: International workshop on requirements engineering: foundation for software quality (REFSQ03), p 23
16. Dumais S (2004) Latent semantic analysis. Annu Rev Inf Sci Technol 38(1):188–230

17. Frakes W, Baeza-Yates R (1992) Information retrieval: data structures and algorithms. Prentice Hall, Englewood Cliffs, New Jersey
18. Greenwood P (2011) Tao: A testbed for aspect oriented software development. http://www.comp.lancs.ac.uk/~greenwop/tao/
19. Gruber T (1993) A translation approach to portable ontology specifications. Knowl Acquis 5(2):199–220
20. Grundy J (1999) Aspect-oriented requirements engineering for component-based software systems. In: Requirements engineering, p 84
21. Kamata M, Tamai T (2007) How does requirements quality relate to project success or failure? In: Requirements engineering conference, 2007. RE'07. 15th IEEE International. IEEE Computer Society, pp 69–78
22. Kayed A, Hirzalla N, Samhan A, Alfayoumi M (2009) Towards an ontology for software product quality attributes. In: 2009 Fourth international conference on internet and web applications and services, IEEE, pp 200–204
23. Khan S, Rehman M (2005) A survey on early separation of concerns. In: Proceedings of the 12th Asia-Pacific software engineering conference. IEEE Computer Society, pp 776–782
24. Khan S, Greenwood P, Garcia A, Rashid A (2008) On the interplay of requirements dependencies and architecture evolution: an exploratory study. In: Proceedings of the 20th international conference on advanced information systems engineering. CAiSE, pp 16–20
25. Ko Y, Park S, Seo J, Choi S (2007) Using classification techniques for informal requirements in the requirements analysis-supporting system. Inf Softw Technol 49(11–12):1128–1140. doi:10.1016/j.infsof.2006.11.007, http://www.sciencedirect.com/science/article/B6V0B-4MRFC79-3/2/00f337d38f2840fe330b5caf12a09c65
26. Kof L (2005) Natural language processing: mature enough for requirements documents analysis? In: Montoyo A, Munoz R, Métais E (eds) Natural Language Processing and Information Systems. Lecture notes in computer science, vol 3513. Springer, Berlin/Heidelberg, pp 3–29. doi:10.1007/11428817_9
27. Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: Proceedings of the 5th annual international conference on systems documentation. ACM, pp 24–26
28. Luisa M, Mariangela F, Pierluigi N (2004) Market research for requirements analysis using linguistic tools. Requir Eng 9(1):40–56
29. Manning C, Schutze H, MITCogNet (1999) Foundations of statistical natural language processing, vol 59. MIT Press, Cambridge, Massachusetts
30. Moreira A, Araújo J, Brito I (2002) Crosscutting quality attributes for requirements engineering. In: Proceedings of the 14th international conference on software engineering and knowledge engineering. ACM, New York, NY, USA, SEKE '02, pp 167–174. doi:10.1145/568760.568790
31. Nuseibeh B, Easterbrook S (2000) Requirements engineering: a roadmap. In: Proceedings of the conference on the future of software engineering. ACM, New York, NY, USA, ICSE '00, pp 35–46. doi:10.1145/336512.336523
32. Patwardhan S, Banerjee S, Pedersen T (2003) Using measures of semantic relatedness for word sense disambiguation. In: Gelbukh A (ed) Computational Linguistics and Intelligent Text Processing. Lecture Notes in Computer Science, vol 2588.Springer, Berlin/Heidelberg
33. Pedersen T, Banerjee S, Patwardhan S (2005) Maximizing semantic relatedness to perform word sense disambiguation. Research Report UMSI 25:2005–2025
34. Rago A, Marcos C (2009) Técnicas de nlp y wsd asistiendo al desarrollo de software orientado a aspectos (in spanish). In: Argentinian symposium on artificial intelligence
35. Rago A, Abait E, Marcos C, Diaz-Pace A (2009) Early aspect identification from use cases using nlp and wsd techniques. In: Proceedings of the 15th workshop on early aspects. ACM, pp 19–24
36. Rashid A, Chitchyan R (2008) Aspect-oriented requirements engineering: a roadmap. In: Proceedings of the 13th international workshop on early aspects. ACM, pp 35–41
37. Sampaio A, Greenwood P, Garcia A, Rashid A (2007) A comparative study of aspect-oriented requirements engineering approaches. In: Empirical software engineering and measurement, 2007. ESEM 2007. First international symposium on, IEEE Computer Society, pp 166–175
38. Shepherd D, Pollock L, Vijay-Shanker K (2006) Towards supporting on-demand virtual remodularization using program graphs. In: Proceedings of the 5th international conference on aspect-oriented software development, March, Citeseer, pp 20–24
39. Siy H, Aryal P, Winter V, Zand M (2007) Aspectual support for specifying requirements in software product lines. In: Proceedings of the early aspects at ICSE: workshops in aspect-oriented requirements engineering and architecture design. IEEE Computer Society, p 2
40. Tabares M, Anaya de Páez R, Arango Isaza F (2008) Un esquema de modelado para soportar la separación y transformación de intereses durante la ingeniería de requisitos orientada por aspectos (in spanish). Avances en Sistemas e Informática 5(1):189–198
41. Tempero E (2009) Experiences in teaching quality attribute scenarios. In: Proceedings of the eleventh Australasian conference on computing education, vol 95. Australian Computer Society, Inc., pp 181–188
42. Zhang H, Ben K (2010) Architectural design of the health watch system with an integrated aspect-oriented modeling approach. In: Computer design and applications (ICCDA), 2010 international conference on, vol 1, pp V1-624–V1-628. doi:10.1109/ICCDA.2010.5540893