Can quality-attribute requirements be identified from early aspects?

QAMiner: a preliminary approach to quality-attribute mining

Alejandro Rago[1,2], Claudia Marcos[1,3], Andrés Diaz-Pace[1,2]

[1] ISISTAN Research Institute, UNICEN University, Campus Universitario
Paraje Arroyo Seco, B7001BBO, Tandil, Bs. As., Argentina
Te: +54 (2293) 439682 Ext. 42 - Fax: +54 (2293) 439681

[2] CONICET, National Council for Scientific and Technical Research
C1033AAJ, Bs. As., Argentina

[3] CIC, Committee for Scientific Research
B1900AYB, La Plata, Argentina

{arago,cmarcos,adiaz}@exa.unicen.edu.ar

Abstract. Specifying good software requirement documents is a difficult task. Many software projects fail because of the omission or bad encapsulation of concerns. A practical way to solve these problems is to use advanced separation of concern techniques, such as aspect-orientation. However, quality attributes are not completely addressed by them. In this work, we present a novel approach to uncover quality-attribute requirements. The identification is performed in an automated-fashion, relying on early aspects to guide it and using ontologies to model domain knowledge. Our tool was evaluated on two well-known systems, and contrasted with architectural documents.

Keywords: quality attribute, software requirement, crosscutting concern, early aspect, use case specification

1 Introduction

The importance of precise and complete software requirements specification has been longly recognized by the software development community [13]. In addition, quality-attribute requirements, which describe constraints on the development and behavior of a software system [3], are a key factor for the success of a software project. Achieving a good separation of concerns improves requirements and reduce problems such as refactorings in later stages [16]. Also, detecting and analyzing quality attributes in early development stages provides insights for system design, reduces risks, and ultimately improves the 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, learning quality attributes becomes usually a time-consuming task for analysts. Recent developments have made it possible to mine concerns semi-automatically from textual documents, applying state-of-the-art natural language processing and information retrieval algorithms [11]. Several approaches to identify crosscutting concerns and to encapsulate them using aspect-oriented techniques are available [2,17]. At the requirements level, early aspects are used to enclose crosscutting properties into single modular units.
Yet, while using aspects increases the separation of concerns, it does not entirely address quality attributes. Many authors have suggested the existence of a relationship between (early) aspects and quality attribute requirements [16,4], but still no work (as far as we know) has dealt with this issue. We believe that many early aspects actually derive into quality-attribute concerns (although, not every early aspect will have quality-attribute connotations). For example, the analysis of a distribution early aspect may reveal an availability quality-attribute.
This work presents a new approach to detect potential quality-attribute requirements. It uses use case specifications and early aspects (detected with another tool) as input. Domain knowledge is modeled in an ontology, which is latter used for identification purposes as well as confidence calculations.
The rest of the paper is organized as follows. Section 2 discusses related works which addressed quality attributes in requirements. Section 3 presents our previous work to improve software requirements. Section 4 introduces our approach to mine quality attributes. Section 5 demonstrates the performance of a prototype tool on two case studies and Section 6 explains the findings and potential extensions of this work.

2 Related works

A growing interest in specification strategies, modeling techniques and semi-automated approaches has emerged to improve software requirement documents. We are mainly concerned with the latter. Existing works are divided in two categories: approaches to specify/detect early aspects and approaches to specify/identify quality attributes.
Moreira et al. [12] introduce an approach to identify and specify quality attributes that crosscut software requirements. It includes a systematic integration between quality attributes and functional requirements. It defines three main activities: (i) identification, (ii) specification, and (iii) integration of requirements.
Döer et al. [8] present an approach which goal is to achieve a minimal, complete and focused set of measurable and traceable nonfunctional requirements. A meta-model is defined to support the approach. It can be instantiated total or partially, in a tailored quality model. This two approaches prescribe a series of activities to address quality attributes. However, Moreira's approach does not discriminate between crosscutting concerns (i.e., early aspects) and quality attributes, and Döer approach is very complex to be carried out and does not encapsulate crosscutting requirements correctly.
Cleland Huang et al. [7] present a automated approach to identify quality attributes in software requirement specifications. It uses a supervised classification

technique to uncover quality attributes scattered across multiple documents. After training, the classifier characterize quality attributes with keywords called indicator terms, which denotes frequent words occurring in the presence of a quality attribute. Casamayor et al. [6] present a nonfunctional classification technique, which also uses semi-supervised learning classification technique to categorize quality-attributes. Requirements are labeled accordingly to the trained classifier. Bass et al. [4] present an approach to identify early aspects using architectural reasoning. It starts defining quality scenarios and moves to design in a semi-automatic fashion, using architectural tactics and their associated frameworks. Compared to our work, Bass' approach takes the opposite direction that QAMiner, starting from quality-attribute scenarios to architectural aspects, in contrast to going from early aspect towards quality attributes.

3 Previous work

Recently, we have investigated on early crosscutting concerns identification and early aspect refactorings [14,15]. Our main objective was to automatize the detection of crosscutting concerns in software requirement specifications, particularly in use case specifications. Given the system use cases, a lexical, syntactical and semantical analysis is applied to accumulate knowledge about concerns. Several advanced natural language techniques and information retrieval algorithms are used, including semantical dictionaries. Afterward, an action-oriented identification graph [18] is built, to unify and ease the concepts occurring in the use cases. In detail, the graph detaches verbs and direct objects placed in the same sentence, and connect them with arcs. Furthermore, a semantical clustering technique is executed to alleviate issues like synonyms and ambiguities (inherent to natural-language writing). Finally, this graph is transversed to look for multiple occurrences of the same behaviors. The output of this approach are sets of semantically-related concerns that are scattered in different documents (that is to say, crosscutting). Each set is composed with many pairs of <verb, direct object> which represent a particular concern. We developed a working prototype, called SAET (Semantic Aspect Extractor Tool), which assist requirement engineers during the crosscutting concern identification process, allowing them to provide feedback and to correct false positives.

4 QAMiner approach

A strategy to improve the understanding of a system, is to identify crosscutting concerns in software requirements specifications. That way, concerns are correctly encapsulated (in early aspects) and we benefit for having a good separation of concerns. However, those crosscutting concerns may still be linked somehow to quality attributes of the system. Our approach, called QAMiner (Quality-Attribute Miner), aims at identifying and uncovering hidden quality attributes in software specifications, making explicit the relationship between early aspects and quality attributes.
QAMiner takes as input information from requirements documents (particularly, use case specifications) and early aspects of a system previously detected

with SAET. A deep analysis is performed over use case specifications and early aspects, using as knowledge source a defined quality-attribute ontology. This ontology it is bounded to software qualities attributes and scenarios domain, built and maintained by software architects using their expertise in software development. With its help, QAMiner is able to determine precise associations between concerns of the system and quality attributes. QAMiner generates as output a map of <quality attribute, confidence>, in which "quality attribute" stands for a particular quality attribute of the ontology and "confidence" stands for a numerical value representing the belief about the relationship between the particular quality attribute and the concern.

Fig. 1: QAMiner activities

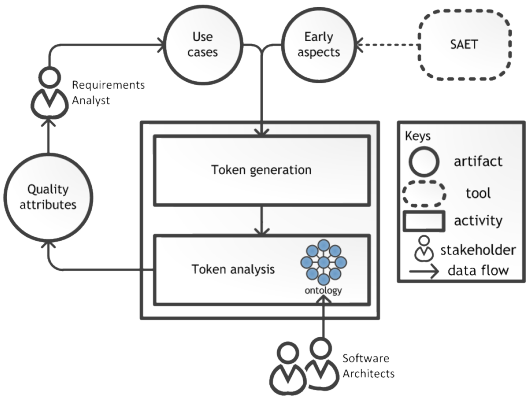


Figure 1 shows the flow of activities executed by QAMiner to detect quality attributes. The workflow is divided in two main stages. The first, Token generation, deals with preprocessing algorithms to reduce noise in the input text (e.g., eliminating stop-words) and the generation of annotations of interest over text (such as weighted tokens). The second, Token analysis, handles the association between the words processed in the previous stage and the quality attributes defined in the ontology. To weight how related this words are, QAMiner leverage on instances loaded in the ontology to measure percentage rates.
There are two steps in which stakeholders interact with QAMiner. To define and maintain complete and minimal quality-attribute ontology and to load representative instances. After the analysis is done and a set of quality attributes is found, those are presented to requirement analysts. It is their responsibility to determine if the candidates are actual manifestations of quality attributes and to establish changes in requirement documents.
In the following subsections, these stages are described in detail.

4.1 Token generation

This stage is responsible of processing the input of QAMiner, that is, the use case specification documents and the early aspects identified with SAET. The main goals of Token generation are the extraction of tokens from inputs, the application of filters to remove noisy information, and the augmentation of tokens using attributes.
QAMiner starts by collecting words from use cases and early aspects and encapsulating them into tokens. A token is a basic unit of text. Then, tokens are preprocessed and transformed using a simple format. Each token can have several attributes attached. Attributes have the form of <attribute, value> (Table 1). Using attributes allows independent transformations to be applied using a pipe-and-filter style [3]. Each filter is a processing unit that produces modifications (e.g., augmentation, refinement or transformations) over tokens and their attributes. QAMiner utilizes five filters. Each of these filters performs the following functions: (i) *lower case*, transforms the word into lower characters, (ii) *stop words*, removes non-useful words in information retrieval activities, such as articles and prepositions, (iii) *stemmer*, transforms the words to its root form, (iv) *weighting*, assigns values to tokens according to their location in documents, (v) *occurrences*, augments tokens with statistical data, precisely with their count number in the documents.

Table 1: Token attributes

Attribute	Description	Value	Example
id	unique token identifier	identifier number	<id, 1001>
kind	originating document	use case or early aspect	<kind, use case> <kind, early aspect>
section	token location in document	brief description, basic flow, etc.	<section, basic flow> <section, alternative flow> <section, special requirements> <section, verb-dir.object pair>
occurrences	token count	count number	<occurrences, 3> <occurrences, 8>
weight	scoring by relevance	score number	<weight, 1> <weight, 5>

For example, let's suppose that the word "stored" is retrieved from a use case. The attribute <id, 2000> is generated. Because the word is located in the basic flow of a use case, attributes <kind, use case> and <section, basic flow> are generated. The lower case filter does not modify the word, because it is already in lower case. The stop words filter does not remove the token, because it is a relevant word. The stemmer filter reduces the word "stored" to its root, "store".

The occurrences filter counts two times this token in the documents and generates the attribute <occurrences, 2>. Finally, using a preestablished weighting scheme, the attribute <weight, 4> is generated.

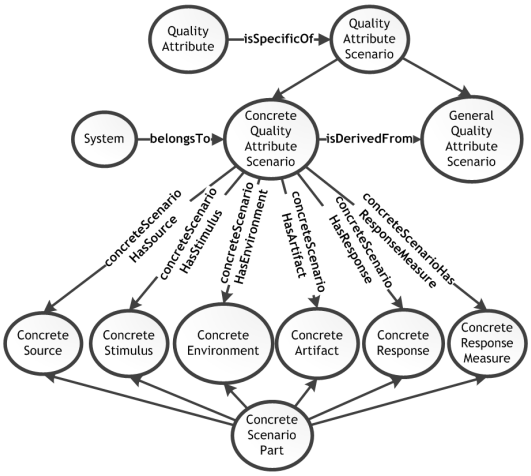
4.2 Token analysis

This stage carries out the analysis necessary to identify potential quality-attributes from the tokens previously extracted. It consist of two main activities. The first activity calculates a confidence value for each token with a particular quality attribute. This confidence represents the membership of a token to a quality attribute. The calculation is performed relying on a domain-defined ontology and exploring the relationships found in it. The second activity builds a map for each early aspect and their associated use cases, summarizing the confidence values of its tokens for each quality attribute.
An *ontology* is a data model that describes *concepts* (also known as classes) in a specific domain, *properties* that describe different characteristics of a concept, and *constraints* over these properties. In addition, *instances* are particular examples of domain information, represented using the concepts of ontology. Our approach builds upon a quality-attribute ontology which models quality attributes and their corresponding scenarios (see Figure 2). The construction rationale of the ontology followed definitions explained in [3]. Many concepts were modeled, like quality attribute and quality-attribute scenarios (both generals and concretes), as well as parts of concrete scenarios, such as source, stimulus, response, among others. Multiple instances of quality-attribute scenarios were loaded. These were define by experimented software architects, using as source of information several internal projects. In this work, we modeled six quality attributes and their corresponding scenarios: Availability, Modifiability, Security, Usability, Performance and Testability. For example, Figure 3 shows instances of a Modifiability and a Availability quality-attribute scenario. Filled boxes denotes instances of concepts, like in 3a "be in degraded mode" is an instance of the concept ConcreteResponseMeasure.

The association of an early aspect and quality attributes is performed by matching its composing tokens with instances of the ontology. That is, for each token the approach finds whether a token matches with parts of concrete scenarios and whose quality attributes are described by them. Matchings between tokens and parts of scenarios are computed using pattern matching techniques and taking advantage of the preprocessing algorithms applied in the Token generation stage. If a token matches more than one part of a single scenario, disambiguation heuristics are applied.

The calculation of the confidence values of a token is carried out as follow. First, QAMiner infers those scenarios related to the matching part. Second, using the relationship "isSpecificOf" of the ontology, the approach counts down how many times those scenarios participate in a quality attribute, and determine a membership percentages for each quality attributes. For example, a token "latency" is found to be a ResponseMeasure part. Exploring the ontology, it is determined that 40 quality-attribute scenarios are related to that particular instance of the

Fig. 2: Quality-attribute ontology



part. Of the 40 scenarios, 30 are associated to a Performance quality attribute, 8 to Availability and 2 to Modifiability. Thus, QAMiner calculates with 75% of confidence that token "latency" is a Performance quality attribute, 20% Availability and 5% Modifiability.
Percentages are adjusted accordingly to the occurrences and weight attributes generated previously. Then, the relationship of an early aspect with each quality attribute is determined as the average of the adjusted confidence values of its composing token. QAMiner generates suggestions selecting those quality attributes with the highest confidence difference(s). This means that, for a single early aspect, it is possible to detect more than one quality attribute associated.

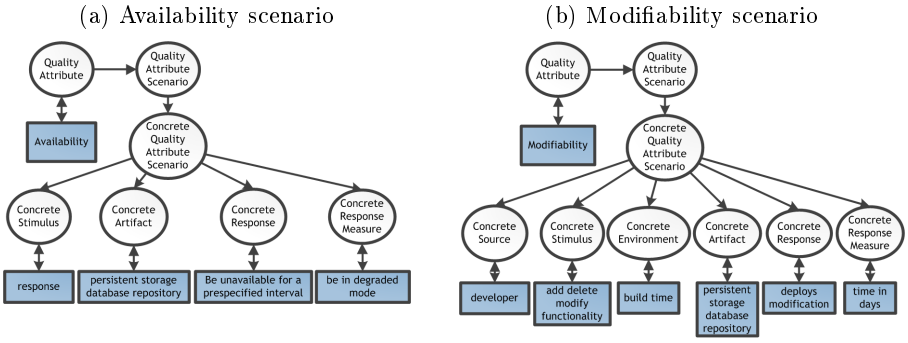
5 Preliminary evaluation

To evaluate the predicting performance of our approach, we run QAMiner over two case studies. The first is the Health Watcher System (HWS) [9,10] and the second the Course Registration System (CRS) [5] To determine the ability to retrieve relevant quality attributes, we use measures from the Information Retrieval area. The results were validated using architectural documents [20,19] of the HWS and design drafts of the CRS [5].

5.1 Case studies

The HWS is a typical web information system for a city's health care program. It allows online access to register complaints, read health notices, and query regarding health issues. The reason for choosing the HWS is twofold. First, requirement and design documents are available. Second, several researchers have

Fig. 3: Quality-attribute scenarios in the ontology



used this case-study and analyzed its quality-attribute properties. This system counts with 9 use case specifications. The CRS is a distributed system to be used within a university intranet. Some artifacts were available, such as use case specification, some analysis classes and unfinished design documents. It allows students to apply to courses and obtain their grade reports, and professors can register new courses and report student grades. This system counts with 8 use case specifications. We detected the following early aspects in HWS and CRS use case specifications, respectively (Table 2). The tests conducted use these early aspects as input of QAMiner.

Table 2: Early aspects

	Health Watcher System	Course Registration System
Early aspects	Data Formatting	Persistency
	Persistency	Entitlement
	Consistency	Communication
	Distribution	Ease-of-use
	Access Control	Data Validation
	Error Handling	Data Processing

5.2 Evaluation framework

We analyze QAMiner results in a similar way to classification tasks. The empirical evaluation of the approach used measures such as *Recall*, *Precision* and *Accuracy* [1], adapted to this particular domain and problem. Particularly, several binary parameters were collected during tests. *True positives* (TP) refer to quality attributes suggested by QAMiner for an early aspect that are real quality

attributes of the system. *False positives* (FP) are those suggestions which are not real quality attributes of the system. *True negatives* (TN) are those quality attributes not suggested for an early aspect which were real quality attributes. And *False negatives* (FN) are those quality attributes not related to an early aspect which were not suggested. Figure 4 depicts the formulas to calculate *Recall*, *Precision* and *Accuracy*. *Precision* can be seen as a measure of exactness or fidelity (i.e., how much of the quality attributes identified are correct), whereas *Recall* is a measure of completeness (i.e., how much of all the existing quality attributes are detected). *Accuracy* is similar to *Recall*, but it also takes into account the non-detection of incorrect quality attributes.

Fig. 4: Evaluation formulas

$$Recall = \frac{TP}{TP+FN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

5.3 Evaluation results

To determine the binary parameters, the suggestions of QAMiner were contrasted with the architectural and design documents of both case studies [9,10,20,19,5]. From these documents, several quality attributes were collected. Table 3 illustrates these quality attributes.

Table 3: Real quality attributes

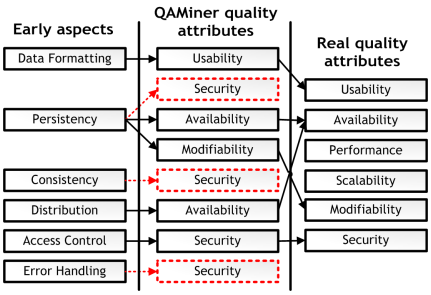
	Health Watcher System	Course Registration System
Quality Attributes	Usability Availability Performance Scalability Modifiability Security	Modifiability Availability Security Usability Performance

After executing our working prototype over the Health Watcher System and the Course Registration System, QAMiner generates suggestions of quality attributes for each early aspect. Figure 5 depicts the rationale to calculate TP, FP,

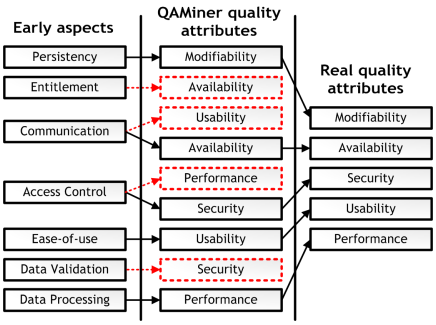
TN, and FN for both HWS (5a) and CRS (5b).

Fig. 5: Evaluation rationale

(a) Rationale for HWS



(b) Rationale for CRS



The first column holds the input early aspects, the second column, QAMiner's suggested quality attribute(s), and the third column, the real quality attributes of the system. From an input early aspect of the first column, QAMiner suggest those quality attributes of the second column which are connected with an arrow. If the suggestions happens to be correct, then a connecting arrow between the predicted quality attribute and the real quality attribute is drawn. If not, the predicted quality attribute in the second column is drawn with dashed lines. In Figure 5a for example, QAMiner suggest three quality attributes for the Persistency early aspect: Security, Availability and Modifiability. Of the three suggestions, two are real quality attribute of the system (Availability and Modifiability) and thus are connected to the quality attributes of the third column, while the other (Security) is not a real quality attribute thus it has a dashed outline.

QAMiner showed promising performance to suggest correct quality attributes (Table 4). In HWS[4], it obtained above 70% of *Recall*, which is high enough due to the lack of instances in the ontology to detect scalability. In CRS, it obtained a 100% of *Recall*, because all quality attributes were found. This outcome is really important, because it means that QAMiner has a wide coverage of the quality attributes of a system. Regarding *Precision*, it was not as good as *Recall*, obtaining approximately a 60% in both case studies. *Accuracy* obtained high scores, of about 90%. This last measure means that not only QAMiner detects quality attributes, but also ignores correctly those quality attributes which are not related to an early aspect.

[4] Because one of the predicted quality attributes was found two times, we use 4 instead of 5 true positives to calculate *recall* and *accuracy* in this case study.

Table 4: Results

	Health Watcher System	Course Registration System
Real QAs	6	5
True positives	5	5
False positives	3	4
False negatives	2	0
True negatives	28	33
Precision	0,625	0,555
Recall	0,714	1,000
Accuracy	0,865	0,905

6 Conclusion

In this work we present a novel approach which takes advantages from early aspects to detect potential quality attributes. Our driven hypothesis was that early aspects were a good source of information to start looking for quality attributes of a system. A ontology of quality scenarios was built and used to uncover words of early aspects related to a particular quality attribute.
A preliminary evaluation of this approach produced promising results, with high *Recall* and *Accuracy* measurements. Moreover, we were able to validate the evaluation using architectural information from research peers.
The main contribution is two-fold. First, we corroborate empirically that many early aspects derive into quality attributes. Second, we develop a prototype which feed from early aspects and suggest quality attributes in a semiautomatic fashion.
However, there is still room for improvements in our tool. We look forward to add more quality attributes and even more instances to the ontology, to increase and *Precision* and *Recall*. In addition, we count with several case studies to continue evaluations and validation of the approach.

Acknowledgments. We would like to thanks to Francisco Bertoni and Sebastian Villanueva, which developed QAMiner and volunteer to carry out the evaluation test, under our supervision.

References

1. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York. (1999)
2. Baniassad, E., Clarke, S.: Finding aspects in requirements with theme/doc. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design p. 16 (2004)
3. Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley Longman Publishing Co., Inc. (2003)

4. Bass, L., Klein, M., Northrop, L.: Identifying aspects using architectural reasoning. Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design p. 51 (2004)
5. Bell, R.: Course registration system. http://sce.uhcl.edu/helm/RUP_course_example/courseregistrationproject/indexcourse.htm (2010)
6. Casamayor, A., Godoy, D., Campo, M.: Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. Information and Software Technology 52(4), 436–445 (2010)
7. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: Automated classification of non-functional requirements. Requir. Eng. 12, 103–120 (May 2007), http://portal.acm.org/citation.cfm?id=1269901.1269904
8. Doerr, J., Kerkow, D., Knethen, A.v., Paech, B.: Eliciting efficiency requirements with use cases (2003)
9. Greenwood, P.: Tao: A testbed for aspect oriented software development. http://www.comp.lancs.ac.uk/~greenwop/tao/ (2010)
10. Khan, S., Greenwood, P., Garcia, A., Rashid, A.: 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 (2008)
11. Kof, L.: Natural language processing: mature enough for requirements documents analysis? Natural Language Processing and Information Systems pp. 91–102 (2005)
12. Moreira, A., Araújo, J.a., Brito, I.: Crosscutting quality attributes for requirements engineering. In: Proceedings of the 14th international conference on Software engineering and knowledge engineering. pp. 167–174. SEKE '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/568760.568790
13. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proceedings of the Conference on The Future of Software Engineering. pp. 35–46. ICSE '00, ACM, New York, NY, USA (2000), http://doi.acm.org/10.1145/336512.336523
14. Rago, A., Abait, E., Marcos, C., Diaz-Pace, A.: Early aspect identification from use cases using nlp and wsd techniques. In: Proceedings of the 15th workshop on Early aspects. pp. 19–24. ACM (2009)
15. Rago, A., Marcos, C.: Técnicas de nlp y wsd asistiendo al desarrollo de software orientado a aspectos. In: Argentinian Symposium on Artificial Inteligence (2009)
16. Rashid, A., Chitchyan, R.: Aspect-oriented requirements engineering: a roadmap. In: Proceedings of the 13th international workshop on Early Aspects. pp. 35–41. ACM (2008)
17. Sampaio, A., Chitchyan, R., Rashid, A., Rayson, P.: Ea-miner: a tool for automating aspect-oriented requirements identification. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. pp. 352–355. ACM (2005)
18. Shepherd, D., Pollock, L., Vijay-Shanker, K.: Towards supporting on-demand virtual remodularization using program graphs. In: Proceedings of the 5th international conference on Aspect-oriented software development, March. pp. 20–24. Citeseer (2006)
19. Tabares, M., Anaya de Páez, R., Arango Isaza, F.: Un esquema de modelado para soportar la separación y transformación de intereses durante la ingeniería de requisitos orientada por aspectos. Avances en Sistemas e Informática 5(1), 189–198 (2008)
20. Zhang, H., Ben, K.: 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 (2010)