ObasCId: An Ontologically-Based Approach for Concern Identification and Classification

Paulo Afonso Parreira Júnior
Department of Computer Science
Federal University of Lavras
Lavras/MG, Brazil
pauloa.junior@dcc.ufla.br

Rosângela Aparecida Dellosso Penteado
Department of Computer
Federal University of São Carlos
São Carlos/SP, Brazil
rosangela@dc.ufscar.br

***Abstract* —The Aspect-Oriented Requirements Engineering (AORE) area intends to provide more appropriate strategies for software concern identification, classification (as crosscutting or non-crosscutting) and modularization, in the early phases of software development cycle. A commonly reported issue about the existing AORE approaches is the lack of appropriate resources (guidelines, processes, catalogs, among others) to support software engineers during the concern identification and classification. This work aims to mitigate this issue by proposing: (i) a reference ontology for the software concerns domain, called *O4C* (*Ontology for Concerns*); and (ii) an ontologically-based approach for AORE, called *ObasCId* (*Ontologically-based Concern Identification and Classification*), that suggests the usage of catalogs of software concerns and a well-defined process for supporting software engineers to perform these activities in a more systematic way. An experimental study was performed on *ObasCId* and its results indicated that this approach may positively contribute for the concern identification and classification effectiveness without harming its execution time**.

Keywords - Crosscutting Concerns; Early-Aspects; Aspect-Oriented Requirements Engineering; Concern Identification and Classification

I. INTRODUCTION

In the context of Requirements Engineering (RE), a concern can be understood as a set of software requirements related the same purpose [9]. Two types of concerns are functional concerns and non-functional concerns. The first one refers to concerns that are related to functional features of the software, such as "Payment" and "Order Management". The last one corresponds to concerns related to non-functional features of the software, such as "Security", "Persistence", and "Logging". Several traditional RE approaches, such as those based on viewpoints, goals, use cases and scenarios have been developed in order to allow the modularization of software concerns in an appropriate way [27]. However, there are some types of concerns that may not be easily modularized, even in the early phases of software development cycle. These concerns are known as *CrossCutting Concerns* or *Early-Aspects* and consist of software concerns whose requirements are spread over requirements of other software concerns [29]. For instance, a security concern may contain requirements related to the encryption and/or authorization properties. These requirements, for instance, may affect some requirements related to "Orders Management" concern.

The non-identification of the software concerns, especially the crosscutting ones, may bring difficulties for the software development and evolution processes, harming the reasoning of the software engineer on the effects caused by the inclusion, removal or update of a requirement over the other ones [9]. The *Aspect-Oriented Requirements Engineering* (AORE) area deals with software concerns during the early phases of software development [8][27], in order to identify, classify (as crosscutting or non-crosscutting), modularize and compose these concerns in a more appropriate way.

Some experimental studies performed on the main AORE approaches [15][28] have pointed out concern identification and classification as bottleneck activities. One of the possible causes of this is the *lack of understanding about the software concerns domain*: there are few studies designed to provide a clear understanding about the software concern concepts, aiming to answer questions such as "which are the main properties of a concern?", "how does a concern affect other software concerns", among others. The knowledge about software concerns domain is spread in different studies, sometimes in a divergent way, what may hind the understanding of researchers and practitioners. Another possible cause is the *lack of appropriate resources (guidelines, processes, catalogs, among others) to support software engineers during the concern identification and classification* [23][24]: several AORE approaches rely only either on the software engineers' expertise or on the usage of keywords for the correct identification of software concerns; in our understanding, this may decrease the effectiveness of these approaches. Section II presents more details about these causes, taking the related works into consideration.

In this context, this work aims to improve the effectiveness of the concern identification and classification activities by dealing with the above mentioned causes. To do this, we propose: (i) a reference ontology for the software concerns domain, called *O4C* (*Ontology for Concerns*), that aims to make clear and precise the description of the concepts of this domain; and (ii) an ontologically-based AORE approach, called *ObasCId* (*Ontologically-based Concern Identification and Classification*), that provides more appropriate resources (catalogs, heuristics, processes) for supporting software engineers during the concern identification and classification. The assessment performed on the *ObasCId* approach provided results that lead us to believe that the usage of this approach may improve the recall of the concern identification and classification, without negatively impacts on the precision and the execution time of these activities.

This paper is organized as follows: Section II presents a discussion about the related works; in Sections III and IV, the *O4C* ontology and the *ObasCId* approach are, respectivey, presented. A description of an experimental study performed on the *ObasCId* approach is in Section V; and, finally, Section VI highlights the final remarks and proposals for future works of this paper.

II. RELATED WORKS

Several AORE approaches have been proposed in last years, especially, for concern identification and classification [23][24]. Several approaches [1][2][6][7][8][19][20][29]

978-1-5090-5086-4/16 $31.00 © 2016 IEEE
DOI 10.1109/SBCARS.2016.10

[31][33] suggest the usage of catalogs of Non-Functional Requirements (NFR catalogs), such as those proposed by Boehm and In [4], Chung and Leite [10] and Cysneiro [12], for aiding software engineers while performing the concern identification and classification activities.

The usage of NFR catalogs in the AORE context is not totally appropriate, because these catalogs are not prepared for the software concerns domain and fail to consider some specific properties of this area. For example, they do not contain information about functional requirements and their relationships. According to Moreira *et al*. [29], functional requirements also may cut-across other software requirements, hence, it is important to consider them during the concern identification and classification. Furthermore, although these approaches suggest the usage of NFR catalogs, they do not present guidelines or processes that indicate how to use them in an appropriate way. In this case, the quality of the results provided by these approaches are widely dependent on the software engineers' expertise.

In other approaches [3][11][27], none resources, such as NFR catalogs, are provided to aid software engineers during the concern identification and classification. Instead, they only suggest the usage of keywords, previously identified by the software engineer from the requirements document, as inputs for the concern identification and classification activities. The main drawback of this strategy is that it does not consider the existence of implicit concerns, *i.e*., concerns that emerge from the existence of other software concerns and are not explicitly mentioned in the requirements document, by means of keywords. For instance, if the software requires a good performance to persist its data, a possible strategy is using concurrency mechanisms, such as connection pooling. Hence, as mentioned in the work of Sampaio *et al*. [28], "Concurrency" is an implicit concern, observed from the existence of two other concerns in the same software: "Persistence" and "Performance".

As stated in the introduction of this paper, the knowledge about software concerns domain is spread in different studies, sometimes in a divergent way. Most of the existing AORE approaches represents the knowledge about software concerns in XML files (templates), developed by the authors of these approaches. These templates are usually presented without the meta-model that describe them and do not share the main concepts and relationships existing in the software concern domain. For instance, the template proposed by Moreira *et al*. [19] does not provide information about the source(s) from which a concern was described, such as a stakeholder, a business document, among others. However, this information can be found in templates of other AORE approaches [1][6][31].

This work differs to those above mentioned, because it: (i) proposes a conceptual model (*O4C* ontology) for the software concerns domain, aiming to make clear and precise the description of the concepts of this domain; (i) proposes the usage of software concerns catalogs as inputs for the concern identification and classification activities, aiming to provide more useful information for aiding the software engineers to perform these activities; (ii) provides a set of activities and heuristics to guide software engineers while using the software concern catalogs; and (iii) suggests that the existing relationships among software concerns and requirements may be used, along with the keywords, to improve the effectiveness of the concern identification and classification activities, especially, for the implicit concerns.

Regarding to the usage of ontologies in the RE area, a systematic mapping conducted by the authors of this paper [26] presented that there are several ontology-based approaches for this area. However, none was specific to the context of AORE. One of the closest works related to this paper is that one proposed by López *et al.* [17]. In this work, the authors presented an ontology for sharing and reusing NFR and design decisions. The proposed ontology aims to store the knowledge related to the NFR and design decisions, based on the description of NFR catalogs. Hence, the researcher may create instances from this ontology that address the NFR and design decisions of his/her interest.

The work of López *et al.* is different from the proposal of this paper, because: (i) their work is not related to the AORE area, therefore, it does not address specific properties of the software concern domain, such as the classification of a concern as non-functional or functional one, the relationships between concerns and their keywords, the decomposition of concerns into sub-concerns, among others; (ii) their work does not present a set of activities or guidelines that helps software engineers on how to use the proposed ontology instances; and (iii) the work does not present any type of an experimental study on the proposal.

III. *ONTOLOGY FOR CONCERNS (O4C)*

Software concerns are the focus of the AORE area, hence, it is important to understand: (i) which are the main concepts regarding to this subject?; (ii) which are the relationships between these concepts?, among others. Providing answers to these questions may minimize the negative impacts of the issue discussed in the introduction of this paper. A well-defined understanding of the software concerns domain may also allow the researchers and practitioners to build AORE methods, techniques and tools that may be widely used, since they are based on shared definitions of this domain.

To do this, a reference ontology for the software concern domain, called *O4C* (*Ontology for Concerns*) was proposed. *Reference ontology* is a special kind of conceptual model, which aims to make clear and precise the description of a domain with the purpose of communication, learning and problem solving [14]. The development of *O4C* considered: (i) the existing works regarding to AORE, gathered by the authors of this paper from a systematic mapping of literature [23][24]; and (ii) the expertise of two researchers that have worked with AORE for 12 (twelve) years. Moreover, the *O4C* ontology was developed in accordance with: (i) the approach for ontology development, called *SABiO* (*Systematic Approach for Building Ontologies*) [14]; and (ii) an UML profile for ontology modelling, called *OntoUML* [15]. A preliminary version of this ontology was proposed in [21][22]; in this paper, the final version of *O4C* is formalized and described according to *SABiO* and *OntoUML*.

The graphical model of *O4C* ontology is presented in Figure 1 and its concepts and relationships are commented in this section.

A. *Concern, FunctionalConcern and NonFunctionalConcern*

The "Concern" concept represents individuals that meet the properties of a software concern (these properties are discussed in this section). Two subtypes of this concept are "FunctionalConcern" and "NonFunctionalConcern". "FunctionalConcern", as mentioned in the section of this paper.

The "Concern" class is stereotyped with «Kind» and their subclasses have the «Subkind» stereotype. In accordance to *OntoUML* [15], these stereotypes correspond to rigid concepts, which means that instances of these concepts will continue to be so as long as they exist. For example, "Person" is a rigid concept because if "John" is an instance of "Person", then it always will be as long as it exists. The difference between the "Kind" and "Subkind" concepts is that the first one provides the principle of identity to its instances and the second one only inherits the principle of another concept. For example, considering the fingerprint as the principle of identity provided by the "Person" concept to its instances, then "Man" and "Woman" are "Subkinds" concepts, because they inherit the identity principle of "Person".

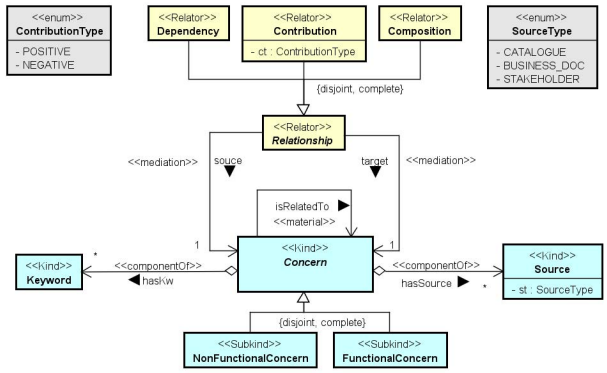


Figure 1. Ontology for Concerns (O4C)

The "Concern", "FunctionalConcern" and "NonFunctionalConcern" concepts are well-known in the AORE community and are reported in several studies [1][2] [3][6][7][8][11] [20][27][29][31][33].

B. Keyword and Source

The "Keyword" concept appears in some AORE approaches [3][11][27], however, none of the analyzed works presented the idea of store these keywords in order to use them in other projects. In the *O4C* ontology, this concept was created aiming to store the keywords commonly used to identify a particular concern. For example, "save", "update" and "persist" may be used to provide indications of the existence of the "Persistence" concern.

The idea represented by "Source" class, its "st" attribute and the "SourceType" enumerated class, refers to the possible sources from which the description of a software concern may be extracted. A software concern may be related to several sources and they are important in the concern identification and classification process, because they can help the software engineer to identify who or what need to be consulted when a particular concern is not being correctly identified.

According to Agostinho *et al*. [1], Brito and Moreira [6] and Whittle and Araújo [31], the possible source types are: (i) *stakeholders*, for example, a project manager, an expert in security, among others; (ii) *NFR catalogs*, such as those proposed by Boehm and In [4], Chung and Leite [10], Cysneiro [12], among others; or (iii) *business documents*, such as a security protocol of a company.

C. Contribution, Dependency and Composition

The possible types of concern relationships are represented by the "Contribution", "Dependency" and "Composition" concepts (sub-concepts of "Relationship"). These classes were stereotyped with «Relator». In *OntoUML*, "Relators" are mediator elements, *i.e.*, elements that mediate the relationship among other ones, making it real. In Figure 1, it is possible to notice a relationship, called "isRelatedTo", stereotyped with «Material». "Material" relationships are applied to relations that depend on a mediator element to exist. For example, the "married to" relationship is only valid while a "marriage" (relator) exists. In the same way, the "isRelatedTo" relationship is only valid while a "Relatioship" (relator) between two concerns exists.

It is also important to highlight the two relationships stereotyped with «Mediation», called "source" and "target". According to *OntoUML*, "Mediation" is a type of relationship that binds the "Relator" to the elements whose relationship is mediated by it. In this case, these relationships describe what are the *source* and the *target* of a concern relationship.

The type of relationship addressed by the concept "Composition" describes the idea of decomposition of a concern into sub-concerns. This concept is important, because a given concern may be too wide and reducing its granularity may facilitate the reasoning of the software engineer on which concerns are really present in the software and which are the appropriate strategies for modularizing them. For instance, the "Security" concern can be decomposed into sub-concerns such as "Authorization", "Encryption", among others. There may be the "Authorization" sub-concern in a specific software, but not the "Encryption" sub-concern.

"Dependency" class defines a dependency relationship between two concerns. This means if an "A" concern (source) depends on "B" (target) and "A" appears in the software requirements document, then "B" need to be there too. This type of information is important because: (i) it allows the software engineer to explore other concerns, before unrecognized by him/her, *i.e*., by saying that "A" depends on "B", he/she should also look for keywords related to "B" concern in the requirements document; and (ii) it allows the software engineer to verify inconsistencies in the requirements document, *i.e*., if a concern "A" depends on "B" and "B" is not described in the software requirements, then the requirements document may be inconsistent.

"Contribution" class represents a mutual influence between different concerns. A contribution can be "Negative" or "Positive", as defined by the "ContributionType" enumeration and the "ct" attribute of the "Contribution" class. An example of contribution is that existing among "Concurrency", "Performance" and "Cost" concerns. The implementation of concurrency mechanisms in the software can positively contribute to the software performance, but not to the project cost.

The "Contribution", "Composition" and "Dependency" concepts are presented quite divergently in the related works. The idea represented by the "Contribution" concept is reported in the approaches proposed by Moreira *et al*. [19] and Soeiro *et al*. [29]. However, in both approaches, the usage of this concept is limited to the project under analysis and there are no guidelines clearly indicated by the authors about how to reuse this knowledge in other projects. The approach proposed by Moreira *et al*. [19] also provides a XML file (template) responsible for specifying the relationships among different concerns, however, this template does not differentiate the types of possible relationships, such as dependency, composition, among others. The "Dependency" concept was found only in the

approach proposed by Soeiro *et al*. [29] and the "Composition" concept was not found in the analyzed works.

In all cases discussed above, the mentioned approaches do not report how the information on the concern relationships may be useful in the process of concern identification and classification. Hence, the adequate application of this information is highly dependent on software engineers' expertise.

IV. *ObasCId* Approach

ObasCId is an ontologically-based AORE approach that proposes a set of activities and heuristics for concern identification and classification from software requirements. The "ontologically-based" expression refers to the fact that *ObasCId* takes the concepts of the *O4C* ontology into account in its conception. The *ObasCId* approach consists of the following phases: (i) Preparing the Catalog of Software Concerns; (ii) Preparing the Requirements Document; and (iii) Performing Concern Identification and Classification.

A. Preparing the Catalog of Software Concerns

This phase has the responsibility of obtaining, preparing or updating a catalog of software concerns to be used in other phases of *ObasCId* approach. By using the concepts defined in the *O4C* ontology, it is possible to store the existing knowledge about specific types of concerns, generating catalogs of software concerns. For example, *O4C* ontology describes the "NonFunctionalConcern" concept, hence, in an *O4C*-based catalog, there will be instances of non-functional concerns, such as "Security", "Persistence", "Logging", among others.

Catalogs of software concerns may be generated from: (i) NFR catalogs, such as those proposed by Boehm and In [4], Chung and Leite [10] and Cysneiro [12]; (ii) the knowledge of experts on AORE; (iii) business documents, such as security and privacy protocols, pattern language, such as the language for Business Resource Management [5], among others; or (iii) historical data of previous projects.

Figure 2 and Figure 3 present two examples of *O4C*-based catalogs. In the catalogs, the stereotypes refer to *O4C* concepts and the classes represent instances of these concepts. Figure 2 shows a part of the catalog generated from historical data of the software *Health Watcher* [13]. *Health Watcher* is an information system that aims to record complaints regarding to health area. The concerns of this software were identified and classified by experts in AORE and health domains.

The proposed catalog presents seven non-functional concerns, related to twenty-eight keywords, and three functional concerns, related to six keywords. In addition, there are two contribution relationships (a positive contribution between "Concurrency" and "Performance" and a negative contribution between "Security" and "Performance") and two composition relationships, between "Complaint" and "AnimalComplaint" and "Complaint" and "FoodComplaint". The idea is this catalog may be used for the identification and classification of non-functional concerns in other software projects.

The catalog of Figure 3, in turn, was built from the concepts represented in a pattern language, called Business Resource Management [5]. This pattern language was designed to assist the development of information systems in the business resource management domain. This catalog has seven functional concerns, seventeen keywords and five relationships: (i) three compositions between the concerns "Transaction" and "Rental", "Transaction" and "Commercialization" and "Transaction" and "Reservation"; and (ii) two dependencies between the concerns "Payment" and "Transaction" and "Delivery" and "Payment".

By combining the non-functional concerns of the catalog presented in Figure 2 with all concerns of the catalog of Figure 3, it is possible to generate a broader catalog that may be used to identify both functional and non-functional concerns of information systems related to business resource management domain. This strategy was used in the experimental study presented in Section V. Section IV.C of this paper present how to use a software concern catalog, such as those previously presented, in order to identify and classify software concerns from requirements documents.

B. Preparing the Requirements Document

This phase allows the software engineer to obtain/prepare/update the requirements document on which will occur the concern identification and classification. The template used to represent the software requirements in the *ObasCId* approach is based on a list of software requirements that contains (for each requirement): (i) the requirement identifier; (ii) the requirement type (functional or non-functional); (iii) a plain-text description; and (iv) a list of other requirements on which it depends. All these information are needed to improve the quality of the concern identification and classification results, as may be explained later in this paper.

Table 1 illustrates a part of the requirements document of *Health Watcher* [13], according to the model described above. In this example, there are two non-functional requirements ("NFR-01" and "NFR-02") and one functional requirement ("FR-01"). In addition, the functional requirement depends on the other two requirements. The full requirements document of *Health Watcher* can be found in [13].

Table 1. Part of the Health Watcher requirements document.

Identifier	Type	Requirement Description	Depends on
FR-01	FR	It allows the state of a complaint to be updated. The complaint must be registered and have the OPENED state.	NFR-01, NFR-02
NFR-01	NFR	The system should have an easy to use GUI, as any person who has access to the internet should be able to use the system. The system should have an on-line HELP to be consulted by any person that uses it.	-
NFR-02	NFR	The response time must not exceed 5 seconds.	-
Legend: Functional Requirement (FR); Non-functional Requirement (NFR)			

C. Performing Concern Identification and Classification

This phase aims to identify and classify the existing concerns of the software from the catalog and the requirements document prepared in the previous phases. This phase is divided into (Figure 4): (i) Identifying concerns from keywords; (ii) Identifying concerns from the interdependence among software requirements; (iii) Specifying main concerns; (iv) Verifying the results of concern identification; and (v) Classifying concerns.

1) Identifying concerns from keywords

This activity aims to identify the software concerns from the software requirements document. This is done by searching for the keywords of each concern presented in the catalog on the description of the software requirements. If any keyword (it is important to take into consideration the grammatical variations of the keyword, such as plural forms, verb conjugation, among others) of a particular concern is in the description of a software requirement, it is stated that this concern is related to the requirement in analysis.

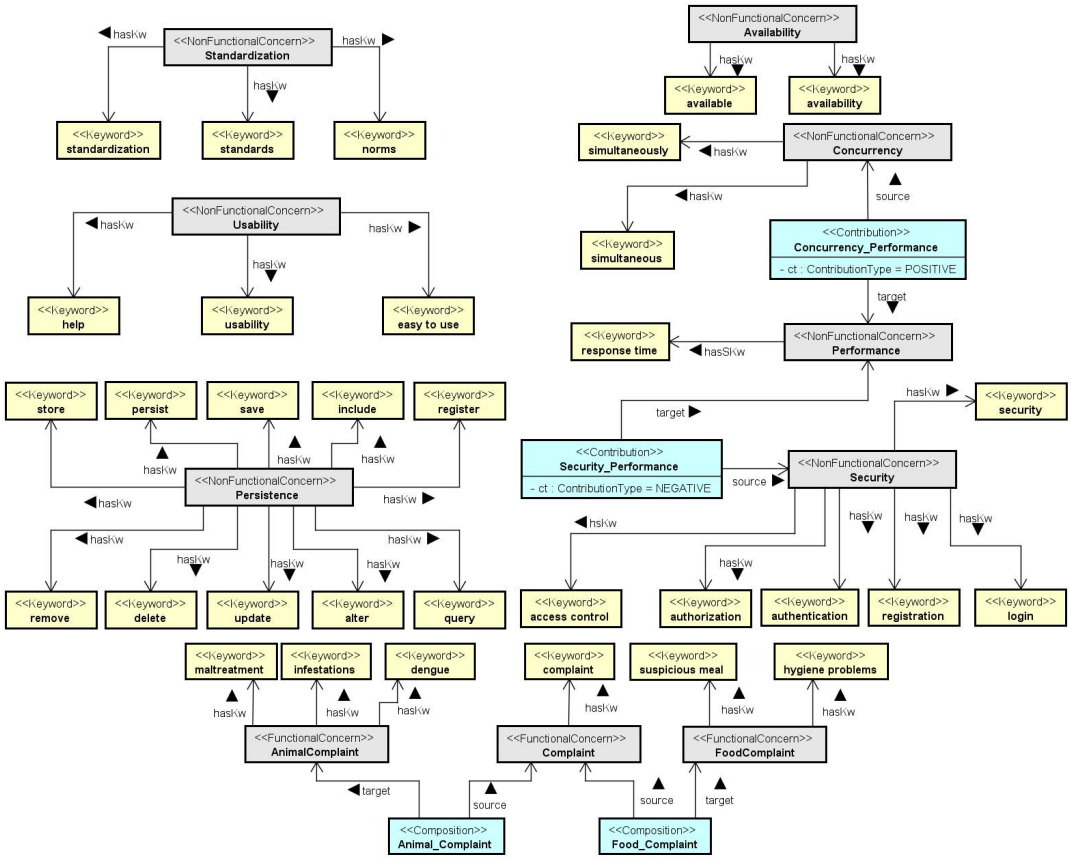


Figure 2. Part of a catalog generated from historical data.

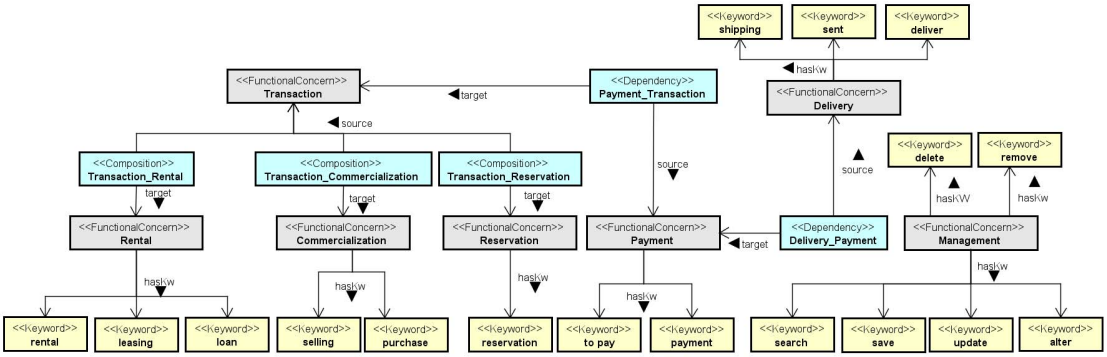
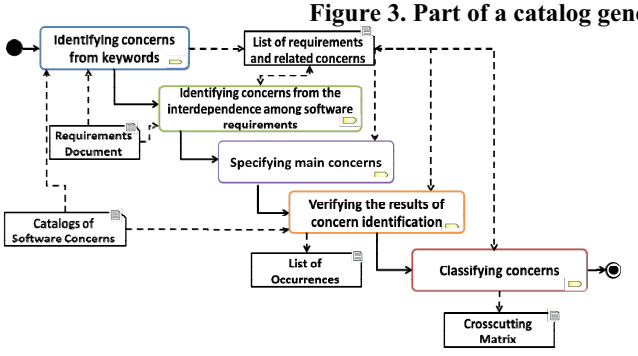


Figure 3. Part of a catalog generated from a pattern language.



Legend: Activity; Artifact.

Figure 4. Overview of the "Performing Concern Identification and Classification" phase

As may be seen in Figure 4, this activity takes the catalog of software concerns and the requirements document as inputs and generates a list of requirements and related concerns as an output, *i.e.*, a list in which, for each requirement there is a set of concerns identified for it.

By taking the requirements of Table 1 and the catalog of Figure 2 as inputs, after executing this activity, it is generated the list of requirements and related concerns presented in Table 2.

Table 2. List of requirements and related concerns.

Requirement FR-01	**Concerns**
It allows the state of a **complaint** to be **updated**. The **complaint** must be **registered** and have the OPENED state.	Persistence
	Complaint
Requirement NFR-02	**Concerns**
The system should have an **easy to use** GUI, as any person who has access to the internet should be able to use the system. The system should have an on-line **HELP** to be consulted by any person that uses it.	Usability
Requirement NFR-03	**Concerns**
The **response time** must not exceed 5 seconds.	Performance
Legend: the keywords used in the concern identification were underlined.	

2) Identifying concerns from the interdependence among software requirements

In this activity, the software engineer has the responsibility of identifying other software concerns, which could not be identified only using keywords. To do this, the dependency relationships among software requirements and the list of requirements and related concerns are used. As results, the list of requirements and related concerns may be updated, including new concerns, if necessary.

To exemplify a situation for which this activity is relevant, consider the requirements presented in Table 1. It may be noticed that the requirement "FR-01" depends on the requirement "NFR-01", which was written aiming to specify the performance behavior of the software. This dependency exists because in the description of the requirement "NFR-01" is clear that the performance attribute must be applied to other functions of the software. Once the requirement "FR-01" depends on the requirement "NFR-01", related to "Performance" concern, then we may assume that "FR-01" is related to this concern too.

After executing this activity, the list of requirements and related concerns is updated, as can be seen in Table 3. The requirement "FR-01" now is related to "Performance" and "Usability" concerns. The reasons for the inclusion of "Usability" are similar to those presented for "Performance" concern. The "Main Concern" column will be explained later in this paper.

Table 3. List of requirements and related concerns updated.

Requirement FR-01	**Concerns**	**Main Concern**
It allows the state of a **complaint** to be **updated**. The **complaint** must be **registered** and have the OPENED state.	Persistence	
	Complaint	**X**
	Usability	
	Performance	
Requirement NFR-02	**Concerns**	**Main Concern**
The system should have an **easy to use** GUI, as any person who has access to the internet should be able to use the system. The system should have an on-line **HELP** to be consulted by any person that uses it.	Usability	X
Requirement NFR-03	**Concerns**	**Main Concern**
The **response time** must not exceed 5 seconds.	Performance	X

3) Specifying main concerns

In this activity, the software engineer must inform what is the main concern of each software requirement. This concern represents the main purpose for which the requirement was written. The result of this activity is the updating of the list of requirements and related concerns; the specification of main concerns is important for the concern classification, as will be presented in the next sections.

In the example of Table 3, the requirements "NFR-01" and "NFR-02" are related to only one concern, which is their main concern. The requirement "FR-01", in turn, is related to four distinct software concerns: "Persistence", "Complaint", "Performance" and "Usability". By considering the description of this requirement, it is possible to notice that it was written in order to specify the feature related to complaint updates. Hence, "Complaint" must be considered the main concern of this requirement.

If there is a requirement for which is difficult to decide which is its main concern, the software engineer may consider rewriting this requirement. It is also important to state that "a requirement with only one concern" does not mean that this concern is the main concern of the requirement. This is possible because the identified concern may be a false positive. Hence, it is important that the software engineer check the requirements with only one concern as well.

4) Verifying the results of concern identification

In this activity, the software engineer has the responsibility of verifying the list of requirements and related concerns, aiming to find potential problems with the concern identification process. This must be done before performing the next activities of the approach.

This activity takes the list of requirements and related concerns and the catalog of software concerns as inputs and may generate a list of occurrences regarding to concern identification process. To produce this list, the software engineer must apply a set of four heuristics, as presented in Table 4. This table presents the description of each heuristic, as well as the reason for the existence of it.

When a heuristic is not satisfied, an occurrence is generated and then it must be analyzed by the software engineer. For instance, one of the proposed heuristics state that each software requirement must be related to its main concern. If a particular requirement "r" is not addressed by any software concern, an occurrence will be generated for this requirement. It is important to notice that not all occurrences represent an error. Hence, the software engineer must check the need to resolve or not each generated occurrence.

Table 4. Heuristics for the verification of the concern identification process.

Heuristic #1
Description: each software requirement is related to its main concern.
Justification: each software requirement must be related to a main concern, because each requirement is written with one purpose.
Heuristic #2
Description: if there is a "positive contribution" relationship "rel" that binds the concerns "A" (source) and "B" (target), and "B" was found in the software requirements, then "A" or any of its sub-concerns was identified too.
Justification: the fact that "A" contributes positively to "B" provides evidences that if "B" was identified, "A" (or any of its sub-concerns) should also be. However, this is not an error. More than one concern can contribute positively to "B" and the software engineer could choose just one option. For example, "Performance" and "Standardization" contribute positively to "Usability", but only one of them may be addressed in the software. However, it is necessary to generate a warning occurrence, since it may indicate concerns that the software engineer had not previously considered.
Heuristic #3
Description: if there is a "dependency" relationship "rel" that binds the concerns "A" (source) and "B" (target), and "A" was found in the software requirements, then "B" or any of its sub-concerns was identified too.
Justification: the fact that "A" depends on "B" means that for that "A" exists, "B" (or any of its sub-concerns) must exist too. For example, the catalog of Figure 3 presents a dependency relationship between "Payment" and "Transaction". Then, for that "Payment" exists, "Transaction" must exist too.
Heuristic #4
Description: if a non-functional concern "A" was found in the software requirements, then "A" (or any of its sub-concerns) is related to one or more functional requirements.
Justification: it is well known in the scientific community that non-functional concerns commonly presents a crosscutting behavior, such as "Logging", "Persistence", "Distribution", "Security", among others (Sampaio *et al*. [28]). Thus, at the end of the concern identification process, if there are non-functional concerns identified in the software that do not affect any functional requirements, the crosscutting behavior of this concern is being omitted. This is not an error occurrence, but is a warning that needs to be checked by the software engineer.

ObasCId approach also provides, for each heuristic, a set of suggestions for solving the occurrence generated by this heuristic. The goal is to help the *ObasCId* users, especially the non-experts, to take more appropriate decisions on how to deal with these occurrences. Due to space limitation, only the suggestions for the heuristic #3 are presented below. The suggestions of remain heuristics must be found in [25]:

- Check the spelling of the keywords related to "B" concern (and its sub-concerns), as well as those related to the software requirements;
- Check the possibility of adding new keywords to the "B" concern (or its sub-concerns); or

- Check the possibility of rewriting the description of some software requirements.

By performing this activity on the list of requirements and related concerns presented in Table 3, taking as input the catalog of Figure 2, it will be generated an occurrence derived of heuristic #2, because the "Concurrency" concern contributes positively to "Performance" (according to the catalog of software concerns), but "Concurrency" was not identified in the software requirements. In this case, we considered that this is not a problem and we ignored this occurrence.

If needed, the software engineer may back to the initial phases of the approach, such as "Preparing the Catalog of Software Concerns" or "Preparing the Requirements Document", aiming to solve the occurrences produced by this activity.

5) Classifying concerns

This activity uses the list of requirements and related concerns to build a *crosscutting matrix* that represents the crosscutting relationships among different software concerns. Crosscutting matrix is a "Main Concern *vs*. CrossCuttting Concern" matrix; when a cell "[C1, C2]" is highlighted, this indicates that "C2" cut-across "C1".

In this activity, we can assume that each software requirement has a main concern "MC" and a set of zero or more related concerns {"C_1", "C_2", … "C_n"). In the *ObasCId* approach, we consider that all concerns "C_1", "C2", … "C_n" cut-across the main concern, "MC". Hence, all cells "[MC, C_1]", "[MC, C_2]", … "[MC, C_n]", must be highlighted. If a requirement is related only to its main concern, none cell of the row "MC" will be highlighted.

From the list of requirements and related concerns of Table 3, it is possible to generate the crosscutting matrix presented in Table 5.

Table 5. Example of a crosscutting matrix.

↓ Main Concerns/CCC →	1: Persist.	2: Compl.	3: Usab.	4: Perf.
1: Persistence				
2: Complaint	X		X	X
3: Usability				
4: Performance				

It is possible to notice that the "FR-01" requirement, whose main concern is "Complaint", is related to other concerns, such as "Persistence", "Usability" and "Performance". Hence, the cells "[Complaint, Persistence]", "[Complaint, Usability]" and "[Complaint, Performance]" were marked with a "X".

By keeping the focus on the columns of a crosscutting matrix, the software engineer will have an overview on which concerns cut-across the behavior of other concerns. The more a concern "A" affects other software concerns, the higher is the likely of "A" be a crosscutting concern. To know which requirements are affected by a specific concern, the list of requirements and related concerns may be used.

In an ideal scenario, each concern "A" should only affect requirements for which it is its main concern. In other words, the column related to "A" concern should contain only empty cells. Hence, in the *ObasCId* approach, all columns with at least a "X" value refer to crosscutting concerns candidates. In the case of Table 5, all concerns, except the "Complaint" (column 2), are considered crosscutting concerns candidates.

The crosscutting matrix proposed in this paper is similar to that presented in the approach proposed by Rashid *et al*. [27]. However, the matrix proposed by Rashid *et al*. is a "Non-functional Concerns *vs*. Viewpoints" matrix. Hence, only the influence of non-functional concerns over functional concerns (called viewpoints in the authors' proposal) may be studied. The advantage of the crosscutting matrix proposed in this paper is that the crosscutting behavior existing among functional concerns on other software concerns can also be analyzed. This is important, because it is well-known that functional concerns also can cut-across other software concerns [19].

Based on the results of the concern identification and classification process, the software engineer may back to the initial phases of the approach, aiming to include/remove/update elements of the catalog or requirements document that will improve the quality of these results.

V. Experimental Study

For the assessment of the *ObasCId* approach, the following GQM-based goal [32] was proposed: *to analyze* the usage of the *ObasCId* approach*, in order* to evaluate*, with respect to* its effectiveness (recall and precision) and efficiency (time of execution)*, from the point of view of* software engineers*, in the context of* a group of undergraduate and graduate in Computer Science.

Aiming to achieve this goal, a group of participants was asked to identify and classify the concerns of two software using as support the *ObasCId* and *Theme/Doc* approaches [3][11].

A. Theme/Doc Overview

The *Theme/Doc* [3][11] approach is based on three main activities: "Identifying key-actions", "Building an action-view" and "Classifying actions as base or crosscutting ones". Identifying concerns with *Theme/Doc* requires that the software engineer provides: (i) a list of key-actions, *i.e*., verbs identified from the software requirements ("Identifying key-actions" activity); and (ii) a set of software requirements. Based on these inputs, the software engineer performs an analysis of the requirements document and generates an action-view artifact ("Building an action-view" activity). An action-view represents the relationships among requirements and key-actions.

The classification of these actions as base or crosscutting ones may be performed by mean of the "Classifying actions as base or crosscutting ones" activity, that requires as inputs the action-view and the set of software requirements. The software engineer initially must examine the requirements that refer to more than one key-action and determine what is the primary action (more important action) of these requirements. Once defined the primary action of a requirement, we say that all other actions of it are affected by the behavior of the primary action. The idea is to separate and isolate actions and requirements into two groups: (i) the "base" group, that is self-contained, *i.e*., the requirements of this group do not refer to actions of the other group; and (ii) the "crosscutting" group, whose requirements can refer to actions of the base group.

The primary actions and the process of action classification are similar to the concepts of main concern and concern classification activity in *ObasCId* approch. However, *ObasCId* takes into consideration the relationships

between requirements and concerns to improve the effectiveness the concern identification and classification process. Furthermore, *ObasCId* provides resources to represent and reuse the knowledge about concern domain in other projects, such as, concern catalogs, heuristics, among others.

Theme/Doc was chosen to be compared to *ObasCId* because: (i) it is based on the usage of keywords; (ii) unlike other approaches [8], *Theme/Doc* does not depend on computational tools for its execution; (iii) it is simple and easy to use; (iv) the authors of this paper had some previous experience on the usage of *Theme/Doc*; and (v) it is a robust approach that has been evaluated in recent experimental studies [15].

B. Planning

The planning of this experimental study was defined according to the Wohlin's proposal [32] and involves the following steps: (i) context selection; (ii) hypotheses formulation; (iii) variables selection; (iv) participants selection; and (v) design and execution of the experimental study.

a) Context selection. This experimental study was conducted with fourteen undergraduate and graduate students in Computer Science from two Federal Universities from Brazil. The requirements documents of *Health Watcher* [13] and of an information system for DVD rental (*LocaDVD*) [30] were used in this study. As already stated in this paper, *Health Watcher* is a well-known application in the AORE area and was chosen because it has a suitable requirements document for concern identification and classification. *LocaDVD*, in turn, was chosen because it is a business resource management application, suitable for be used with catalogs for software concerns created from the pattern language proposed by Braga *et al*. [5], such as the catalog of Figure 3.

b) Hypotheses formulation. An important part of the hypotheses formulation step is the specification of the metrics that will be used in the experimental study. Based on these metrics, the researcher may establish hypotheses and draw conclusions from the results of the experiment. In this work, three metrics were used: (i) **Recall (Re) -** the proportion of the amount of correctly identified and classified concerns on the amount of existing concerns; (ii) **Precision (Pr) -** the proportion of the amount of correctly identified and classified concerns on the amount of identified concern; and (iii) **Execution Time (T) -** time (in minutes) spent for performing the activities proposed in the experimental study.

Based on these metrics, six hypotheses were developed, two related to recall, two for the precision and two for the execution time (Table 6).

Table 6. Hypotheses of the experimental study.

Hypotheses for Recall	
H_{0Re}	There is no difference of using *ObasCId* or *Theme/Doc*, regarding to the recall, that is, H_{0Re}: $Re_{ObasCId} = Re_{Theme/Doc}$
H_{1Re}	There is difference of using *ObasCId* or *Theme/Doc*, regarding to the recall, that is, H_{1Re}: $Re_{ObasCId} \neq Re_{Theme/Doc}$
Hypotheses for Precision	
H_{0Pr}	There is no difference of using *ObasCId* or *Theme/Doc*, regarding to the precision, that is, H_{0Pr}: $Pr_{ObasCId} = Pr_{Theme/Doc}$
H_{1Pr}	There is difference of using *ObasCId* or *Theme/Doc*, regarding to the precision, that is, H_{1Pr}: $Pr_{ObasCId} \neq Pr_{Theme/Doc}$
Hypotheses for Execution Time	
H_{0T}	There is no difference of using *ObasCId* or *Theme/Doc*, regarding to the execution time, that is, H_{0T}: $T_{ObasCId} = T_{Theme/Doc}$
H_{1T}	There is difference of using *ObasCId* or *Theme/Doc*, regarding to the execution time, that is, H_{1T}: $T_{ObasCId} \neq T_{Theme/Doc}$

c) Variables and participants selection. Independent variables are those manipulated and controlled during the experimental study. In this study, the only independent variable is the approach for concern identification and classification (*ObasCId* and *Theme/Doc*). The dependent variables are those under evaluation and whose variations must be observed. In this experiment, the recall, precision and execution time metrics are dependent variables. The participants of this study were selected through a non-probability for convenience sampling.

d) Design and execution of the experimental study. The distribution of the participants was performed aiming to form two homogeneous groups, regarding to the participants' expertise. Each group had seven participants and their expertise was verified by the application of a profile characterization questionnaire. This questionnaire took into account the knowledge of the participants about the AORE area and the approaches used in the experiment.

The experimental study was planned in phases to minimize the effect of participants' knowledge of the dependent variables. Before starting the experiment, a training was conducted, in order to homogenize the knowledge of participants on AORE and on *Theme/Doc* and *ObasCId* approaches. During the training, it was not informed to the participants what approach was developed by the authors of this paper.

The execution of the experimental study occurred in two phases. In the *first phase*, participants should identify the non-functional concerns present in the requirements document of the *Health Watcher* and classify them as crosscutting or non-crosscutting. To do this, the Group 1 used the *Theme/Doc* approach and Group 2, the *ObasCId*. In the second phase, participants should identify the functional and non-functional concerns of the *LocaDVD* and also classify them as crosscutting or non-crosscutting. To do this, the Group 1 used the *ObasCId* approach and Group 2, *Theme/Doc*. The participants had to perform all activities proposed by *Theme/Doc*. In the case of *ObasCId*, the participants had to perform the activities presented in Figure 4, *i.e.*, only the activities of the "Performing Concern Identification and Classification" phase.

The part of the *Health Watcher* requirements document analyzed by the participants had six types of non-functional crosscutting concerns: "Security", "Concurrency", "Usability", "Performance", "Availability" and "Persistence". Functional concerns were not considered, because it was not found a source that could be used to generate a catalog of functional concerns regarding to the health complaint domain. For the *LocaDVD* software, the requirements document had four functional concerns ("Payment", "Transaction", "Resource" and "Destination") and two non-functional concerns ("Logging" and "Persistence"); three of these six concerns were crosscutting ones ("Logging", "Persistence" and "Transaction"). The size of requirements documents of both software was similar. To calculate the values of the recall and precision metrics, it was considered the amount of concern correctly identified and classified by each participant, individually.

In the first phase of the experiment, the participants of the *Group 2* also received a catalog of non-functional concerns, created by the authors of this paper from the NFR catalogs proposed by Boehm and In [4], Chung and Leite [10] and Cysneiro [12]. In the second phase of the experiment, along with the requirements document of *LocaDVD*, the participants of *Group 1* received the catalog

of non-functional concerns used in the first phase of the experiment and a catalog of functional concerns generated from the patterns of the Business Resource Management language [5].

C. Results and Discussion

Table 7 presents the results obtained by both groups of participants, regarding to the *Health Watcher* software (first phase). Taking into account the values for recall, the participants who used the *ObasCId* approach had, on average, more promising results than those who used the *Theme/Doc*. It is also possible to notice that there is no relevant difference between the two approaches, regarding to the precision. Table 7 still presents that the execution time provided by *ObasCId* (46 min) was higher than that one provided by *Theme/Doc* approach (41 min). This is due to the participants who used *ObasCId* had other artifacts to be analyzed, *i.e*., the catalogs of software concerns, as well as some new activities to perform. However, we noted that the difference between the two values (5 minutes) is not significant. Although the participants who used the *ObasCId* approach had to perform additional tasks, the usage of the catalogs and the proposed process may have led the participants to perform the concern identification and classification activities in a more focused way. This may have minimized the impact on the execution time provided by *ObasCId* approach.

Table 7. Experimental results - first phase.

***Theme/Doc* (Group 1)**				***ObasCId* (Group 2)**			
Partic.	**Recall (Re)**	**Precision (Pr)**	**Time (min)**	**Partic.**	**Recall (Re)**	**Precision (Pr)**	**Time (min)**
P1	42,85	75,00	43	P8	71,42	71,00	62
P2	42,85	100,00	48	P9	85,71	100,00	39
P3	42,85	100,00	49	P10	85,71	100,00	54
P4	28,57	66,00	48	P11	71,42	100,00	37
P5	57,14	80,00	36	P12	57,14	75,00	43
P6	42,85	100,00	31	P13	71,42	80,00	42
P7	28,57	100,00	34	P14	71,42	100,00	42
Avg.	**40,81**	**88,71**	**41,28**	**Avg.**	**73,46**	**89,42**	**45,57**

To improve the discussion about the recall values, Table 8 presents: (i) the list of concerns of *Health Watcher* - first column; (ii) the concerns identified by each participant who used the *Theme/Doc* approach - from second to eighth columns; (iii) the percentage of participants who identified each concern - ninth column; and (iv) the same information previously described to the *ObasCId* approach – from tenth to the eighteenth column.

Table 8. Health Watcher concern identification.

#	Participants *Theme/Doc*							%	Participants *ObasCId*							%
	1	**2**	**3**	**4**	**5**	**6**	**7**		**8**	**9**	**10**	**11**	**12**	**13**	**14**	
1	X	X						28	X	X	X	X	X			57
2	X	X	X	X	X	X	X	100	X	X	X	X	X	X	X	100
3		X						14		X	X	X		X	X	71
4			X	X	X	X		57	X	X	X	X		X	X	85
5					X	X	X	43	X	X	X			X	X	71
6	X		X		X			43	X	X	X	X	X			71
Average								**47,5**	**Average**							**75,8**
Legend: (1) Persistence; (2) Security; (3) Concurrency; (4) Usability; (5) Performance; (6) Availability.																

Based on this table, it is possible to notice that only one of the participants who used the *Theme/Doc* approach was able to identify the "Concurrency" concern; "Concurrency" was an implicit concern of the *Health Watcher* requirements document, *i.e*., a concern not explicitly mentioned by means of keywords. Regarding to the participants who used *ObasCId* approach, just two participants did not identify this concern. We believe this happens due to the usage of dependency relationships among software concerns during the concern identification and classification activities, as proposed by *ObasCId* approach. For all concerns, the percentage of participants who identified them is always higher for *ObasCId* approach than for the *Theme/Doc*. Consequently, on average, the percentage of participants who identified any concern using *ObasCId* approach (75,8%) is higher than that one who used *Theme/Doc* (47,5%).

The same type of information presented for the *Health Watcher* are also presented for *LocaDVD* (second phase), as can be seen in Table 9.

Table 9. Experimental results - second phase.

***ObasCId* (Group 1)**				***Theme/Doc* (Group 2)**			
Partic.	**Recall (Re)**	**Precision (Pr)**	**Time (min)**	**Partic.**	**Recall (Re)**	**Precision (Pr)**	**Time (min)**
P1	83,00	83,00	32	P8	33,00	66,00	18
P2	83,00	71,00	22	P9	66,00	80,00	29
P3	100,00	75,00	18	P10	66,00	80,00	15
P4	66,00	100,00	42	P11	33,00	100,00	32
P5	66,00	80,00	37	P12	71,00	71,00	13
P6	100,00	86,00	22	P13	50,00	60,00	18
P7	83,00	71,00	25	P14	50,00	75,00	21
Avg.	**83,00**	**80,85**	**28,28**	**Avg.**	**52,71**	**76,00**	**20,85**

Some important facts about the results of Table 9 are: (i) Sampaio *et al.* [28] stated the precision of AORE approaches is satisfactory, but the recall not. This situation was observed in the case of *Theme/Doc* approach, but not for the *ObasCId* approach. The recall provided by *ObasCId* is quite similar to the precision. This may be due to the support provided by *ObasCId* approach for the software engineers to perform the concern identification and classification;

(ii) the execution time provided by both approaches reduced when it is compared to the execution time needed to identify and classify the concerns of the *Health Watcher* software; however, the difference between the *ObasCId* and *Theme/Doc* approaches continues, *i.e.*, less time was needed for the execution of *Theme/Doc* approach. The reduction may be explained by the features of the software used. Although both software contain a similar number of concerns and requirements, the domain of the *LocaDVD* software is more common than the domain of *Health Watcher*. This could have facilitated the process of reading and understanding the requirements document of *LocaDVD*; and

(iii) the recall provided by *ObasCId* approach is still higher than the recall provided by *Theme/Doc*, even using different software and participants; the precision provided by *ObasCId* approach remains higher than that provided by the *Theme/Doc*; however, the difference was not significant.

D. Hypothesis tests

To verify the hypotheses defined in Table 8, the *t-test* was applied [18]. Regarding to the *Health Watcher* software, comparing the average values for recall provided by the approaches *Theme/Doc* (average = 40,81) and *ObasCId* (average = 73,46), the H_{0Re} null hypothesis may be rejected with significance level of 99,9% (*p-value* = 0,0004). This situation also happens for the *LocaDVD* software, regarding to the recall. Regarding to the average time spent by the participants to perform the activities of the *Theme/Doc* and *ObasCId*, it was not possible to obtain statistical evidences, with significance level equal or higher than 95%, to state that these values are different. For both software, we obtained the same situation for precision values.

E. Threats to validity

The main threats to validity of this study are: **1) Conclusion validity**. This type of threat refers to issues that affect the ability to draw correct conclusions about the experimental results. An example of this type of threat is the

choice of the statistical methods for data analysis. In this study, the *t-test* was used, which requires normally distributed data. To verify if the data is normally distributed, we have applied a test known as *Shapiro-Wilk* test [18] and the values for recall, precision and time metrics were considered normalized with a significance level of 99,9%.

2) Internal validity. It refers to issues that may affect the ability to ensure that the results were, in fact, obtained from the treatments (*i.e.* the AORE approaches: *ObasCId* and *Theme/Doc*) and not by coincidence. A threat of this type can be related to the strategy used to select and group the participants of the experimental study. To mitigate this threat, we did not demonstrate expectations for any approach during the training phase. In addition, the participants were grouped according to their levels of experience.

3) External validity. This type of threat refers to issues that affect the ability to generalize the results of an experiment to a wider context. In this case, the relevant factors that could have influenced the results of this study are: (i) the size of the applications used in the study; (ii) the quality of the resources (software concerns catalogs and the requirements documents) presented to the participants; (iii) the amount of participants of the study; and (iv) the usage of undergraduate and graduate students in Computer Science. In order to mitigate these potential threats, we intend to replicate this experiment with other groups of participants and different applications.

VI. FINAL REMARKS

This paper presented an ontology for concerns (*O4C*) and ontologically-based AORE approach (*ObasCId*). The main innovation of *ObasCId* approach are the catalogs built from *O4C* concepts and the set of activities to support the software engineers during the concern identification and classification. An experimental study performed on *ObasCId* showed evidences that the usage of this approach may improve the values for recall, without negatively impact on the execution time and precision.

The main limitation of this approach is that the quality of the results provided by it is influenced by: (i) the existence of good catalogs of software concerns; (ii) the expertise of the software engineers in performing some activities of *ObasCId* approach, such as "Specifying main concerns"; and (iii) the lack of computational tools for supporting the execution of *ObasCId* in medium and high scale software.

Aiming to mitigate these limitations, as future work proposals, we intend to: (i) register other types of concerns as catalogs of the *O4C* ontology; (ii) create a computational tool for concern identification and classification, based on the *ObasCId* approach; and (iii) propose heuristics that aid software engineers to find the main concern of a requirement. Furthermore, we intent to introduce *ObasCId* in a real professional environment, aiming to figure out how easy/hard is to understand and effectively use the approach in a software organization.

REFERENCES

[1] Agostinho, S. *et al*. Metadata-driven approach for aspect-oriented requirements analysis. 10th International Conference on Enterprise Information Systems. Barcelona, Spain, 2008.

[2] Alencar, F. *et al*. Towards modular i* models, ACM Symposium on Applied Computing, 2010.

[3] Baniassad, E.; Clarke, S. Theme: An approach for aspect-oriented analysis and design. 26th ICSE. Washington, 2004.

[4] Boehm, B; In, H. Identifying Quality-Requirement Conflicts. IEEE Software v. 13(2), 1996.

[5] Braga, R. T. V.; Germano, Fernão Stella Rodrigues ; Masiero, Paulo Cesar . A Pattern Language for Business Resource Management. 6th PLoP, 1999.

[6] Brito, I.; Moreira A. Towards a Composition Process for Aspect-Oriented Requirements. EA Work. at AOSD. Boston, USA, 2003.

[7] Chernak, Y. Requirements Composition Table Explained. In: 20th RE Conference. Chicago, Illinois, USA, 2012.

[8] Chitchyan, R.; Sampaio, A.; Rashid, A.; Rayson, P. A tool suite for aspect-oriented requirements engineering. Int. Work. on Early-Aspects at ICSE. New York, USA, 2006.

[9] Chitchyan, R.*et al*. A. Report synthesizing state-of-the-art in aspect-oriented requirements engineering, architectures and design. Technical Report. Lancaster University: Lancaster, 259 p., 2005.

[10] Chung, L.; Leite, J. S. P. Non-Functional Requirements in Software Engineering: Springer, 441 p., 2000.

[11] Clarke, S.; Baniassad, E. Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley, 2005.

[12] Cysneiro, L. M. Catalogues on Non-Functional Requirements. Available at: http://www.math.yorku.ca/~cysneiro/nfrs/nfrs.htm. Last access: July, 2016.

[13] Health Watcher. Available at: http://www.cin.ufpe.br/ ~scbs/testbed/requirements /aore/. Last access: July, 2016.

[14] Falbo, R. A. SABiO: Systematic Approach for Building Ontologies. 2011. Available at: http://www.inf.ufes.br/~falbo/files/SABiO.pdf. Last access: July, 2016.

[15] Guizzardi, G. Ontological Foundations for Structural Conceptual Models. PhD. thesis. Univeristy of Twente, 2005.

[16] Herrera, J. *et al*. Revealing Crosscutting Concerns in Textual Requirements Documents: An Exploratory Study with Industry Systems. 26th SBES. Natal, RN, 2012.

[17] López, C.; Cysneiro, L. M.; Astudillo, H. NFR Ontology: Sharing and Reusing NFR and Design Rationale Knowledge. Int. Workshop on Managing Requirements Knowledge. USA, 2008.

[18] Montgomery, D. C. Design and Analysis of Experiments, 5ª ed., Wiley, 2000.

[19] Moreira, A.; Rashid, A.; Araújo, J. Multi-Dimensional Separation of Concerns in Requirements Engineering. 13th RE. Paris, 2005.

[20] Mussbacher, G.; Amyot, D.; Araújo, J.; Moreira, A. Requirements modeling with the aspect-oriented user requirements notation (AoURN): A case study. Transactions on aspect-oriented software development. Springer-Verlag, Berlin, Heidelberg, p. 23-68, 2010.

[21] Parreira Júnior, P. A.; Penteado, R. A. D. Crosscutting Concerns Identification Supported by Ontologies: A Preliminary Study. LBIP. Springer International Publishing, 2015.

[22] Parreira Júnior, P. A.; Penteado, R. A. D. OnTheme/Doc: An Ontology-Based Approach for Crosscutting Concern Identification from Software Requirements. XVII ICEIS. Barcelona, Espanha, 2015.

[23] Parreira Júnior, P. A.; Penteado, R. A. D. An Overview on Aspect-Oriented Requirements Engineering Area. LBIP. 16ed.: Springer International Publishing, 2015, v. 227, p. 244-264.

[24] Parreira Júnior, P. A.; Penteado, R. A. D. Aspect-Oriented Requirements Engineering: A Systematic Mapping. XVI ICEIS. Lisboa, Portugal, 2014.

[25] Parreira Júnior, P. A. *ObasCId*: uma Abordagem Ontologicamente Fundamentada para EROA. Doctorade Thesis. 2015. UFSCar/São Carlos/SP. 197p (*in portuguese*).

[26] Parreira Júnior, P. A.; Penteado, R. A. D. Domain Ontologies in the Context of Requirements Engineering: A Systematic Mapping. XII ACS/IEEE. Marrakech, Morocco, 2015.

[27] Rashid, A.; Moreira, A.; Araújo, J. Modularisation and composition of aspectual requirements. 2nd AOSD. New York, USA, 2003.

[28] Sampaio, A.; Greenwood P.; Garcia, A. F.; Rashid, A. A Comparative Study of Aspect-Oriented Requirements Engineering Approaches. I Int. Symp. on Empirical SE and Measurement. Madrid, Spain, 2007.

[29] Soeiro E.; Brito, I. S; Moreira, A. An XML-Based Language for Specification and Composition of Aspectual Concerns. 8th Int. Conf. on Enterprise Information Systems. Paphos, Cyprus, 2006.

[30] Viana, M. C. Building the Graphical User Interface Layer and a Wizard for the GRENJ Framework. Master dissertation, UFSCar, São Carlos, 2009 (*in portuguese*).

[31] Whittle J.; Araújo, J. Scenario Modeling with Aspects. In: IEEE Software, v. 151(4), p. 157-172, 2004.

[32] Wohlin, C.; Runeson, P.; Höst, M.; Regnell, B.; Wesslén, A. Experiment. in SE. Springer-Verlag Berlin Heidelberg. 236p. 2012.

[33] Zheng, X.; Liu, X.; Liu, S. Use case and non-functional scenario template-based approach to identify aspects. 2nd Int. Conf. on Computer Eng. and Applications. Bali Island, Indonesia, 2010.