Parreira Júnior and Penteado *Journal of the Brazilian Computer Society* (2018) 24:3
DOI 10.1186/s13173-017-0067-6

Journal of the
Brazilian Computer Society

RESEARCH Open Access

ObasCId(-Tool): an ontologically based approach for concern identification and classification and its computational support

Paulo Afonso Parreira Júnior[1*] and Rosângela Aparecida Dellosso Penteado[2]

Abstract

The aspect-oriented requirements engineering (AORE) area intends to provide more appropriated 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 appropriated 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 concern domain, called O4C (Ontology for Concerns); (ii) an ontologically based approach for AORE, called ObasCId, 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; and (iii) a computational support, called ObasCId-Tool, that automates some activities of the ObasCId. Two quasi-experimental studies were performed on ObasCId and ObasCId-Tool, and their results indicated that these technologies 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

Introduction

In the context of requirements engineering (RE), a concern can be understood as a set of software requirements related to the same purpose [1]. The two main categories of software concerns are *functional* and *non-functional concerns*. The first one regards to concerns that are related to the functional features of the software, such as "Payment," "Order Management," "Reservation," among others. The last one corresponds to concerns related to the non-functional features of the software, such as "Security," "Persistence," "Performance," "Logging," among others. 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 a more appropriated way [2]. However, there are some 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 (functional or non-functional) whose requirements are spread over requirements of other software concerns [3]. For instance, a security concern may contain requirements related to the encryption and/or authorization mechanisms. These requirements, in turn, may affect (cut across) some requirements related to "Orders Management" concern, for instance.

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 engineers on the effects caused by the inclusion, removal, or modification of a requirement over the other ones [1]. The aspect-oriented requirements engineering (AORE) area deals with software concerns during the early phases of software development [2, 4], in order to identify, classify, modularize, and compose these concerns in a more appropriated way.

* Correspondence: pauloa.junior@dcc.ufla.br
[1]Department of Computer Science, Federal University of Lavras, Lavras, MG, Brazil
Full list of author information is available at the end of the article

Some experimental studies performed on the main AORE approaches [5–8] have pointed out the concern identification and classification as bottleneck activities. One of the possible causes of this is the *lack of understanding about the software concern 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?" and "how does a concern affect other software concerns," among others. Generally, the knowledge about software concern domain is spread in different studies, sometimes in a divergent way, which may preclude the understanding of researchers and practitioners. Another possible cause is the *lack of appropriated resources (guidelines, processes, catalogs, computational tools, among others) to support software engineers during the concern identification and classification* [9, 10]: several AORE approaches rely only either on the software engineers' expertise or on the usage of keywords for the correct identification and classification of software concerns, which may decrease the effectiveness of these approaches. In addition, most of the computational supports for AORE either do not address the concern identification and classification activities, or are not available in a public way, or have obsolete documentation, which may indicate that they were discontinued. The "Related works" section of this paper presents more details about these causes, taking the related works into account.

In this context, this work aims to improve the effectiveness of the concern identification and classification activities by dealing with the previous mentioned causes. To do this, we propose (i) a reference ontology for the software concern domain, called O4C (Ontology for Concerns), that aims to make clear and precise the description of the concepts of this domain; (ii) an ontologically based AORE approach, called ObasCId (Ontologically based Concern Identification and Classification), that provides more appropriated resources (catalogs, heuristics, and processes) for supporting software engineers during the concern identification and classification; and (iii) a computational support, called ObasCId-Tool, that automates several of the activities and artifacts proposed in ObasCId. The assessment performed on the ObasCId and ObasCId-Tool provided results that lead us to believe that the usage of these resources may improve the recall of the concern identification and classification, without negative impacts on the precision and the execution time of these activities. It is important to state that this paper is an extension of previous studies [7, 11] and it improves the content presented in them as follows: (i) a better description of the concepts and relationships of the O4C ontology is presented and (ii) a computational support for concern identification and classification is presented and its assessment by means of a quasi-experimental study is described.

This paper is organized as follows: the "Related works" section presents a discussion about the related works; in the "Ontology for Concerns (O4C)," "ObasCId approach," and "ObasCId-Tool" sections, the O4C ontology, the ObasCId approach and its computational support, and ObasCId-Tool are presented, respectively. The description of the quasi-experimental studies performed on the ObasCId and ObasCId-Tool are in the "Quasi-experimental study I" and "Quasi-experimental study II" sections. Finally, the "Final remarks" section highlights the final remarks of this paper, including the limitations of the presented work and the proposals for future works.

Related works

Several AORE approaches have been proposed in the last years, especially, for concern identification and classification [9, 10]. Many of these approaches ([3, 4, 12–18] suggest the usage of catalogs of non-functional requirements (NFR catalogs), such as those proposed by Boehm and In [19], Chung and Leite [20], and Cysneiro [21], 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, since these catalogs are not prepared for the software concern 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. [17], functional requirements also may cut across other software requirements; hence, it is important to take them into account during the concern identification and classification activities. 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 appropriated way.

In other approaches [2, 22, 23], no resources 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 requirement 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. [8],

"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 concern domain is spread in different studies, sometimes in a divergent way. Most of the existing AORE approaches represent 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 describes 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. [17] 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 [12, 16, 18].

In regard to computational supports, in a previous work, we have noticed that there is a lack of tools that support the concern identification and classification activities [24]. This observation makes sense, since most of AORE approaches do not provide resources for aiding software engineers while performing these activities. To the best that we know, EA-Miner [4] is the main tool for concern identification and classification; however, it is not available in a public way. In addition, EA-Miner depends on the WMATRIX [25] tool, which is responsible for running natural language processing routines for English-only texts.

This work differs from those above mentioned, because it (i) proposes a conceptual model (O4C ontology) for the software concern domain, aiming to make clear and precise the description of the concepts of this domain; (ii) proposes the building and the usage of software concern 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; (iii) provides a set of activities and heuristics to guide software engineers while using the software concern catalogs; (iv) 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; and (v) proposes a language-independent tool for concern identification and classification.

Regarding 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 of them was specific to the context of AORE. One of the closest works related to this paper is that one proposed by López et al. [27]. 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. [27] 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, the relationships among software concerns and their keywords, the decomposition of concerns into sub-concerns, among others; (ii) their work does not present neither a set of activities/guidelines that helps software engineers on how to use the proposed ontology instances nor a computational tool for automating their proposal; and (iii) this work does not present any type of an experimental study on their proposal.

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 related to this domain, (ii) which are the relationships among 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 concern 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 type of conceptual model, which aims to make clear and precise the description of a domain with the purpose of communication, learning, and problem solving [28]. The development of O4C considered (i) the existing works regarding AORE, gathered by the authors of this paper from a systematic mapping of literature [9, 10] and (ii) the expertise of two researchers that have worked with AORE for 12 years. Moreover, the O4C ontology was developed in accordance with (i) the approach for ontology development, called SABiO (Systematic Approach for Building Ontologies) [28], and (ii) an UML profile for ontology modeling, called OntoUML [29]. A preliminary version of this ontology was proposed in [6, 7, 11]; in this paper, the final version of O4C is formalized and described according to SABiO and OntoUML approaches.

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

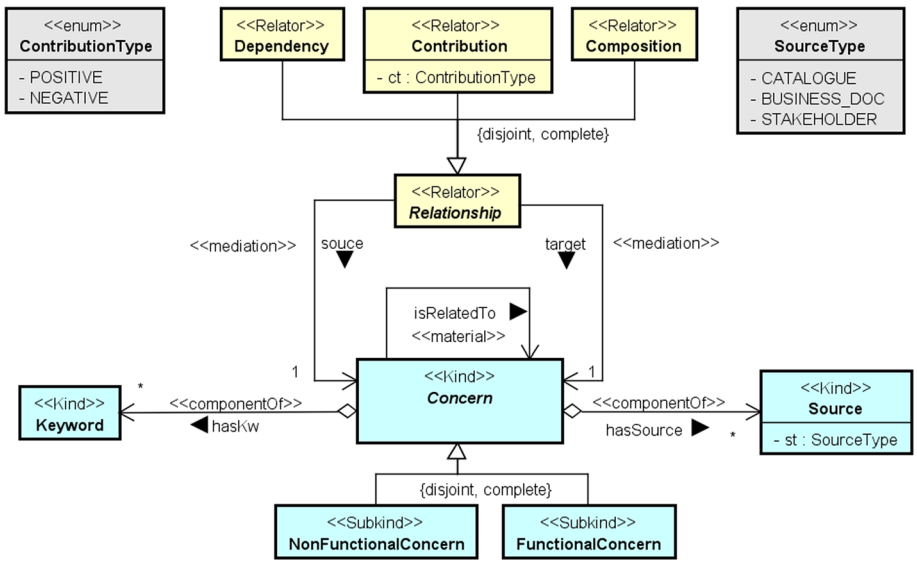

Fig. 1 Ontology for Concerns (O4C)

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" regards to concerns that are related to the functional features of the software, such as "Payment" and "Order Management." The "NonFunctionalConcern" concept, in turn, corresponds to concerns related to then non-functional features of the software, such as "Security," "Persistence," and "Logging."

The "Concern" class is stereotyped with «Kind» and their subclasses have the «Subkind» stereotype. In accordance to OntoUML [29], 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 it, 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 inherits this 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, since they inherit the identity principle of "Person."

The "Concern," "FunctionalConcern," and "NonFunctionalConcern" concepts are well-known in the AORE community and are reported in several studies [2–4, 12–18, 22, 23].

Keyword and Source

The "Keyword" concept appears in some AORE approaches [2, 22, 23]; however, none of the analyzed works presented the idea of storing these keywords in order to use them in further projects. In the O4C ontology, this concept was created aiming to store the keywords commonly used to identify a particular software concern. For example, "save," "update," and "persist" may be used to provide indications of the existence of the "Persistence" concern.

The idea represented by the "Source" class, its "st" attribute, and the "SourceType" enumerated class (Fig. 1) regards 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 activities, because they can help the software engineer to identify who or what needs to be consulted when a particular concern is not being correctly identified.

According to Agostinho et al. [12], Brito and Moreira [16], and Whittle and Araújo [18], 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 [19], Chung and Leite [20], Cysneiro [21], among others; or (iii) business documents, such as a security protocol of a company.

Contribution, Dependency, and Composition

The possible types of software concern relationships are represented by the "Contribution," "Dependency," and "Composition" concepts (sub-concepts of "Relationship"). The classes that represent these concepts were stereotyped with «Relator». In OntoUML, "Relators" are mediator elements, i.e., elements that mediate the relationship among other ones, making it real. In Fig. 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 "Relationship" (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 engineers on which concerns are really present in the software and which are the more appropriated strategies for modularizing them. For instance, the "Security" concern may be decomposed into "Authorization," "Encryption," among others. There may be the "Authorization" sub-concern in a specific software but not the "Encryption" sub-concern.

The "Dependency" concept 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" needs to be there as well. This type of information is important, because (i) it allows the software engineer to explore other software concerns, before being 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 engineers 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.

The "Contribution" concept 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 may be found among the "Concurrency," "Performance," and "Cost" concerns: the implementation of concurrency mechanisms in the software may positively contribute to the software performance. On the other hand, this may negatively contribute to the project cost. This type of relationship may be used as a guide to help software engineers to deal with implicit software concerns, as presented in later sections of this paper.

The "Contribution," "Composition," and "Dependency" concepts are quite divergently presented in the related works. The idea represented by the "Contribution" concept is reported in the approaches proposed by Moreira et al. [17] and Soeiro et al. [3]. 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. [17] 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. [3] and the "Composition" concept was not found in the analyzed works.

In all previous discussed cases, the cited 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.

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 the concern identification and classification.

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 [19], Chung and Leite [20], and Cysneiro [21]; (ii) the knowledge of experts on AORE; (iii) business documents, such as security and privacy protocols, pattern language, among others; or (iii) historical data of previous projects.

Figures 2 and 3 present two examples of O4C-based catalogs, represented by UML class diagrams. In both diagrams, 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 [30–32]. Health Watcher is an information system that aims to store complaints regarding 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 28 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 catalog of Fig. 3, in turn, was built from the concepts represented in a pattern language, called business resource management [33]. This pattern language was designed to assist the development of information systems in the business resource management domain. This catalog has seven functional concerns, 17 keywords, and five relationships: (i) three compositions between the "Transaction" and "Rental," "Transaction" and "Commercialization," and "Transaction" and "Reservation" software concerns and (ii) two dependencies between the "Payment" and "Transaction" and "Delivery" and "Payment" concerns.

By combining the non-functional concerns of the catalog presented in Fig. 2 with all concerns of the catalog of Fig. 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. The "Performing the concern identification and classification" section of this paper presents how to use a software concern

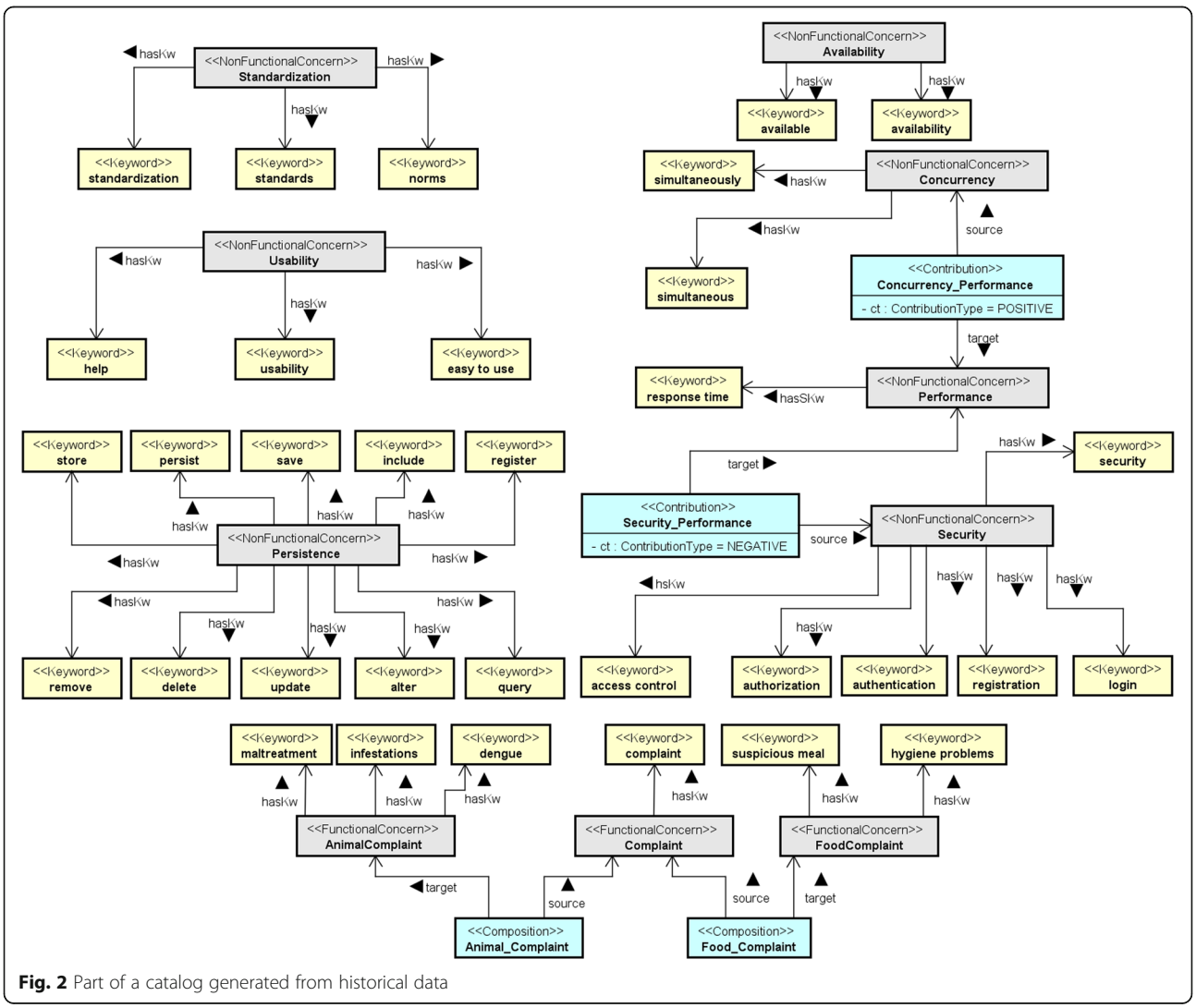

Fig. 2 Part of a catalog generated from historical data

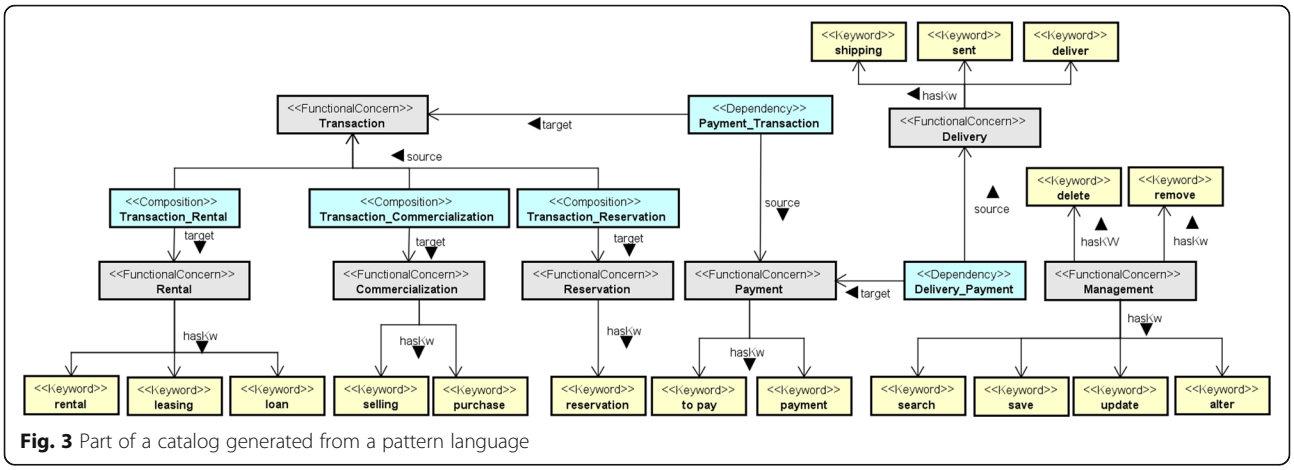

Fig. 3 Part of a catalog generated from a pattern language

catalog, such as those previously presented, in order to identify and classify software concerns from requirements documents. It is important to state that the process of concern identification and classification may implicate in changes on the catalog of software concerns. Hence, this phase, "preparing the catalog of software concerns", must be faced as a part of an iterative and incremental process of AORE.

Preparing the requirements document

This phase allows the software engineers to obtain/prepare/update the requirements document on which the concern identification and classification will occur. 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 this information is 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, 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 can be found in Health Watcher [30–32].

Table 1 Part of the Health Watcher requirements document

Identifier	Type	Requirement description	Dependencies
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 online HELP to be consulted by any person that uses it.	–
NFR-02	NFR	The response time must not exceed 5 s.	–

FR functional requirement, *NFR* non-functional requirement

Performing the 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 (Fig. 4) (i) identifying concerns from keywords, (ii) identifying concerns from the interdependence among software requirements, (iii) specifying the main concerns, (iv) verifying the results of concern identification, and (v) classifying concerns.

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 cataloged concern in the description of the software requirements. If any keyword of a particular concern is in the description of a software requirement, it is stated that this concern affects (is related) to the requirement in analysis.

As may be seen in Fig. 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 Fig. 2 as inputs, after executing this activity, the list of requirements and related concerns presented in Table 2 is generated. The list of requirements and related concerns will be increased with new types of concerns in the next activities of the approach. This is possible due to the other types of resources and mechanisms proposed by ObasCId, such as

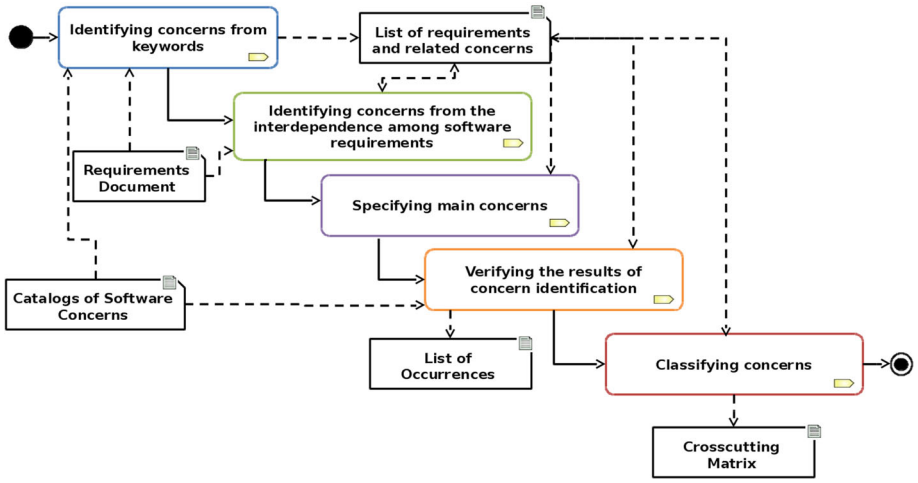

Fig. 4 Overview of the "Performing the concern identification and classification" phase

the identification of concerns based on the interdependence among software requirements.

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 needed.

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.

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-01	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-02	Concerns
The *response time* must not exceed 5 s.	Performance

The keywords stored in the catalog that match the requirement descriptions are italicized

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 in the "Specifying the main concerns" section of this paper.

Specifying the main concerns

In this activity, the software engineer must inform what is the main concern of each software requirement. A *main 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 the main concerns is important for the concern classification activity, 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 it is difficult to decide which is its main concern, the software engineer may consider rewriting this requirement. It is also important to state that having "a requirement with only one

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-01	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-02	Concerns	Main concern
The *response time* must not exceed 5 s.	Performance	X

concern" does not mean that this concern is the main concern of the requirement, since the identified concern may be a false positive. Hence, it is important that the software engineer checks the requirements with only one concern, before deciding about its main concern.

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 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 concern identification process. To produce this list, the software engineer must check 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 states 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.

ObasCId approach also provides, for each heuristic, a set of suggestions for solving the occurrence generated by this heuristic. The goal of these suggestions is to let the software engineers aim to make more appropriated decisions on how to deal with these occurrences. Due to space limitation, only the suggestions for heuristic 3 are presented below. The suggestions of the remaining heuristics may be found in [24]:

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 should be 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. This is especially important in cases where implicit concerns in the software exist.
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 Fig. 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 [8]. 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.

- 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 Fig. 2, it will generate an occurrence derived from heuristic 2, since the "Concurrency" concern contributes positively to "Performance" (according to the catalog of software concerns), but "Concurrency"

was not identified in the software requirements. This fact indicates that "Concurrency" may be a candidate of an implicit concern and the software engineers must discuss about the relevance of this concern in the software under analysis. In this case, we will just ignore this occurrence.

According to Fig. 4, if needed, the software engineers may go 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.

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. A crosscutting matrix is a "Concern vs. Concern" matrix and, when a cell "[C1, C2]" is highlighted, this indicates that "C2" affects (cut across) "C1."

In this activity, we can assume that each software requirement has a main concern "MC"(defined in the "specify the main concerns" activity—"Specifying the main concerns" section) and a set of zero or more related concerns {"C1," "C2," … "Cn"}. In the ObasCId approach, we consider that all concerns "C1," "C2," … "Cn" cut across the main concern "MC." Hence, all cells "[MC, C1]," "[MC, C2]," … "[MC, Cn]" must be highlighted. If a requirement is related only to its main concern, no 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.

Based on Table 3, it is possible to notice that the "FR-01" requirement, whose main concern is "Complaint," is related to "Persistence," "Usability," and "Performance" concerns. Hence, the cells "[Complaint, Persistence]," "[Complaint, Usability]," and "[Complaint, Performance]" of Table 5 were marked with an "X" symbol.

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 likelihood of "A" to be a crosscutting concern. To know which requirements are affected by a specific concern, the list of requirements and related concerns (Table 3) may be used.

Table 5 Crosscutting matrix

↓ Main conc./concerns →	1: Persist.	2: Compl.	3: Usab.	4: Perfor.
1: Persistence				
2: Complaint	X		X	X
3: Usability				
4: Performance				

In an ideal scenario, each concern should only affect requirements for which it is its main concern. In other words, the column related to this concern should contain only empty cells. Hence, in the ObasCId approach, all columns with at least an "X" symbol regard to *crosscutting concerns candidates*. In the case of Table5, all concerns, except the "Complaint" (column 2), are considered crosscutting concern candidates.

The crosscutting matrix proposed in this paper is similar to that presented in the approach proposed by Rashid et al. [2]. 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 [17].

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

ObasCId-Tool

Baniassad and Clarke [22] argue that the intuition or even the domain knowledge of a software engineer is not sufficiently enough to identify potential crosscutting concerns in medium and large software products within a reasonable period of time. Sampaio et al. [8] corroborate this opinion, reinforcing the importance of the existence of computational tools to improve the effectiveness of AORE approaches. In this context, a computational tool that automates several activities and artifacts of the ObasCId approach, called ObasCId-Tool, was developed.

ObasCId-Tool overview

ObasCId-Tool is a responsive web-based tool, freely available at http://obascidtool-obascidtool.1d35.starter-us-east-1.openshiftapps.com/. It was developed based on the following technologies: (i) the JavaServer Faces (JSF) specification and the framework Mojarra (https://javaserverfaces.java.net/) that implements the JSF specification; (ii) a CSS framework for development of responsive web-based applications, called Bootstrap (http://getbootstrap.com/); (iii) the MySQL database management system; and (iv) the Apache Lucene search engine (https://lucene.apache.org/core/), used for the implementation of the concern identification algorithms.

It is also important to highlight that, unlike EA-Miner, ObasCId-Tool is a language-independent concern identification tool, that is, the researcher may build catalogs and requirements documents, as well as perform the concern identification and classification in the language of his/her interest. In addition, ObasCId-Tool implements internationalization (i18n) resources and its graphical user interface is available in English and Brazilian Portuguese.

Figure 5 presents the architecture of the ObasCId-Tool, highlighting its main components, as well as the dependencies existing among them. In this figure, the rectangular boxes specify the tool's modules and the dotted arrows indicate the dependencies among these modules. A cylinder represents a data repository maintained/used by the tool. ObasCId-Tool consists of five modules: (i) Repositories Query Module; (ii) Researchers Management Module; (iii) Concerns Catalogs Management Module; (iv) Requirements Documents Management Module; and (v) Concern Identification and Classification Module.

In order to use the modules (iii), (iv), and (v), the researcher must be signed up in the tool. Hence, these modules depend, directly or indirectly, on the "Researchers Management Module." A registered researcher may create and maintain catalogs of software concerns and requirements documents, using the "Concerns Catalogs Management" and the "Requirements Document Management" modules.

The "Repositories Query Module" may be used by any user (registered or not) interested in the ObasCId-Tool. This module provides options for querying and viewing public catalogs and requirements documents stored in the tool. In order to identify and classify the software concerns, the researcher must have at least one catalog of software concerns and a requirements document stored in his/her account. Hence, the "Concern Identification and Classification Module" depends on the "Concerns Catalogs Management" and the "Requirements Document Management" modules. There is no explicit dependency between these modules, so the researcher may freely manage their requirements documents and catalogs of software concerns, in his/her own way. Due to space limitations, this paper only presents more details about the "Concern Identification and Classification," "Concerns Catalogs Management," and "Requirements Documents Management" modules. Information about the other modules may be found at Parreira Júnior [24].

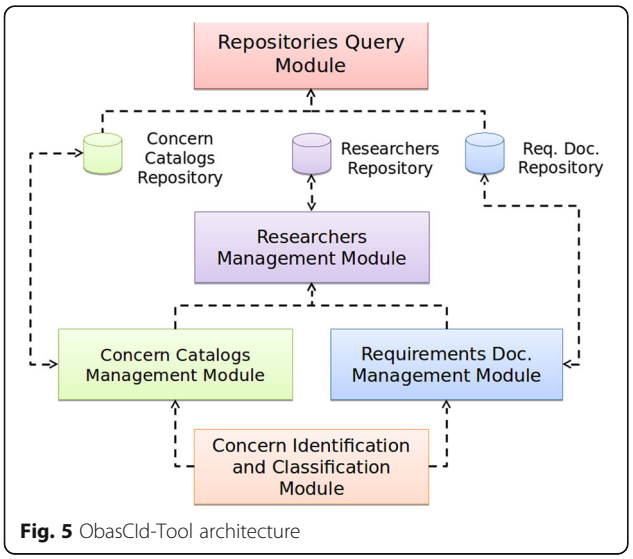

Fig. 5 ObasCId-Tool architecture

Concerns catalog management module

This module allows the researcher to build and maintain catalogs of software concerns. It represents a wizard for instantiation of the concepts and relationships defined in O4C ontology ("Ontology for Concerns (O4C)" section). To register a catalog of software concerns, the researcher must provide a unique name for this catalog and its type of license, which may be "Private" or "Public." Optionally, the user may inform a description of his/her catalog. A "Public" catalog allows anyone to query and view its data. If this person was a registered user, he/she may also import this catalog into his/her personal account. The "Private" type of license restricts the usage of the catalog to the context of its owner's account. Hence, it may not be queried or imported by the other users of the tool. This is useful when the researcher is preparing his/her catalog and does not want to make it available until he/she finalizes it. The concept of public/private license is applied to the requirements documents as well.

Management of software concerns

A concern catalog may have several software concerns. To register a concern, the researcher must inform the concern name that must be unique in the context of the current catalog and the type of this concern, which may be "Functional" or "Non-functional." As optional information, the researcher may provide a description of the concern.

Another important resource of a software concern is its keywords. Figure 6 presents the screen used for registering the keywords of a software concern. In order to register a new keyword, the researcher must inform its description that must be unique in the context of the current concern. Keywords with more than one word, such as "response time" and "access control," are allowed and they should be enclosed in double quotation marks. In the case of keywords, the description represents the keyword itself. The term "description" was used because

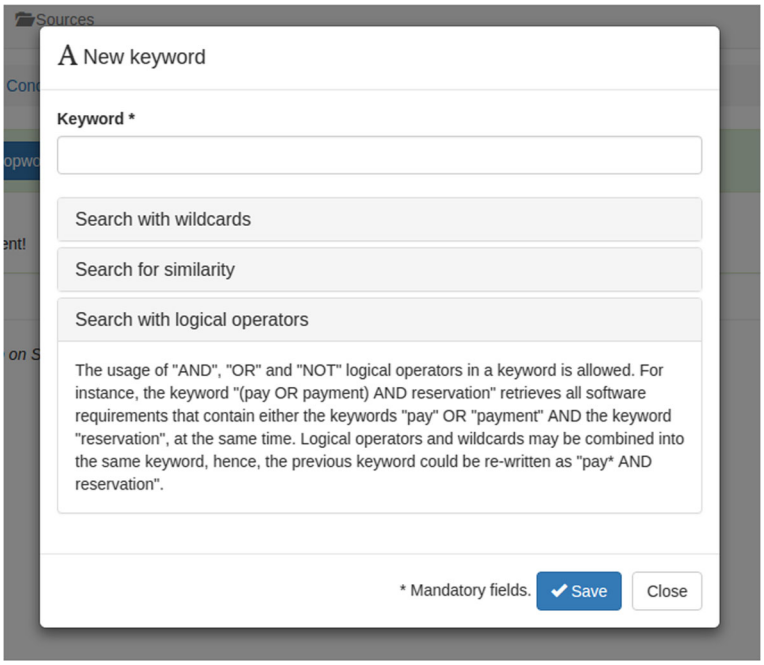

Fig. 6 Keywords in ObasCId-Tool

a keyword may be composed by a set of terms separated by logic operators, as explained bellow.

In addition, the description of a keyword may not belong to the researcher's list of stopwords. A pre-defined list of stopwords for English and Brazilian Portuguese languages, obtained from well-known public stopword repositories [34], is automatically registered for each new researcher. If the researcher is sure about the inclusion of a keyword that appears in his/her list of stopwords, he/she may update his/her list of stopwords. Stopwords are words that have no intrinsic meaning, and hence, they are not suitable for identification of specific concepts [35], such as software concerns. The usage of stopword lists in the ObasCId-Tool is useful because (i) it serves as a guide for the registration of more appropriated keywords and (ii) it may avoid the incidence of several false positives during concern identification process.

The concept of keywords is an important resource for the identification of concerns from the requirements document. Therefore, ObasCId-Tool provides three mechanisms for creating keywords that may contribute more effectively to this activity. These mechanisms are (i) search with wildcards, (ii) search for similarity, and (iii) search with logical operators. All these mechanisms are support by Apache Lucene tool.

Figure 6 presents a description on how to use logical operators for building keywords. Regarding wildcards, the following wildcards may be used in a keyword: "?"—keywords that match with until one character replacement. For instance, the keywords "test" and "text" are captured by the expression "te?t"; "*"—keywords that contain zero or more characters at the position where the wildcard is. For instance, the keywords "tests" and "tester" are captured by the expression "test*." These wildcards may be used in any position of a keyword, except in its beginning.

Similar keywords may be found from a search for similarity. This type of search uses the well-known "Levenshtein Distance" algorithm that returns a value between 0 and 1, where the more the value is close to 1 the more similar are the compared keywords. To use this resource, the researcher must append the "~" symbol at the end of the keyword of interest, along with the threshold required to accept another keyword as similar to its. For instance, "roam~0.5" will capture keywords with a "Levenshtein Distance" equal or higher than 0.5, such as "roma" keyword.

Management of relationships among software concerns

Relationships among different concerns in a catalog may be created and maintained in the ObasCId-Tool. As explained in the "Ontology for Concerns (O4C)" section, the possible types of relationships are "Composition," "Dependency," "Negative Contribution," and "Positive Contribution." To register a new relationship, the following information must be provided (Fig. 7): (i) the source concern of the relationship, (ii) the target concern of the relationship, and (iii) the type of the relationship. The meaning of the target and source concerns depends on

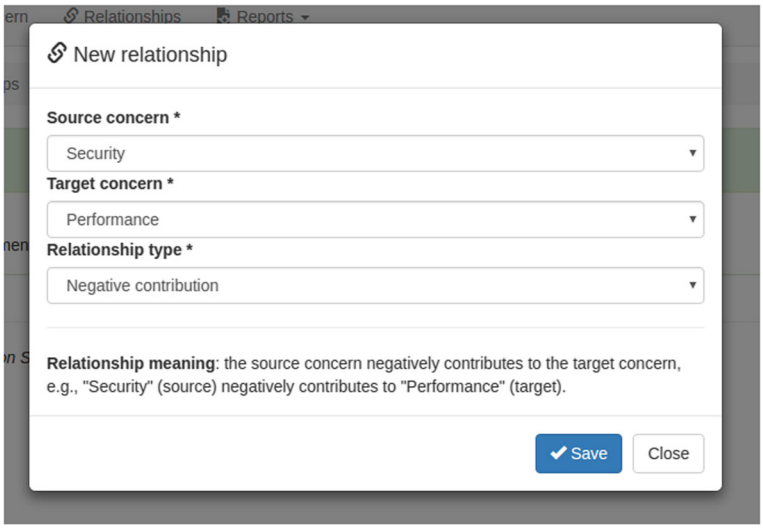

Fig. 7 Concern relationships in ObasCId-Tool

the type of the relationship. For example, in a dependency relationship, the source concern depends on the target concern. In the case of a negative contribution, the source concern negatively contributes to the target concern. To make clear this understanding, ObasCId-Tool presents the meaning of each relationship type along with a practical example of its usage, as can be seen at the bottom of Fig. 7.

Requirements documents management module

The focus of the ObasCId-Tool is identifying and classifying concerns from software requirements. Hence, an important feature of this tool is the management of requirements documents. As the process of managing requirements documents are quite similar to the process of managing catalogs of software concerns, we will not present it in this paper. More details about this may be found at Parreira Júnior [24].

Concern identification and classification module

This module implements the five activities proposed for the concern identification and classification phase of the ObasCId approach. To implement these activities, the concept of "Identification Unit" was proposed. Each identification unit has (i) a name that must be unique, (ii) a requirements document on which the concerns will be identified, and (iii) a catalog of software concerns that will be used in the process of concern identification and classification.

Figure 8 illustrates the results of the execution of an identification unit, taking into consideration the requirements document present in Table 1, as well as the catalog of software concerns of Fig. 2.

Figure 8a presents a list of occurrences generated after performing the concern identification activities. Figure 8b presents the name of the requirements document on which the concerns were identified and a summary of the identification process, highlighting (i) the list of concerns of the used catalog, (ii) the list of identified concerns, (iii) the amount of software requirements, (iv) the amount of requirements affected by software concerns, among others. Figure 8c highlights the "requirements filter" functionality that will be explained later in this text. Finally, Fig. 8d illustrates the list of software requirements along with the concerns that affect them (the main concern of each requirement is highlighted with the string "main concern"). Besides the name of the concern, the part of the description of the requirement that allowed the identification of this concern is presented.

To obtain these results, each software requirement under analysis was indexed as a document in Apache Lucene tool and, for each keyword in the catalog of software concerns, a search was performed using this tool as well. The results of these searches are gathered in the ObasCId-Tool data structures to be further presented to the researchers.

Considering that the identification process is complete, the researcher may verify the crosscutting behavior of the identified concerns. To do this, he/she may access the "Crosscutting Matrix" generated by ObasCId-Tool. Figure 9 displays the crosscutting matrix generated with the data of the previous example.

If a researcher wants to know what are the software requirements affected by a specific concern, he/she must return to the list of requirements and related concerns (Fig. 8) and apply the "requirements filter." For example, in order to know what are the

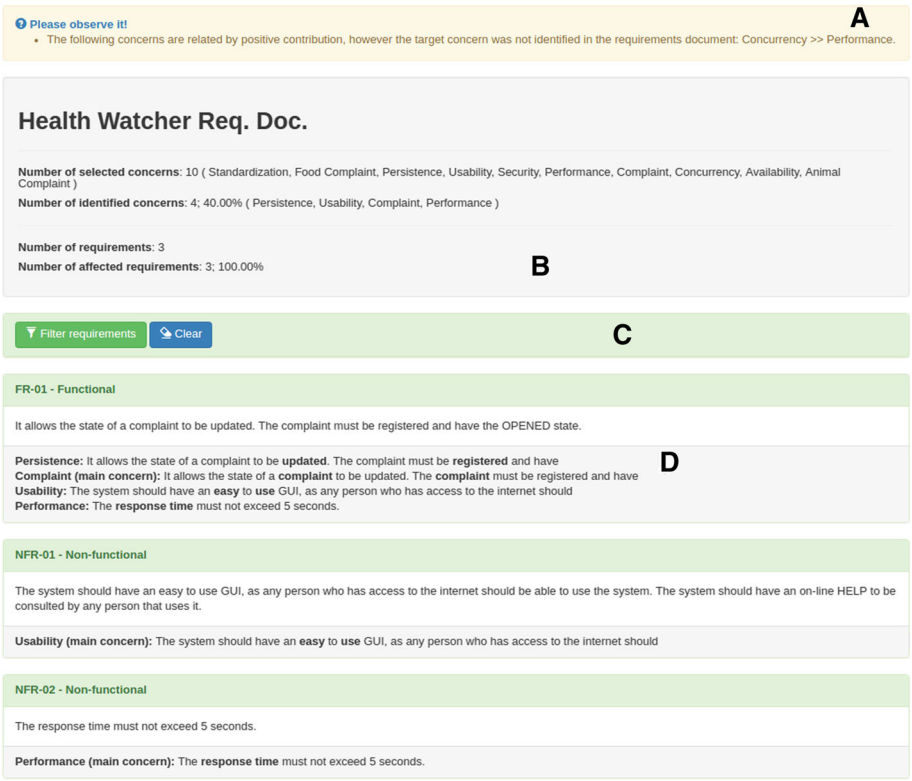

Fig. 8 Results of the concern identification process in ObasCId-Tool

requirements affected by the "Persistence" concern, when the "Complaint" is the main concern, the researcher must initially filter the requirements by choosing "Complaint" as the main concern and "Persistence" as the crosscutting concern, as may be seen in Fig. 10a. Hence, only the requirements compatible with this filter will be presented (Fig. 10b.

The researcher may also inform either the main or the crosscutting concern in a filter. This allows the researcher to find out (i) what are the requirements

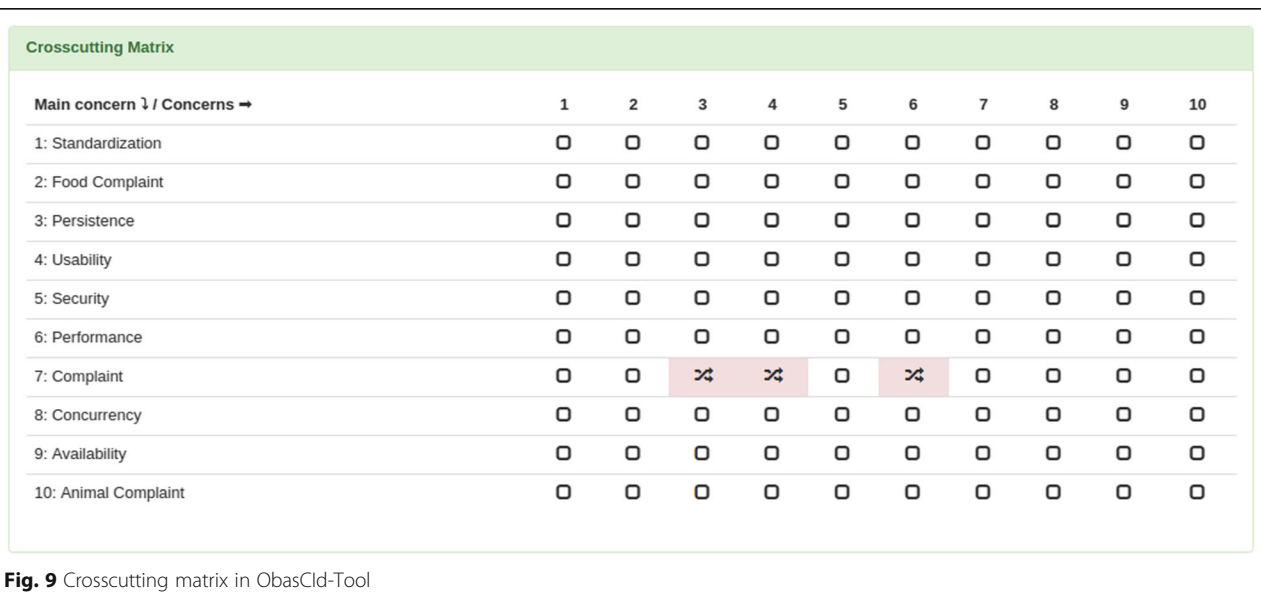

Fig. 9 Crosscutting matrix in ObasCId-Tool

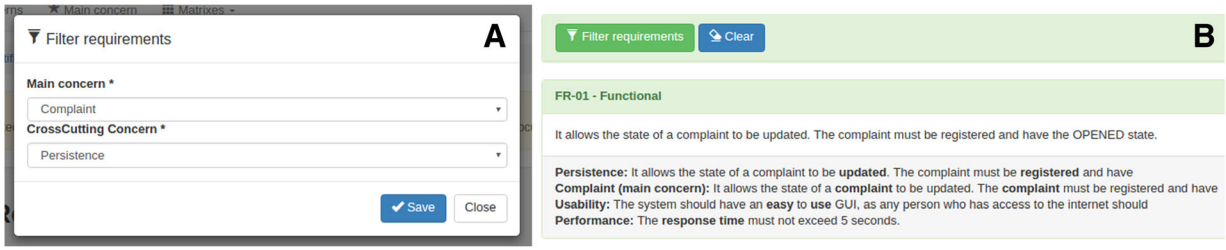

Fig. 10 "Requirements filter" in ObasCId-Tool

affected by a specific crosscutting concern or (ii) what are the requirements of a particular main concern.

Quasi-experimental study I

For the assessment of the ObasCId approach, the following GQM (goal, question, metric)-based goal [36] 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 undergraduates and graduates 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 [22, 23]. It is important to highlight that all resources used in the quasi-experimental studies presented in this paper are available at Parreira Júnior [24]—an English version of these resources may be found in https://goo.gl/cs13QZ. These studies are classified as quasi-experimental ones due to the non-randomization of the participant selection. Actually, the participants of these studies were selected through a non-probability for convenience sampling.

Theme/Doc overview

The Theme/Doc [22, 23] 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; it 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 (the more important action) of these requirements. Once the primary action of a requirement is defined, 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 approach. 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 [4], 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 [5].

Planning of the quasi-experimental study

The planning of this quasi-experimental study was defined according to Wohlin's proposal [37] and involves the following steps: (i) context selection, (ii) hypotheses formulation, (iii) variable selection, (iv) selection of the participants, and (v) design and execution of the quasi-experimental study.

a) Context selection

This quasi-experimental study was conducted with 24 undergraduate and graduate students in Computer

Science from three federal universities from Brazil (UFG, UFLA, and UFSCar). The requirements documents of Health Watcher [30–32] and of an information system for DVD rental (LocaDVD—[38]) 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 to be used with catalogs for software concerns created from the pattern language proposed by Braga et al. [33], such as the catalog of Fig. 3.

b) Hypotheses formulation

An important part of the hypotheses formulation step is the specification of the metrics that will be used in the quasi-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 quasi-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).

c) Variable and participant selection

Independent variables are those manipulated and controlled during the quasi-experimental study. In this study, the two independent variables are (i) the approach for concern identification and classification (ObasCId and Theme/Doc) and (ii) the software systems used in the study (Health Watcher and LocaDVD). 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.

Table 6 Hypotheses used in the quasi-experimental study I

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

d) Design and execution of the quasi-experimental study

The distribution of the participants was performed aiming to form two homogeneous groups, regarding the participants' expertise. Each group had 12 participants and their expertise were 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. Before starting the experimental study, a 120-min training about the main concepts of AORE and the ObasCId/Theme approaches was performed, in order to homogenize the knowledge of participants. In this training, the participants had a practical experience on the concern identification and classification process through exercises with ObasCId/Theme approaches. During the training, it was not informed to the participants what approach was developed by the authors of this paper. It is important to state this study was performed in three different moments (each one at a different university) and the collected data were gathered to be analyzed and discussed.

The execution of the quasi-experimental study occurred in two phases. In the first phase, participants should identify the non-functional concerns presented in the requirements document of the Health Watcher and classify them as crosscutting or non-crosscutting. To do this, 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, 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 Fig. 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 to be a source that could be used to generate a catalog of functional concerns

regarding 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 systems was similar. To calculate the values of the recall and precision metrics, it considered the amount of concern correctly identified and classified by each participant, individually.

Results and discussion

Table 7 presents the results obtained by both groups of participants, regarding the Health Watcher software (first phase). Taking into account the values for recall, the participants who used the ObasCId approach had, on mean, more promising results than those who used Theme/Doc. It is also possible to notice that there is no relevant difference between the two approaches, regarding the precision.

Table 7 still presents that the execution time provided by ObasCId (48.50 min) was higher than that one provided by Theme/Doc approach (40.50 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 (8 min) 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.

The same type of information presented for the Health Watcher is also presented for LocaDVD (second phase), as can be seen in Table 8.

Some important facts about the results of Tables 7 and 8 are:

- Sampaio et al. [8] 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;
- 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 systems of 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
- 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

Table 7 Quasi-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)
P01	42.85	75.00	43.00	P13	71.42	71.00	62.00
P02	42.85	100.00	48.00	P14	85.71	100.00	39.00
P03	42.85	100.00	49.00	P15	85.71	100.00	54.00
P04	28.57	66.00	48.00	P16	71.42	100.00	37.00
P05	57.14	80.00	36.00	P17	57.14	75.00	43.00
P06	42.85	100.00	31.00	P18	71.42	80.00	42.00
P07	28.57	100.00	34.00	P19	71.42	100.00	42.00
P08	35.76	75.00	38.00	P20	77.42	90.00	45.00
P09	27.16	66.00	60.00	P21	60.00	75.00	60.00
P10	30.32	80.00	39.00	P22	67.00	75.00	58.00
P11	55.55	80.00	30.00	P23	72.00	100.00	43.00
P12	60.60	100.00	30.00	P24	66.00	75.00	57.00
Avg.	41.25	85.16	40.50	Avg.	71.38	86.75	48.50

Table 8 Quasi-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)
P01	83.00	83.00	32.00	P13	33.00	66.00	18.00
P02	83.00	71.00	22.00	P14	66.00	80.00	29.00
P03	100.00	75.00	18.00	P15	66.00	80.00	15.00
P04	66.00	100.00	42.00	P16	33.00	100.00	32.00
P05	66.00	80.00	37.00	P17	71.00	71.00	13.00
P06	100.00	86.00	22.00	P18	50.00	60.00	18.00
P07	83.00	71.00	25.00	P19	50.00	75.00	21.00
P08	85.00	75.00	27.00	P20	50.00	60.00	17.00
P09	75.00	75.00	42.00	P21	66.00	80.00	30.00
P10	100.00	75.00	30.00	P22	33.00	80.00	27.00
P11	70.00	100.00	27.00	P23	50.00	75.00	21.00
P12	85.00	60.00	22.00	P24	33.00	60.00	25.00
Avg.	83.00	79.25	28.83	Avg.	50.08	74.00	22.16

higher than that provided by the Theme/Doc; however, the difference was not significant.

Aiming to reinforce the experimental results, we replicated the quasi-experiment presented in this section, using the requirements document of the ObasCId-Tool. More information about this are in the Appendix of this work.

Hypothesis tests

To verify the hypotheses defined in Tables 7 and 8, the *t* test was applied [39]. Regarding the Health Watcher software, comparing the mean values for recall provided by the approaches Theme/Doc (mean = 41.25) and ObasCId (mean = 71.38), the H_{0Re} null hypothesis may be rejected with significance level of 99.9% (p value = 0.00004). This situation also happens for the LocaDVD software, regarding the recall. Regarding the mean 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 systems of software, we obtained the same situation for precision values.

Threats to validity

The main threats to validity of this study are:

- Conclusion and construct validities

These types of threats refer 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 [39], and the values for recall, precision, and time metrics were considered normalized with a significance level of 99.9%. Moreover, in this study, we did not take into consideration the effort and tiredness level of the participants, which may influence the results of the experiment. For example, after 120 min of training and 60 min of work with the approaches, participants' capability of making correct decisions was certainly impaired.

- 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 quasi-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.

- 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 concern catalogs and the requirements documents) presented to the participants—we performed a pilot study, with different participants, aiming to

evaluate the quality of the resources used in this quasi-experimental study; (iii) the amount of participants of the study—aiming to mitigate this issue, we performed a new quasi-experimental study and presented it in the Appendix of this paper; and (iv) the usage of undergraduate and graduate students in Computer Science.

Quasi-experimental study II

The evaluation goal of this second quasi-experimental study was to analyze the usage of the ObasCId-Tool, in order to evaluate it, with respect to its ease of use and its utility, from the point of view of software engineers, in the context of a group of undergraduate and graduate students in Computer Science.

Planning of the quasi-experimental study

The planning of this experiment was also carried out according to the model proposed by Wohlin et al. [37].

a) Context selection

The context of this study consists in the usage of ObasCId-Tool, aiming to manage the necessary resources for software concern identification and classification. This study was carried out with 24 undergraduate and graduate students in Computer Science from three Brazilian Universities (UGF, UFLA, and UFSCar).

b) Hypotheses formulation

In order to evaluate the ease of use and the utility of ObasCId-Tool, the TAM (technology acceptance model—Davis, [40]) was used. This model aims to explain the behavior of people regarding the acceptance of a technology and has been used in recent studies [41] for software product assessments. The TAM model defines main constructs [40]: (i) perceived utility, which measures how much a person believes that using a given technology increases his/her productivity and (ii) perceived ease of use, which measures how much a person believes that the usage of a given technology is easy.

It also suggests the construction of questionnaires with statements regarding the ease of use and the utility of the technology under analysis. For each statement, the respondent should choose one of the following options, according to his/her opinion about it: "totally disagree," "strongly disagree," "partially disagree," "neutral," "partially agree," "strongly agree," and "totally agree." In this context, seven metrics were proposed: (M1) percentage of participants that assigned the "totally disagree" option, (M2) percentage of participants that assigned the "strongly disagree" option, (M3) percentage of participants that assigned the "partially disagree" option, (M4) percentage of participants that assigned the "neutral" option, (M5) percentage of participants that assigned the "partially agree" option, (M6) percentage of participants that assigned the "strongly agree" option, and (M7) percentage of participants that assigned the "totally agree" option. The statements of the questionnaires proposed in this work are presented in Tables 10 and 11, along with the results of this quasi-experimental study.

Based on seven metrics, four hypotheses were elaborated for this study, two related to the utility construct, and two related to the ease of use construct (Table 9).

c) Selection of variables and participants

The dependent variables under analysis in this quasi-experimental study are the percentages of responses for each statement of the TAM-based questionnaire. Fourteen of the 24 participants were undergraduate students and ten were graduate students. In terms of AORE experience, according to the results of the profile characterization questionnaire, all the participants had low levels of knowledge on this subject. When asked about their level of knowledge about requirements engineering, all the participants stated that they were at an intermediate level, whose contact with this subject occurred through software engineering disciplines.

d) Design and execution of the experiment

It is important to notice that none of the participants had previously used ObasCId-Tool. Hence, before the beginning of the experiment, a 40-min training was conducted, aiming to present this tool to the them. In addition, a 90-min training about the main concepts of AORE and about the ObasCId approach was performed. In this training, the participants had a practical experience on the concern identification and classification process through exercises with ObasCId approach. This was done so that the participants could have a more conscious opinion about the statements presented in Tables 10 and 11.

In the execution phase of the quasi-experimental study, the participants should perform a series of activities in the ObasCId-Tool, which consisted in managing the needed resources to identify and classify concerns from software requirements. Some examples of activities were "registering a new catalog," "registering a new concern for an existing catalog," among others. In addition, after finishing all activities, the participants were asked to fill out a questionnaire that contained the statements presented in Tables 10 and 11.

Results and discussion

Tables 10 and 11 present, respectively, the results obtained through two electronic questionnaires filled out

Table 9 Hypotheses used in the quasi-experimental study II

Hypotheses for Recall	
H_{0U}	There is no consensus on the utility construct, regarding the usage of the ObasCId-Tool, i.e., H_{0U}: M1 + M2 + M3 + M4 = M5 + M6 + M7.
H_{1U}	There is a consensus on the utility construct, regarding the usage of the ObasCId-Tool, i.e., H_{1U}: M1 + M2 + M3 + M4 ≠ M5 + M6 + M7.
Hypotheses for Precision	
H_{0EoU}	There is no consensus on the ease of use construct, regarding the usage of the ObasCId-Tool, i.e., H_{0EoU}: M1 + M2 + M3 + M4 = M5 + M6 + M7.
H_{1EoU}	There is a consensus on the utility construct, regarding the usage of the ObasCId-Tool, i.e., H_{1EoU}: M1 + M2 + M3 + M4 ≠ M5 + M6 + M7.

by the participants, with respect the utility and ease of use of ObasCId-Tool. The first column of these tables contains the statements presented to the participants; columns 2 to 8 show the percentage of participants who chose options 1 to 7, respectively. Finally, the ninth and tenth columns present, respectively, the percentage of participants who chose negative/neutral (1, 2, 3, or 4) and positive (5, 6, or 7) options.

In general, it may be noticed that, for both utility and ease of use constructs, the ObasCId-Tool obtained an amount of positive opinions higher than the negative/neutral ones. The most well-accepted statement for ease of use construct was "Using ObasCId-Tool is a good idea", while the statements with the worst results were the last five ones. This indicates that navigability and feedback mechanisms of ObasCId-Tool should be improved.

Regarding the utility construct, two statements had almost 100% of positive opinions: "ObasCId-Tool is useful in the process of concern identification and classification" and "I will recommend the usage of ObasCId-Tool." This indicates that the participants found in ObasCId-Tool potentially positive resources for the process of concern identification and classification. The statement with the lowest approval was "I intend to integrate ObasCId-Tool into my work routine." This is justified by the fact that the participants were students and they belonged to different research areas.

Hypothesis tests

To verify the validity or not of the null hypothesis regarding utility of ObasCId-Tool, H_{0U}, one should compare the amount of positive opinions about this construct to the amount of negative/neutral opinions. However, before doing this, it must be checked whether the sample data conforms to the normal probability distribution. Hence, the Shapiro-Wilk test [39] was applied to the set of positive and negative/neutral opinions presented in Table 10 and for both sets (negative/neutral and positive opinions); the data were not normally distributed, restricting the use of the t test. Hence, the Mann-Whitney test [39], a non-parametric test that does not require normally distributed data for its execution, was applied. Comparing the mean value of positive opinions about the ease of use of the ObasCId-Tool to the mean value of negative/neutral opinions, the null hypothesis H_{0U} could be rejected with degree of significance $p = 0.00094$. That is, with approximately 99.9% of confidence, it may be stated that the mean amount of positive opinions differs from the amount of negative/neutral opinions.

Regarding the ease of use construct, hypothesis H_{0EoU}, we verified the amount of the users' opinions regarding the ease of use of the tool. Initially, the Shapiro-Wilk test was applied to verify the normality of the sets of opinions of the users. It was verified that, for both sets (negative/neutral and positive opinions), the data were not normally distributed. Hence, the Mann-Whitney was applied to these data sets as well. Comparing the amount of positive opinions about the ease of use of the ObasCId-Tool to the amount of negative/neutral opinions, the first part of the null hypothesis H_{0EoU} could be rejected with degree of significance $p = 0.00452$.

Table 10 Quasi-experimental results—utility construct

Statements	Option (no. of participants)							Neg	Pos
	1	2	3	4	5	6	7		
I liked to work with ObasCId-Tool.	0	0	2	3	4	7	8	*5*	*19*
The access to ObasCId-Tool is simple.	0	1	2	1	4	6	10	*4*	*20*
Using ObasCId-Tool is a good idea.	0	0	1	1	7	7	8	*2*	*22*
In ObasCId-Tool, I always know where I am and how to arrive where I want.	0	0	3	4	7	6	4	*7*	*17*
The main functionalities of ObasCId-Tool are clear and easy to find.	0	0	0	8	5	6	5	*8*	*16*
My interaction with ObasCId-Tool is clear and understandable.	0	0	3	5	5	5	6	*8*	*16*
In ObasCId-Tool, I always know how to find the information that I need.	0	3	1	3	5	2	10	*7*	*17*
ObasCId-Tool has well-defined and understandable GUI elements.	0	0	2	4	4	4	10	*6*	*18*

1 totally disagree, *2* strongly disagree, *3* partially disagree, 4 neutral, *5* partially agree, *6* strongly agree, *7* totally agree

Table 11 Quasi-experimental results—ease of use construct

Statements	Option (no. of participants)							Neg	Pos
	1	2	3	4	5	6	7		
Using ObasCId-Tool is important and adds value to my work.	0	0	4	5	5	5	5	*9*	*15*
ObasCId-Tool is useful in the process of concern identification and classification.	0	0	0	1	3	8	12	*1*	*23*
Using ObasCId-Tool may improve the quality of the results of the concern identification and classification process.	0	0	1	1	2	9	11	*2*	*22*
ObasCId-Tool may improve my productivity while performing the concern identification and classification.	0	1	1	6	3	3	10	*8*	*16*
ObasCId-Tool produces the results that I hope of a tool for concern identification and classification.	0	1	3	8	4	4	4	*12*	*12*
I intend to integrate ObasCId-Tool into my work routine.	1	5	5	4	3	3	3	*15*	*9*
I will recommend the usage of ObasCId-Tool.	0	0	0	2	4	5	13	*2*	*22*
The main concepts of AORE were addressed by ObasCId-Tool.	0	0	0	3	5	10	6	*3*	*21*

1 totally disagree, *2* strongly disagree, *3* partially disagree, *4* neutral, *5* partially agree, *6* strongly agree, *7* totally agree

Since the percentage of negative/neutral opinions is lower than that of positive opinions, ObasCId-Tool can be considered satisfactory with respect to the utility and ease of use constructs.

Threats to validity

Conclusion and construct validities

Analogous to what was said for the quasi-experimental study I, an example of these types of threats is the choice of appropriated statistical methods for data analysis. In the case of this second study, two statistical tests were adopted: *t test* and *Mann-Whitney*. *t* test requires normally distributed data; hence, the Shapiro-Wilk normality test was applied to confirm this situation before the application of this statistical test. For the cases in which the Shapiro-Wilk test did not indicate normality of the data set, the Mann-Whitney test was applied. Regarding construct validity, it is important to state that we did not consider the results generated by the participants after finishing the activities proposed in this quasi-experiment. Hence, a participant may choose positive alternatives for the propositions presented in the TAM questionnaire related to ease of use; however, the activities performed by him/her were not well-concluded.

Internal validity

A point that may have influenced the results of this quasi-experimental study was the use of undergraduate and graduate students as participants. However, no expectations were expressed in favor or against the analyzed tool, so that the participants were not influenced. In addition, all students were submitted to the same fixed duration training so that none of them had privileges over the other ones.

External validity

Other factors that may have influenced the results of this quasi-experimental study are (i) the quality of the resources (forms, questionnaires, among others) presented to the participants—we performed a pilot study, with different participants, aiming to evaluate the quality of the resources used in this quasi-experimental study and (ii) the number of samples (participants) studied.

Final remarks

Based on what was previously presented in this paper, the main contributions of this work are (i) the O4C ontology, which provides a clear conceptual definition regarding the software concern domain, highlighting the key concepts and relationships involved in it. The usage of the concepts contained in O4C may collaborate with the building of AORE approaches, methods, and tools that are compatible with each other, since they are based on a common conceptualization; (ii) the ObasCId approach, which provides resources (activities, guidelines, heuristics, among others) that aim to help software engineers to perform the activities of the process of concern identification and classification in a more appropriated way; and (iii) the ObasCId-Tool computational support that provides support for building and sharing catalogs of software concerns, as well as for the identification and classification of software concerns from requirements documents.

As a main limitation of the ObasCId approach, we may cite the quality of the results of the concern identification and classification process proposed by this approach which strongly depends on the existence of good catalogs of software concerns. Furthermore, the cost/effort to maintain a good catalog may be impeditive, mainly for small companies. Moreover, the quality of the results of the "specify the main concerns" activity strongly depends on the expertise of the software engineers that are using the ObasCId approach. In addition, this activity is not automated by the ObasCId-Tool, what may affect the productivity of the development/maintenance team. Another limitation is the lack of experimental studies that compare the ObasCId(-Tool) to other AORE approaches/tools proposed in the literature.

Regarding the lack of computational support comparison, this was mainly due to the lack of available tools in the literature that support the concern identification and classification activities. Most of the proposed tools are for other AORE activities, such as concern representation and composition. EA-Miner tool provides support for concern identification and classification, but it is unavailable for public usage.

Based on the contributions and limitations previously presented, it is possible to propose some future works, such as extending the ObasCId approach, as well as the ObasCId-Tool, so that they may address other AORE activities, such as concern composition and conflict detection. The O4C ontology may be improved, as well as the ObasCId(-Tool), to allow the usage of other types of relationships among requirements in the process of concern identification and classification.

Another possible extension of the products proposed in this paper is to create methods for automatic transformation of catalogs of NFR requirements into O4C-based catalogs. The idea is to try to establishing relationships among the O4C concepts and the concepts of the NFR catalogs available in the literature, in order to allow the construction of this type of procedure. As stated before, the quality of the results provided by the ObasCId approach depends on the existence of good catalogs of software concerns. However, the cost/effort to create/maintain this catalog may be impeditive. The idea of using well-known and validated catalogs of NFR requirements to create instances of software concerns may help the developers to get a better start point to maintain their own catalogs of software concerns.

Regarding the dependency between the results of the "specify the main concerns" activity and the expertise of the software engineers that are using the ObasCId approach, we believe that creating heuristics for aiding software engineers to specify the main concern of a given requirement is an interesting work. Such heuristics will serve as guidelines to simplify the decision making of the software engineers, reducing the dependence of the quality of the results of the concern identification and classification process on the expertise of the professionals who apply it. In addition, the usage of heuristics could be considered as a starting point for the automation of this activity in the ObasCId-Tool.

Aiming to improve the analyst team productivity, someone may propose mechanisms for the collaborative construction of catalogs of software concerns, as well as for the control of different versions of these catalogs. These resources could improve the productivity of the team involved in the process of building catalogs, as well as facilitate the tracking of the changes made by each member of this team. We believe this purpose may reduce the cost/effort associated to the creation and maintenance of good software concern catalogs, the same way that collaborative/distributed software development may reduce the development/maintenance cost of a software product.

Finally, the last interesting work we propose as a future extension of this work is building domain-specific catalogs of software concerns. For example, with the help of experts in some domain, such as embedded systems and business documents related to this domain, it is possible to build catalogs of concerns to be imported and updated by other researchers/professionals in the ObasCId-Tool. From an economic perspective, these catalogs could be sold by companies specialized in some domain, ensuring minimum recall and precision values provided by these catalogs in the context of concern identification and classification. This idea would be an interesting way of reducing the cost/effort in using ObasCId approach.

We also believe that performing new experimental studies on the ObasCId(-Tool) approach, comparing it with other AORE approaches/tools, is needed. Other types of experiments that could be performed on the ObasCId-Tool are (i) performance and scalability tests, using large-scale software requirements documents; (ii) tests to verify the most appropriated threshold values for similarity searches; and (iii) tests to verify the effectiveness of the ObasCId-Tool in terms of recall and precision, when its results are confronted to the manual usage of the ObasCId approach, considering documents of requirements of different sizes, among others.

Appendix

Replication of the quasi-experimental study I (see "Quasi-experimental study I" section)

The goal of this replication is the same as that of quasi-experimental study I, presented in the "Quasi-experimental study I" section of this paper: 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 undergraduates and graduates in Computer Science.

Aiming to achieve this goal, a group of 24 participants was asked to identify and classify the concerns of a system of software, using as support the ObasCId and Theme/Doc approaches [22, 23]. It is important to highlight that all resources used in the quasi-experimental studies presented in this paper are available at Parreira Júnior [24]—an English version of these resources may be found in https://goo.gl/cs13QZ.

A.1 Planning of the replication

a) Context selection. This replication study was conducted with 24 undergraduate and graduate students in

Computer Science from a federal university from Brazil. The requirements document of ObasCId-Tool [24] was used in this study—an English version of the requirements is available at https://goo.gl/cs13QZ. It was chosen because the developer of ObasCId-Tool (first author of this paper) was able to generate an oracle with the existing concerns in this software and a catalog for concern identification and classification that allows the use of ObasCId approach.

b) Hypotheses formulation. The same six hypotheses used in the quasi-experimental study I (Table 6) were considered in this replication.

c) Variable and participant selection. Independent variables are those manipulated and controlled during the quasi-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.

d) Design and execution of the replication study. In the quasi-experimental study I, we concluded that performance of ObasCId approach was higher than that of Theme/Doc, regarding recall metric, independent of the group of participants. Hence, we decided to keep the distribution of the participants into two groups of 12 participants. The replication study was planned in just one phase: the participants should identify the functional and non-functional concerns presented in the requirements document of the ObasCId-Tool and classify them as crosscutting or non-crosscutting. To do this, group 1 used the Theme/Doc approach and group 2, the ObasCId.

The ObasCId-Tool requirements document analyzed by the participants had four types of non-functional crosscutting concerns: "Security," "Persistence," "Usability," and "Responsivity" and five functional concerns: "Management of concern catalogs," "Management of researchers," "Management of requirement documents," "Repositories query," and "Concern identification and classification"; two of these five concerns were base ones "Repositories query" and "Concern identification and classification". 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.

A.2 Results and discussion

Table 12 presents the results obtained by both groups of participants, regarding the ObasCId-Tool software. It is possible to notice that ObasCId approach had, on mean, more promising results than those who used Theme/Doc, in terms of recall; however, there is no relevant difference between the two approaches, regarding the precision and execution time. These results are similar to those obtained in the quasi-experimental study I.

Regarding the hypothesis tests, the H_{0Re} null hypothesis may be rejected (Table 6) with significance level of 99.9% (p value = 0.0016). This situation also happened for the HealthWatcher and LocaDVD systems of software. However, regarding execution time and precision metrics, it was not possible to obtain statistical evidences, with significance level equal or higher than 95%, to state that their values are different. The expressive increasing of the execution time between the quasi-experiment and the replication studies (approx. 40 min) may be explained by the size of the requirements

Table 12 Replication study results

Theme/Doc (Group 1)				ObasCId (Group 2)			
Partic.	Recall (Re)	Precision (Pr)	Time (min)	Partic.	Recall (Re)	Precision (Pr)	Time (min)
P01	44.44	89.00	80.00	P13	66.67	89.00	90.00
P02	33.33	67.00	95.00	P14	66.67	89.00	95.00
P03	55.56	100.00	85.00	P15	77.79	89.00	100.00
P04	22.22	89.00	80.00	P16	88.89	100.00	80.00
P05	11.11	78.00	70.00	P17	66.67	78.00	85.00
P06	66.67	89.00	75.00	P18	66.67	78.00	70.00
P07	55.56	89.00	88.00	P19	55.56	56.00	92.00
P08	77.78	56.00	87.00	P20	100.00	89.00	87.00
P09	22.22	100.00	88.00	P21	66.67	89.00	88.00
P10	22.22	78.00	80.00	P22	66.67	100.00	93.00
P11	11.11	78.00	77.00	P23	55.56	67.00	97.00
P12	33.33	67.00	78.00	P24	100.00	67.00	100.00
Avg.	37.96	81.48	81.92	Avg.	71.38	82.41	89.75

document used. ObasCId-Tool contains 30 requirements while HealthWatcher and LocaDVD have 10 and 8, respectively.

A.3 Threats to validity

The threats to validity of this replication study are similar to those ones presented in the quasi-experimental study I and will not be discussed again.

Abbreviations

AORE: Aspect-oriented requirements engineering; FR: Functional requirements; GQM: Goal, question, metric; i18n: Internationalization; JSF: JavaServer Faces; NFR: Non-functional requirements; O4C: Ontology for Concerns; ObasCId: Ontologically based Concern Identification and Classification; Pr: Precision; Re: Recall; RE: Requirements engineering; SABiO: Systematic Approach for Building Ontologies; TAM: Technology acceptance model

Funding

The authors declare that they have no funding.

Availability of data and materials

The authors declare that all data and materials are available at Parreira Júnior [24].

Authors' contributions

PAPJ This paper aims to present the products generated from the doctorate degree of the author. RADP She was the supervisor of PAPJ in his doctorate degree. Her contribution is mainly based on the academic orientation and as a partner for making decisions during the doctorate degree. Both authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Author details

[1]Department of Computer Science, Federal University of Lavras, Lavras, MG, Brazil. [2]Department of Computer, Federal University of São Carlos, São Carlos, SP, Brazil.

Received: 27 December 2016 Accepted: 14 December 2017
Published online: 11 January 2018

References

1. Chitchyan R et al (2005) A report synthesizing state-of-the-art in aspect-oriented requirements engineering, architectures and design. Technical report. Lancaster University, Lancaster, p 259 2005
2. Rashid A, Moreira A, Araújo J (2003) Modularisation and composition of aspectual requirements. In: 2nd AOSD. New York, USA, pp 11–20 2003
3. Soeiro E, Brito IS, Moreira A (2006) An XML-based language for specification and composition of aspectual concerns. In: 8th international conference on Enterprise information systems. Paphos, Cyprus, pp 410–419 2006
4. Chitchyan R, Sampaio A, Rashid A, Rayson PA (2006) Tool suite for aspect-oriented requirements engineering. In: International workshop on early-aspects at ICSE, New York, pp 19–26 2006
5. Herrera J et al (2012) Revealing crosscutting concerns in textual requirements documents: an exploratory study with industry systems. In: 26th Brazilian symposium on software engineering (SBES). Natal, RN, pp 111–120 2012
6. Parreira Júnior PA, Penteado RAD (2015) Crosscutting concerns identification supported by ontologies: a preliminary study. LBIP. Springer International Publishing, Switzerland, pp 385–407 2015a
7. Parreira Júnior PA, Penteado RAD (2015) OnTheme/Doc: an ontology-based approach for crosscutting concern identification from software requirements. In: XVII ICEIS. Barcelona, Spain, pp 188–200 2015b
8. Sampaio A, Greenwood P, Garcia AF (2007) Rashid, A. A comparative study of aspect-oriented requirements engineering approaches. In: I international symposium on empirical SE and measurement. Madrid, Spain, pp. 166-175, 2007
9. Parreira Júnior PA, Penteado RAD (2015) An overview on aspect-oriented requirements engineering area. LBIP. 16ed.: Springer International Publishing, Switzerland, 2015c, v. 227, p. 244-264
10. Parreira Júnior PA, Penteado RAD (2014) Aspect-oriented requirements engineering: a systematic mapping. In: XVI ICEIS. Lisboa, Portugal, pp 83–95 2014
11. Parreira Júnior PA, Penteado RAD (2016) ObasCId: an ontologically-based approach for concern identification and classification. In: X Brazilian symposium on software components, architectures, and reuse (SBCARS)„ Maringá/PR, pp 141–150 2016
12. Agostinho S et al (2008) Metadata-driven approach for aspect-oriented requirements analysis. In: 10th international conference on enterprise information systems, Barcelona, pp 1–6 2008
13. Alencar F et al (2010) Towards modular i* models, ACM symposium on applied computing, pp 292–297 2010
14. Chernak Y (2012) Requirements composition table explained. In: 20th requirements engineering conference, Chicago, pp 273–278 2012
15. Zheng X, Liu X, Liu S (2010) Use case and non-functional scenario template-based approach to identify aspects. 2nd Int. Conf. on Computer Eng. and Applications, Bali Island, pp 89–93 2010
16. Brito I, Moreira A (2003) Towards a composition process for aspect-oriented requirements. In: Early-aspect workshop at AOSD, Boston, pp 113–119 2003
17. Moreira A, Rashid A, Araújo J (2005) Multi-dimensional separation of concerns in requirements engineering. In: 13th requirements engineering. Paris, pp. 285-296, 2005
18. Whittle J, Araújo J (2004) Scenario modeling with aspects. IEEE Softw 151(4): 157–172 2004
19. Boehm B, In H (1996) Identifying quality-requirement conflicts. IEEE Softw 13(2):1996
20. Chung L, Leite JSP (2000) Non-functional requirements in software engineering. Springer, p 441 2000
21. Cysneiro LM (2016) Catalogues on non-functional requirements. Available at: http://www.math.yorku.ca/~cysneiro/nfrs/nfrs.htm. Last access: Dec. 2016
22. Baniassad E, Clarke S (2004) Theme: an approach for aspect-oriented analysis and design. In: 26th international conference on software engineering, Washington, pp 158–167 2004
23. Clarke S, Baniassad E (2005) Aspect-oriented analysis and design: the theme approach, Addison-Wesley, p 400 2005
24. Parreira Júnior PA (2015) ObasCId: uma Abordagem Ontologicamente Fundamentada para EROA. Doctorate Thesis. 2015. UFSCar/São Carlos/SP, p 197 (in Portuguese)
25. WMATRIX (2016) Corpus analysis and comparison tool. Lancaster University. Available at: http://ucrel.lancs.ac.uk/wmatrix/. Last access: Dec. 2016
26. Parreira Júnior PA, Penteado RAD (2015) Domain ontologies in the context of requirements engineering: a systematic mapping. In: XII ACS/IEEE. Marrakech, Morocco, pp 1–8 2015d
27. López C, Cysneiro LM, Astudillo H (2008) NDR ontology: sharing and reusing NFR and design rationale knowledge. In: International workshop on managing requirements knowledge. USA, pp. 1-10, 2008
28. Falbo RA (2016) SABiO: systematic approach for building ontologies. 2011. Available at: http://www.inf.ufes.br/~falbo/files/SABiO.pdf. Last access: Dec. 2016
29. Guizzardi G (2005) Ontological foundations for structural conceptual models. PhD. thesis. Univeristy of Twente, 2005
30. Greenwood P et al (2007) On the impact of aspectual decompositions on design stability: an empirical study. In: European Conference on Object-Oriented Programming (ECOOP), Berlin, Germany, pp 176–200 2007
31. Greenwood P et al (2007) On the contributions of an end-to-end AOSD testbed. In: Workshop aspect-oriented requirements engineering and architecture design (early-aspects), Minneapolis, MN, USA, pp 1–8 2007
32. Health Watcher (2016) Available at: http://www.cin.ufpe.br/~scbs/testbed/requirements /aore/. Last access: Dec. 2016
33. Braga RTV, Germano FSR, Masiero PC (1999) A pattern language for business resource management. In: 6th PLoP, pp 2–7 1999
34. Ranks NL. Available at: http://www.ranks.nl/stopwords. Last access: Dec. 2016
35. Manning CD, Raghanvan P, Schütze H (2008) Introduction to information retrieval, Cambridge University Press. 482 p., 2008

36. Basili V, Rombach H (1994) Goal question metric paradigm. In: Encyclopedia of software engineering, p 2, 6 1994
37. Wohlin C, Runeson P, Höst M, Regnell B, Wesslén A (2012) Experiment. in SE. Springer-Verlag Berlin Heidelberg. 236p. 2012
38. Viana MC (2009) Building the graphical user interface layer and a wizard for the GRENJ framework. Master dissertation, UFSCar, São Carlos 2009 (in Portuguese)
39. Montgomery DC (2000) Design and analysis of experiments, 5ª ed. Wiley, New York, p 752 2000
40. Davis FD (1993) User acceptance of information technology: system characteristics, user preceptions and behavioral impacts. Int J Man Mach Stud 38:475–487
41. Hernandes ECM (2014) Abordagem Orientada à Informação para Análise Qualitativa com suporte de Visualização e Mineração de Texto. Doctorate Thesis, UFSCar, São Carlos, 2014 (in Portuguese)