Requirements Eng (2015) 20:139–161
DOI 10.1007/s00766-013-0193-4

ORIGINAL ARTICLE

Early identification of crosscutting concerns with the Language Extended Lexicon

Leandro Antonelli · Gustavo Rossi · Julio Cesar Sampaio do Prado Leite · João Araújo

Received: 28 February 2013 / Accepted: 7 November 2013 / Published online: 26 November 2013

Abstract Large-scale software applications are complex systems that involve a myriad of different concerns. Ideally, these concerns should be organized into separated and different modules, but often some of these concerns overlap and crosscut each other. Such a situation is problematic, as concerns are tangled and scattered into different modules; thus, design and source code become difficult to produce and maintain. The Modularity community has been addressing crosscutting concerns by developing techniques based on separation of concerns. This separation must be done as early as possible during software construction to obtain a more modular and consequently better maintainable software, where evolution is performed with less effort and the possibility of introducing unforeseen mistakes is minimal. In this paper, we propose a strategy to identify crosscutting concerns at requirements level, i.e., at early stages in the software development process, by using the Language Extended Lexicon.

Keywords Requirements engineering · Modularity · Crosscutting concerns · Language Extended Lexicon

L. Antonelli (✉) · G. Rossi
Lifia, Fac. de Informática, UNLP, Buenos Aires, Argentina
e-mail: lanto@lifia.info.unlp.edu.ar

G. Rossi
e-mail: gustavo@lifia.info.unlp.edu.ar

J. C. S. P. Leite
Dep. Informática, PUC-Rio, Rio de Janeiro, RJ, Brazil
URL: http://www.inf.puc-rio.br/∼julio

J. Araújo
CITI, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisbon, Portugal
e-mail: joao.araujo@fct.unl.pt

1 Introduction

The development of software systems is a complex activity since different sorts of actors with different backgrounds are involved. During development, they perform distinct tasks at different moments with the objective of building diverse software products. This means that it is difficult to define a precise schedule to organize the development at the beginning of the project. Initially, there is not much detailed information; the result is a coarse-grained schedule, and its deadlines become a goal to accomplish instead of a reasonable estimation for the delivery of the product. As a consequence, overcoming a delay becomes difficult. In other engineering disciplines such as civil engineering, it is usual for some tasks to be repetitive and increasing the number of people can reduce the overall schedule. Software engineering is different as, in general, tasks are very specific, and if we add more people to overcome the delay, we often lose more time (at least at the beginning when new people are incorporated). This fact is known as Brooks' Law [12]. Furthermore, the nature of the software also makes its development a complex task. To illustrate this, we can list four well-known software characteristics (adapted from [12]) that make software a different kind of artifact altogether, contributing to the complexity of its construction.

(i) Invisibility: Software is not visible as a product, like a building, a car, or a plane. This characteristic imposes a barrier for human perception of size, form, and structure.

(ii) Modifiability/changeability: Software is built by descriptions, which are very malleable since change can be made abstractly by rewriting. This creates the illusion that software is easy to change.

(iii) Conformity: Software has to conform to its infrastructure, which can also be software. It does not exist per se. Changes in the infrastructure imply changes in the software.
(iv) Complexity: Software needs reification by other artifacts, including software, to transform itself from an idea into a product. This characteristic corroborates how difficult it is for humans to have a clear idea of its limits, interaction with the environment, and its shape.

As such, software has to deal with a lot of concerns. In order to deal with complexity, the traditional engineering strategy is to use decomposition, that is, the "divide and conquer" strategy: dividing a problem into smaller pieces that can be solved separately. After the solution of each piece, all the partial solutions must be joined together to obtain the solution to the entire original problem. This strategy is used in software system development in two of the most widespread paradigms: structured design and object-oriented design. In structured design, the entire system is decomposed into a hierarchy of modules where each module solves a particular part of the problem. The integration of all modules comprises the target system. In object-oriented design, a proposed solution consists of a hierarchy of classes arranged by the "is specialization of" relation. Then, each class must have a set of well-defined responsibilities, and the integration of all the classes into one single environment will produce the targeted system [29]. Both techniques rely on encapsulating each concern in one building block: a module or a class. However, this strategy fails when there is a need to deal with pieces that are spread over other pieces, because they are common to several of them. Usually, this happens when injecting quality attributes into a product, because the operationalization of a quality attribute is similar for different pieces. A recurring example is authorization: Different pieces of a software system may require operations that guarantee that the proper authorization is granted for the execution of that piece. As such, authorization is an example of a crosscutting concern, that is, it crosscuts different pieces of a whole software application. Crosscutting concerns are characterized by being tangled and scattered. They are tangled to other concerns into a piece and scattered into different pieces. The Modularity community [also known as the aspect-oriented software development (AOSD) community] deals with crosscutting concerns, also known as aspects, in order to tackle the problems of tangling and scattering.

Software development is a succession of descriptions in different languages where a previous description is necessary for the next. If changes are incorporated into a description, they spread over previous and succeeding descriptions, which have to be changed in order to maintain conformity. For instance, Boehm [9] states that if a mistake occurs in a requirements description and it is only corrected in code description, after implementation, the correction cost could be multiplied by up to 200. Moreover, Mizuno [22] developed the "waterfall of errors" in which he states that in every stage of software development, the possibility of occurrences of mistakes is stronger than in the previous one, because every stage relies on prior artifacts. Even if wrong products may produce good work, it would still be based on wrong products, so it would, in fact, be wrong work after all, that is, why it is necessary to make the right decision as soon as possible during software development. This means that it is extremely important to identify crosscutting concerns early on in order to avoid rework, i.e., reduce the evolution effort.

Yu et al. [33] focused on discovering aspects from goals. Goal-based models support the description and analysis of intentions that underlie a new software system. Some goal models such as i* also model the actors that hold these intentions. These goals are "roughly speaking, precursors of requirements." Nuseibeh argued that the problem world, inhabited by customers and users, is a fertile ground for identifying concerns. Indeed, he states that the problem world is often the most appropriate source for early identification of concerns [23]. Rashid and Moreira [27] also argue that it is possible to identify crosscutting concerns during domain analysis, a stage that is carried out before requirements are written.

There are several approaches to identify crosscutting concerns such as the ones depicted in [7, 10, 25, 26, 28, 30], but by analyzing them in detail (see Sect. 2), we detect some shortcomings that can compromise the effectiveness of the aspect identification itself in an application domain. To mitigate these shortcomings, we propose the Language Extended Lexicon (LEL) approach. LEL is a vocabulary oriented technique to describe an application [18] and identify crosscutting concerns. It can be defined as a glossary whose aim is to understand the language of the application [18]. This glossary must be built very early in software development stages because the LEL construction makes it possible to learn about the application domain and allows requirements engineers to write requirements with the knowledge gained from the LEL construction. Although there are techniques to derive requirements from LEL [2], its main goal is to understand the language of the application domain. The construction of a LEL requires an elicitation strategy at a meta-level, that is, the elicitation of the application language, the language that is spoken in the universe of discourse in which a future software or a future evolution of a software will be placed. This metalanguage is represented by the LEL, a lexicon where there is a

definition for the connotation and the denotation of phrases and words from the universe of discourse.

The approach we propose identifies crosscutting concerns very early in software development, even prior to a requirements specification. It is very important to identify crosscutting concerns as early as possible because the development team can use the strategic information of crosscutting concerns in order to build the products they need (requirements specification, architectural design, source code, etc.). Moreover, performing the identification so early avoids the reworking of artifacts and, otherwise, necessary if crosscutting concerns were identified in the middle of the development. Another important consideration is that LEL can be constructed by practitioners and can be reused to identify crosscutting concerns with the strategy we propose. The language we use to represent the application (LEL) has a good expressiveness, but above all, it uses conversational language without introducing any kind of formalism which would make difficult its use by application experts.

In summary, the aim of this paper is to provide a strategy to identify crosscutting concerns using LEL. The two main contributions of this work are as follows:

(i) First, the strategy is based on regular LEL construction with only one additional issue: The requirements engineer must group symbols into states. Our strategy provides guidelines to organize LEL, making groups of symbols according to states.
(ii) Second, the approach identifies crosscutting concerns by performing some calculations of the relationships between symbols belonging to different groups. This calculation is the main contribution to our strategy. It can be performed manually assisted by a spreadsheet, although there is a specific tool being implemented [4]. This calculation provides information to help the requirements engineer make the final decision about what are the identified crosscutting concerns.

The rest of the paper is structured as follows. Section 2 discusses some related work. Section 3 provides the background necessary to understand our approach. Section 4 describes the approach, while Sect. 5 applies it to two case studies. Finally, Sect. 6 presents some conclusions and suggestions on future works.

2 Related work

This section describes approaches whose aim is identifying crosscutting concerns at early stages, as this is also one of the goals of this paper. At the end of this section, we also mention some approaches related to identifying aspects in source code.

Baniassad et al. [7] describe a general approach to identify crosscutting concerns. Baniassad and Clarke [6] present Theme/Doc, a systematic approach to model aspectual requirements. Sampaio et al. [28, 29] propose a tool to mine aspect in requirements. The tool is configurable, and it can work with viewpoints, use cases, etc., starting its analysis on the colloquial language from documents. Rago et al. [25] considers important to use not only verbs but also objects on which verbs act. Rashid et al. [26] propose a model with viewpoints with group requirements. It is very interesting to group requirements instead of working inside them. The strategy consists in identifying viewpoints which impact on many requirements. Shepherd et al. [30] detect crosscutting concerns analyzing naming conventions. The approach is based on making lists of words that have a related meaning or denotation. Bounour et al. [10] present a categorization of techniques to identify crosscutting concerns mainly in source code. Some techniques rely on execution flow or dataflow, and others rely on structure or dispersion of the source code. There are two lines of work that focus on composition instead of identification. Chitchyan et al. [14] use the natural language defined in a meta-model. Araújo et al. [5] use state machines to make compositions of crosscutting concerns. These approaches are described in more detail, as follows.

Baniassad et al. [7] propose a way of organizing requirements by dividing them into core and crosscutting ones, and they also describe some activities to perform this organization. Their work performs an analogy between requirements documents and source code, and they suggest organizing requirements in the same way, specifying core requirements apart from crosscutting requirements, both of which must be related through impact requirements. They also propose an approach to specify requirements based on 4 activities: (i) identification, (ii) capture, (iii) composition, and (iv) analysis. Identification means analyzing the requirements in order to separate core requirements from crosscutting ones. This separation can be done through three techniques: (i) aspect terms, (ii) impact requirements, and (iii) scattered concerns. Aspect terms are terms or expressions that refer to quality attributes (nonfunctional requirements such as safety, security, and performance). It is easy to implement this kind of strategy because it consists in a data mining approach which compares the extracted words with a database of candidate terms. Identification through impact requirements involves analyzing the relationship between the requirements. One requirement is related to another when there is an explicit mention of the second requirement in order to qualify it in some way. The scattered concerns technique relies on identifying scattered knowledge that is repeated in several requirements. This strategy is analogous to the identification of aspects in source code.

After identification through one of the strategies described in the previous paragraph, capture must be performed. It consists in reorganizing requirements by isolating crosscutting requirements from core requirements in order to avoid redundancy of information. After capture, requirements are modeled into two groups; thus, it is necessary to specify the relationship between them, that is, the goal of composition. Basically, each aspectual requirement must be related to a core requirement. After composition, a new model is obtained, which contains the same information as the original model. Nevertheless, the new model is modularized and cleared of repeated information. The last activity, analysis, involves detecting conflict and inconsistencies.

Baniassad and Clarke [6] propose a technique called Theme/Doc to model requirements showing the relationships between different elements. Theme/Doc is based on the notion of theme which represents a feature of the system. Multiple features can be combined in order to describe the whole functionality of the system. There are two types of themes: base themes and crosscutting themes; the latter have behavior overlapping with functionality from the base themes. Theme/Doc works on the premise that if two behaviors are described in the same requirement, they are related. The strategy of identification consists of four steps: (i) identification of actions and entities, (ii) categorization of actions into themes, (iii) identification of crosscutting themes, and (iv) analysis of themes. The first step, identification of actions and entities, involves analyzing the requirements of the system and identifying key actions and key entities. The second step, categorization of actions into themes, consists in determining which of the key actions must be considered themes. Some actions are not sufficiently relevant to be considered themes since they are subactions. In this step, the action view must be built, a diagram composed of two elements: actions and requirements. Each action must be related to the requirements where it is mentioned. This diagram helps to define themes, but it is a highly intuitive process. The third step, identification of crosscutting themes, is performed with a diagram similar to the action view, but only containing the actions considered as themes. This diagram shows requirements that are shared by two or more themes. If a requirement is shared by two or more themes, an aspect may have been found, because each theme needs the other to provide the functionality. Nevertheless, it could also be possible to have found a requirement that includes several other requirements. Thus, it is necessary to verify whether the requirement can be written into several requirements where each one relates to only one theme. We will have to found an aspect if that rewriting is not possible. The last step involves analyzing each theme in relation to the requirement associated with key actions. This analysis is used to confirm the identification.

Sampaio et al. [28, 29] propose a tool, EA-Miner, to identify and isolate base concerns, early aspects, and crosscutting relationships between these elements. Since the tool is configurable, base concerns depend on the model used. For example, if it is UML, a base concern is a use case, while if it is a model oriented to viewpoints, a base concern is a viewpoint. Early aspects are semantic entities which crosscut base concerns. The authors emphasize that the strategy uses documents without any specific type of structure. EA-Miner uses some natural language processing techniques: parts of speech (POS), semantic tagging and word frequencies, and concordances. All of this is provided by WMatrix tool. The authors argue that the best way to analyze the great amount of information produced in the requirements step is through a tool. Nevertheless, they also recognize the importance of an expert to validate the results provided by the tool. The strategy consists of the following steps. The first one is identification, where EA-Miner parses the different structure files and sends them to WMatrix in order to perform the POS and semantic tagging. EA-Miner analyzes results and produces a new model over which the screening out activity is performed. Finally, the results are shown to the user who must refine them by deleting or adding entities. The most sensitive activity is the process that WMatrix performs where it identifies nouns and verbs in POS. Then, the semantic tag that relates words and expressions to concepts is also sensitive. The weakness in this tool is that it does not detect small variations in words (for example, singular to plural). Moreover, some of the identifications are performed based on a dictionary; thus, if the dictionary is not complete, the results are not either.

Rago et al. [25] state that the majority of the techniques to identify crosscutting concerns rely on searching for verbs, but they consider it more useful and accurate to identify the objects on which verbs act. However, since natural language not always includes information about the objects affected by verbs, it is also necessary to consider verbs without objects. The strategy consists of the following steps. First, a lexical, syntactic, and semantic analyses are performed. Basically, it involves discourse tagging and semantic disambiguation of the words. This is performed on the basic and alternative flow of the use cases, and afterward on nonfunctional requirements and additional requirements. All this information makes possible the production of a graph with nodes representing the verbs and direct objects which are referred to by other verbs and direct objects. This relationship is achieved through intermediate nodes which group semantically related verbs and nouns. The graph is traversed in order to determine aspect candidates, sorting them by feasibility. Candidates are groups of verbs which crosscut some use cases, the total number of which must be greater than a

certain value. Then, the verbs in these groups are analyzed to identify the most representative verb of all. Additionally, it is decided whether the concerns are functional or non-functional. In order to determine the ranking of each crosscutting concern, the following variables are analyzed: crosscutting use cases, the number of crosscuts produced by the group of verbs, relevancy of the occurrence of the group of verbs, relevancy of the occurrence of the direct objects, and characteristics of the functional or nonfunctional requirements.

Rashid et al. [26] propose an approach to modularize core functionality (modeled through viewpoints grouped in requirements) and crosscutting concerns, and show how to compose them. The strategy begins by identifying and specifying the stakeholders' concerns and requirements. It is useful to relate concerns to requirements through a matrix. Then, observing the matrix, it is possible to identify which concerns crosscut which requirements. This is considered a coarse-grained identification. After that, it is necessary to define composition rules between concerns and requirements in order to establish fine-grained relations between them. These rules work at requirements level, and they require actions and operators, which can include the following: constraints, outcome, enforce, ensure, provide, and apply. If two aspects are restricted to the same requirement, a balance must be performed between the aspects considering the contribution.

Shepherd et al. [30] detects crosscutting concerns analyzing naming conventions. He claims that the lexical chain is a good technique because it overcomes the problem of syntactic analysis. The approach consists in making lists of words with a related meaning or denotation. Then, giving to the word "related" the meaning of certain semantic distance, the approach identifies crosscutting concerns by finding groups of related words.

Bounour et al. [10] present a classification of aspect mining approaches. Some techniques are based on analyzing static source code, while others rely on execution flow. Bounour identifies a clone detection category where approaches conduct a lexical analysis of the information structure. There is an approach which uses program dependence graph (PDG) containing information of semantic nature, such as control and dataflow of a program. Bounour also mentions a technique which combines an abstract syntax tree (AST) with tokenization representations of source code. This technique uses parsers to obtain a syntactical representation of the source code (AST). Then, the clone detection algorithm searches for similar subtrees in the AST. The AST technique has its limitations; in that, only homogeneous concerns can be identified, because the approach relies on detecting similar patterns in the tree and minor differences from subparts of the tree making it impossible to detect an aspect. Bounour also states that there is a formal concept analysis approach (FCA) based on finding meaningful groupings of elements that have common properties. In order to achieve this grouping, several use cases are executed and two constraints must be analyzed to find concept candidates. The first constraint states that the attributes of the concept must belong to more than one class. The second states that different methods from the same class are contained in more than a use case. Finally, Bounour also describes an approach based on scattering called fan-in analysis, which mines source code to find symptoms of code scattering. This technique states that a method with a lot of distributed calls must be considered a concern because the method implements a crosscutting functionality. The technique considers that the amount of calls (fan-in) is a good measure for the importance and scattering of the discovered concerns. We do a kind of fan-in analysis in our approach as well because we measure references of behavioral responses between symbols from different sates.

There are two lines of work that focus on composition instead of identification. Chitchyan et al. [14] claim that syntactic composition has its limitations, so they use natural language. They define a meta-model with subjects and objects. Araújo et al. [5] present an approach to modeling scenario-based requirements using aspect-oriented principles. Aspectual scenarios were modeled using pattern specifications, and a technique is described to compose aspectual and non-aspectual scenarios and to transform them into a set of executable state machines. In this way, they show how to separate aspects during scenario development but also how to generate a composed behavioral description for simulating the scenarios.

This section has described several techniques to identify crosscutting concerns at early stages of software development. Some techniques are based on requirements statements and use cases, while others analyze products from previous stages. In general, all the techniques are based on analyzing verbs in order to identify crosscutting concerns. However, other techniques also rely on objects. Moreover, it is important to group elements in order to identify how they relate to each other. We will now summarize the benefits and drawbacks of the different techniques used to identify crosscutting concerns at early stages.

Baniassad et al. [7] propose a group of activities to factorize requirements, but no specific strategy is suggested for this. Baniassad and Clarke [6] do propose a specific strategy to identify crosscutting concerns, although the strategy depends on the subjectivity of the practitioner, who must decide which actions are themes. The practitioner must also provide some keywords to begin the analysis. Sampaio et al. [28, 29] devise a strategy that requires much human support, because the proposed tool provides results, but the practitioner must analyze that

information so that the tool can continue the process. Rago et al. [25] consider the analysis of verbs and objects, which is important to enrich the process, but they perform the analysis after requirements are specified because they work with use cases. Rashid et al. [26] proposes an interesting and uncommon technique for grouping requirements into viewpoints; nevertheless, the strategy is subjective. Also, all these techniques were not conceived to describe domain properties. It is worth mentioning that our approach does not aim to propose a composition language as some of the other approaches do, but their mechanisms can be adapted and incorporated in our approach as a part of future work.

In this paper, a novel strategy is proposed to identify crosscutting concerns that solves some of the previously discussed problems, namely: (i) diminish subjectivity, (ii) perform the analysis as early as possible, (iii) consider objects and not only verbs, and (iv) grouping the artifacts used to perform the analysis. Our approach consists in analyzing LEL, an artifact constructed very early in software development, even previous to requirements specification. The LEL symbols are categorized, and we rely on verbs as well as objects. Symbols are grouped into states, and the strategy involves performing some objective calculations between the references of LEL symbols from different groups in order to rank the crosscutting concerns probability. This ranking helps the requirements engineer to make his final decision as to what are the concerns.

3 Language Extended Lexicon (LEL)

This section describes the LEL. This technique captures the language related to an existing or future application. LEL stands as an important source of raw material to write requirements, which is the basic element we analyze to identify crosscutting concerns.

We use a vocabulary oriented strategy to capture the application language, that is, the words and expressions used by different actors in the context of the application which have a particular meaning in such a context. LEL takes a simple idea as a starting point, describing the *language* of the application domain before describing the application. As such, it is a metalanguage used to gather or elicit requirements, which aims at describing the meaning of words and phrases specific to a given application domain. LEL can be understood as a glossary whose goal is to capture the definitions of relevant terms belonging to an application for which requirements will be written. LEL is anchored in a simple idea: "understand the language of a problem or a domain, without worrying about understanding the problem." Thus, LEL construction relies on the identification of symbols which are important to the application and must be defined in order to understand the language of the application domain. The immediate potential advantage of this is reusability of specifications in a domain where we can develop a family of applications, but this issue will not be addressed in this paper.

This section presents some basic concepts of LEL construction. We will not explain in detail the process of constructing LEL because its construction is prior to our approach; the reader who is interested in this can refer to Leite and Franco [18] and Breitman and Leite [11], who describe the process to obtain LEL. Moreover, there is a tool named C&L to assist in this process [13, 19].

Terms (symbols) are defined through two attributes: notion and behavioral responses. Notion describes the symbol denotation, which are the intrinsic and substantial characteristics of the symbol. Behavioral responses describe connotation, i.e., the relationship between the term being described and others. It is our assumption that behavioral responses have a conceptual relationship with the identification of *impact requirements*.

Each symbol of the LEL belongs to one of four categories: subject, object, verb, and state. This categorization guides and assists the requirements engineer during the description of attributes. Table 1 shows each category with its characteristics and how to describe them.

We will use, as an example, the same banking application we have mentioned in the previous section. A bank allows its clients to save money. The bank opens accounts for its clients to deposit and withdraw money. The bank also allows them to consult their account balance. As the bank needs some kind of security, it must verify the identity of the person who operates the account, and it also keeps a record of each operation in a log after it is performed. Finally, the process of opening and using an

Table 1 LEL categories

Category	Characteristics	Notion	Behavioral responses
Subject	Active elements which perform actions	Characteristics or condition that subject satisfies	Actions that subject performs
Object	Passive elements on which subjects perform actions	Characteristics or attributes that object has	Actions that are performed on object
Verb	Actions that subjects perform on objects	Goal that verb pursues	Steps needed to complete the action
State	Situations in which subjects and objects can be	Situation represented	Actions that must be performed to change into another state

Table 2 Symbols of the banking application grouped by category

Category	Symbols
Subject	Bank
	Client
Object	Account
	Log
Verb	Open an account
	Activate
	Record
	Verify
	Operate
	Deposit
	Withdraw
	Consult balance
State	Signed
	Blocked
	Activated

account follows 3 states. First of all, the bank and the client agree to open an account. After that, the bank opens the account, but it will be blocked because the bank must first verify some documentation from the client, and although the client will have his account, he will not be able to operate it. Finally, the bank must activate the account in order to allow the client to deposit, withdraw, or consult the balance. Table 2 shows symbols for this example. Symbols are grouped by category.

The following example provides the description of one symbol to show how the descriptions of notion and behavioral responses conform to the template defined. Descriptions of symbols have underlined words; these words are expressions that are defined in LEL too. They represent a kind of link which can be navigated to explore the definition of the other word. The symbol is *client*, and as it is a subject, its *notion* describes who a *client* is, while the behavioral responses describe the actions the *client* can perform on his *account* (Fig. 1).

4 Identification of crosscutting concerns within LEL

Our approach relies on the LEL in order to identify crosscutting concerns. The LEL must be constructed as usual; once it is built, the requirements engineer must make groups of symbols according to the state identified in LEL. Then, the references from the behavioral responses must be counted. After that, the strategy ranks groups ordering by probability, from the most probable to the least probable crosscutting concerns candidates. It is important to emphasize that the strategy only ranks the concerns according to their possibility to be considered crosscutting, and the strategy does not determine a limit above which all concerns are crosscutting. Moreover, the approach cannot ensure that the group with the highest rank is effectively a crosscutting concern, because the group might have core symbols which could alter the counting. That is, the requirements engineer must analyze each group in particular in order to determine if the group is composed of crosscutting concerns symbols, which makes the whole group a crosscutting concern. If the group has core symbols which alter the values, then, although the group is ranked high, it must not be considered a crosscutting concern. This distinction between core symbols and crosscutting concerns symbols must be drawn by the requirements engineer.

The detailed steps to the approach are as follows:

(i) Construction of LEL organized into groups. The LEL must be constructed as usual, describing the notion and behavioral responses for every symbol. Then, each symbol must be related to a state.
(ii) Reference counting. The references from each symbol to the behavioral responses of another group's symbols must be counted. Then, the counts for the whole group must be summed up, as the basis of the approach is finding group candidates for crosscutting concerns.
(iii) Ranking of groups. Groups of symbols must be sorted out according to their probability of being crosscutting concerns candidates.
(iv) Final analysis. The most suitable groups to qualify as crosscutting concerns must be identified. The strategy ranks the groups, and the requirements engineer must look into each group to determine if the whole group can be considered a crosscutting concern or if there are core symbols which could have altered the rank of the group.

Figure 2 illustrates this process.

The essence of our approach is based on identifying crosscutting concerns as the symbols most frequently

Fig. 1 *Client* symbol description

Subject: client
Notion
Person that opens an account at the bank
Behavioral responses
The client can deposit money into his account.
The client can withdraw money from his account.
The client can consult balance from his account.

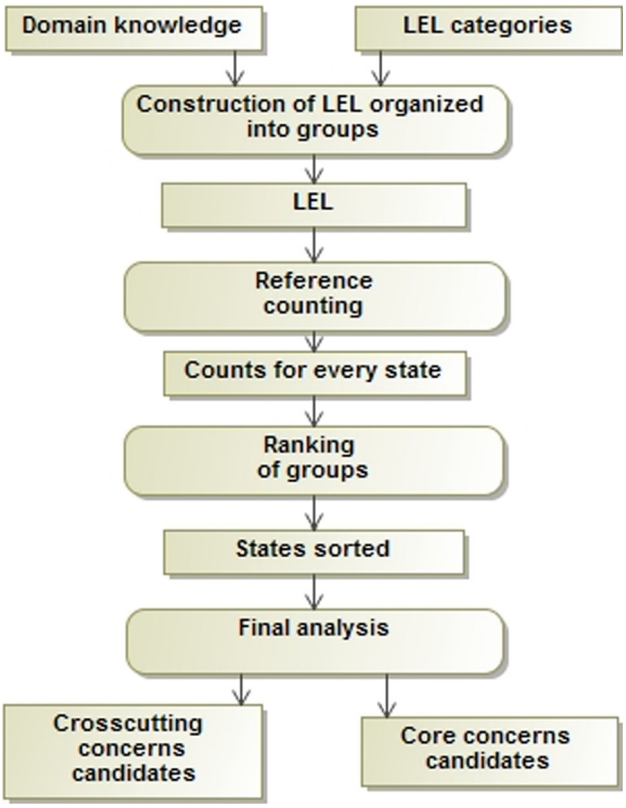

Fig. 2 The approach in a nutshell

referred to by the behavioral responses of other symbols [1]. The approach is complex as it has to deal with a large number of symbols. We will now explain the basic mechanism of the approach, going into detail in the following sections.

Consider the following verb symbols and their behavioral responses. The example does not show a complete definition for every symbol, that is, their notion and behavioral responses, because our approach only makes use of behavioral responses (Fig. 3).

For example, verbs *deposit*, *withdraw*, and *consult* have references to verbs *verify* and *record* in their behavioral responses. These references are marked in dark gray in Table 3. In Fig. 4, a directed graph shows the links between symbols *deposit*, *withdraw*, and *consult* and symbols *record* and *verify*. This means that besides the basic functionality of *deposit*, *withdraw*, and *consult*, each of the verbs must perform the acts of *verifying* and *recording* in order to complete the whole functionality. Since the essence of our approach leads to the identification of the most frequently referred to symbols, and *record* and *verify* are referred to by *deposit*, *withdraw*, and *consult*, then our approach identifies *record* and *verify* as crosscutting concerns. It is important to notice that this is the basis of our approach, but in fact, the approach also uses state symbols and makes groups according to them.

As it has been mentioned previously, the essence of the approach lies in analyzing the references between the behavioral responses of symbols. We propose doing that because the references in the behavioral responses have a connection with invocation of procedures in source code. In fact, the behavioral responses of verb symbols are a kind of module decomposition, but this analysis is not performed at the level of symbols. While constructing LEL we do not have a building block as we do in requirements, design, or code, so we define a group of

Verb: Deposit
Behavioral responses
The bank must verify that the person who operates is the owner of the account.
The bank adds the money to the account.
The bank records the operation in a log.

Verb: Withdraw
Behavioral response
The bank must verify that the person who operates is the owner of the account.
The bank must check that the account has enough money to perform the withdrawal.
The bank must check that the owner of the account has not withdrawn money more times than the limit allows.
The bank records the operation in a log.

Verb: Consult the balance
Behavioral responses
The bank must verify that the person who operates is the owner of the account.
The bank shows the amount.
The bank records the operation in a log.

Verb: Record
Behavioral responses
Bank records operations in the log.

Verb: Verify
Behavioral responses
Bank verifies identity of the client.

Fig. 3 *Deposit*, *withdraw*, *consult the balance*, *record*, and *verify* symbols description

symbols (cluster) determined by states as the building block we will use to analyze references. We have chosen to make groups of symbols with states because the state symbols in LEL model the nodes of a state machine, and Mahoney [20] and Mahoney and Elrad [21] claim that crosscutting behavior is often modeled using state machines. The reason for this lies in that applications are complex reactive systems with state machines describing their behavior. Thus, the application reacts (responds) to a stimulus according to its state. Sometimes, only one state reacts to a stimulus, while some other times, different states react to the same stimulus. That is, there are stimuli which crosscut many (maybe all) of the states. Clearly, the stimuli which crosscut all of the states are crosscutting concerns, because they are concerns for the entire state machine.

As an example of crosscutting concerns modeled as a state machine, consider the access control to a *bank account*. The system asks for a password to gain access to *deposit*, *withdraw*, and *consult balance*. *Accepting password* is a state where an automatic teller machine (ATM) waits for the client to type 4 digits; after that, the system verifies the password entered. If the password is incorrect, the system waits for a new password, but if it is correct, the system displays a menu and waits for the operation the *client* wants to perform: *deposit*, *withdraw*, or *consult the balance*. In addition, in certain states, the session can be canceled. As cancellation can occur in many different states, the implementation needs duplicated code in each state in order to handle cancellation. This duplicated code must in fact be factorized in aspects, as cancellation is a crosscutting concern. We argue that LEL description organized into states will allow for a similar analysis and will identify the same crosscutting concerns. Figure 5 shows the state machine.

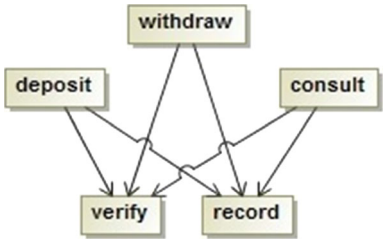

Fig. 4 References to symbols *verify* and *record*

The following sections describe each step to the approach.

4.1 Construction of LEL organized into groups

Our approach demands that LEL symbols be organized into state symbols. It is necessary to make groups of symbols because we need to modularize the knowledge in building blocks in order to determine which building blocks are core functionality and which are crosscutting concerns. As a software system can be reduced to a state machine [20, 21], and as these states are natural to the application, we group symbols according to states. The process of identification and definition of symbols includes the following steps:

(i) The first step is LEL construction. LEL must be built by performing: (a) identification of symbols; (b) categorization of each symbol; (c) description of symbols

Table 3 Adjacency matrix of references between symbols of the banking application

	Bank	Client	Account	Log	Open an account	Activate	Record	Verify	Operate	Deposit	Withdraw	Consult balance	Signed	Blocked	Activated
Bank		1	1	1	1		1	1	1						
Client			3							1	1	1			
Account		3								1	1	1			
Log	2						1		2			1			
Open an account	1														
Activate	1		1												
Record	1			1											
Verify	1	1													
Operate		1	1												
Deposit	3		2	1			1	1	1						
Withdraw	4		3	1			1	1	1						
Consult balance	3		1	1			1	1	1						
Signed	1	1	1		1									1	
Blocked	1	1	2			1			1						1
Activated		1	1						1						

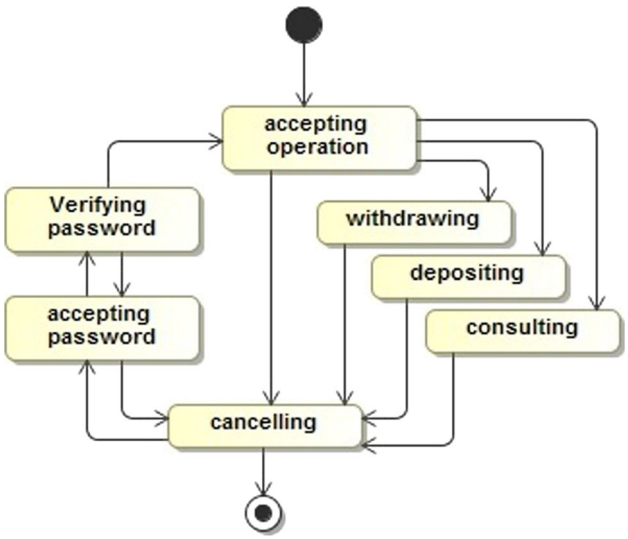

Fig. 5 State machine of the ATM operation

according to the category; and (d) identification of synonyms. In order to identify symbols, if written material is analyzed, the most frequent words must be considered; if oral communication is analyzed, words which are pronounced with stress must be considered. Categorization and description of symbols must be done according to the basic categorization which determines the description of notion and behavioral responses. Although our approach only deals with behavioral responses, it is important to describe notions while describing each symbol. Notions are significant because their description helps to identify synonyms. Two symbols with a similar description of notion are candidates for synonyms. Notions are also important because they may contain words that must be also described in LEL, so they constitute a source of symbols too.

(ii) The second step is to identify the main symbols in the application. A subject or an object which is a principal element in the application language and which has an associated state machine must be considered. In the banking application, the following could be candidates for main symbols: *account*, *client*, *bank*, *deposit*, *withdraw*, and *consult balance*. However, only *account*, *client*, and *bank* are considered because they correspond to subject and object, respectively. Finally, from the previous symbols, only account is considered the main symbol of the application because it has an associated state machine. All the states wherein the object or subject can be must be identified (LEL has a symbol category which is "state"). The symbols for the banking application are *signed*, *blocked*, and *activated*. The associated state machine is very simple. The first state is *signed*, which has a transition to *blocked*. Then, *blocked* has a transition to *activated* which is the final state.

Table 4 Symbols related to each state in the banking application

State	Subject	Object	Verb
Signed	Bank	Account	Open an account
	Client		
Blocked		Log	Activate
			Record
			Verify
Activated			Operate
			Deposit
			Withdraw
			Consult balance

(iii) For every symbol (subjects, objects, and verbs), we must identify to which state symbol it is related. A symbol could be associated with one state for different reasons. It could be related because the symbol appears or is created in that state. Alternatively, the symbol may be coupled or more involved with the state (although the symbol is created in a previous state). Table 4 shows the symbols of the banking application related to each state. Detailed guidelines to relate symbols are listed below:

- A symbol is related to a state because…
  - the action can only be performed in that state.
  - the object can only be used in that state.
  - the subject can only act in that state.
- A symbol is related to a state because the symbol is created in that state.
- A symbol is related to a state because although the symbol is created in another state…
  - the action is performed frequently in this state.
  - the object is repeatedly used in this state.
  - the subject performs many actions in this state.
- The initial state groups symbols created before the beginning of the workflow.

It is worth mentioning that we propose an analysis at group level because an analysis at the level of symbols may be problematic, since symbols will not necessarily be represented as modules, while states describe a conceptual entity which in some way will be represented in the final application. In the example, *verify* and *record* would be aspects, but it is only a matter of granularity. We cannot

ensure that every symbol identified will be an aspect, because it could have a very low level of granularity.

4.2 Reference counting

Reference counting consists in calculating the number of references from behavioral responses of symbols of each group to symbols in other groups. It is important to exclude references from the same group, because we want to measure the coupling among different groups. Then, the average of references is calculated according to the number of symbols that the group has. This average must be used instead of the absolute frequency because groups can have very different numbers of symbols, and bigger groups need to have proportionally more references than smaller ones to be considered coupled. Thus, general average of a group is calculated by counting references from behavioral responses of symbols of other groups to any symbol of the group (considering all categories). Then, this value must be divided by the number of symbols. In addition to this, it is important to count the references from each group separately in order to measure the scattering among different groups. Finally, in order to obtain accuracy in the analysis, the average of references must be calculated by categories: subject, object, and verbs. Hence, average by category (subject, object, and verbs) is calculated by counting references from behavioral responses of symbols of other groups to the symbols of specific category of the group. Then, this value must be divided by the number of symbols of the considered category. This distinction is important, because traditional methodologies analyze only verbs, while our approach also considers subjects and objects. Therefore, these data can be used to compare our results to those from other methodologies. The algorithm in Fig. 6 summarizes the counting strategy. Table 5 shows the references between symbols grouped by states. Self-references, of course, are removed.

Table 6 summarizes the references to each state grouped into categories of LEL symbols, and it also shows general values. Let us explain values for the first group, *signed*. The first column, *number of groups that have references*, has a value of 2 because group *signed* has references from the other two groups: *blocked* and *activated*. Then, the following column gives more detail as to the number of references to the whole group, and the three following columns show references to each category of symbols. The last column, *average of references to verbs*, has a null value, because the *signed* group has only one verb, *open an account,* and it has no references. Then, the column *average of references to objects* has a value of 11, because the *signed* group has only one object, *account,* which has 3 references from the *blocked* group and 8 references from the *activated* group, so the total of references is 11. As it is only one symbol, the average of total references divided by the number of symbols is 11. The column *average of references to subjects* has a value of 10, because the *signed* group has 2 subjects, *bank* and *client*. *Bank* has 6 references from *blocked* and 10 references from *activated*, making a total of 16 references. The *Client* symbol has 2 references from *blocked* and 2 references from *activated*, which gives a total of 4 references. Therefore, *client* and *bank* have a total of 20 references, and the average of references to subjects is 10 (20 references divided by 2 subjects). Finally, *signed* has a grand total of 31 references (11 references to the object and 20 references to the subjects), and as the group has 5 symbols (*signed*, *bank*, *client*, *account*, and *open an account*), the general average is 6.2.

4.3 Ranking of groups

Van Den Berg and Conejero [32] define two characteristics that determine the presence of crosscutting concerns: scattering and tangling. He states that scattering occurs when "a source element is related to multiple target elements," whereas tangling occurs when "a target element is related to multiple source elements." In our approach, we refer to elements (source and target) as a group of symbols. We use the references from behavioral responses in order to measure tangling and scattering. Hence, scattering and tangling can be interpreted as (i) different groups that refer

```
For each group G in LEL
        totalGroup := 0
        For each category C in (subject, objet, verb)
                totalCategory := 0
                For each symbol S in G where S belongs to C
                        Count in referencesToSymbol references from symbol S2 to symbol S where
S2 does not belong to G
                        totalCategory := totalCategory + referencesToSymbol
                averageCategory := totalCategory / numberOfSymbolOfCategory C
                totalgroup := totalgroup + totalCategory
        averageGeneral := totalGroup / numberOfSymbol
        Count in differentGroups different group G2 where G2 has a symbol S2 where S2 has references
to S
```

Fig. 6 Counting strategy algorithm

Table 5 References between different groups of the banking application

		Signed					Blocked					Activated				
		Signed	Bank	Client	Account	Open an account	Blocked	Log	Activate	Record	Verify	Activated	Operate	Deposit	Withdraw	Consult balance
Signed	Signed						1									
	Bank							1		1	1		1			
	Client													1	1	1
	Account													1	1	1
	Open an account															
Blocked	Blocked		1	1	2							1	1			
	Log		2									2				1
	Activate		1		1											
	Record		1													
	Verify		1	1												
Activated	Activated			1	1											
	Operate			1	1											
	Deposit		3		2			1		1	1					
	Withdraw		4		3			1		1	1					
	Consult balance		3		1			1		1	1					

Table 6 Reference counting for the groups of the banking application

		Number of groups that have references	General average of references	Average of references to subjects	Average of references to objects	Average of references to verbs
1	Signed	2	6.2	10.0	11.0	0
2	Blocked	2	2.6	0	4.0	2.7
3	Activated	2	2.4	0	0	2.3

to them and (ii) average of references. These two variables determine 4 possible situations. Group candidates for crosscutting concerns must maximize both variables, because (i) the number of different groups indicates how scattered the group is, while (ii) the average of references indicates how coupled (tangled) the group is. Groups which maximize (i) but have a low rank in (ii) are groups which are referred to by many other groups, but have a low general average of references. One scenario of the low general average is the one in which references are all concentrated on a subset of symbols of the group. So, this subset of symbols could be considered a candidate for crosscutting concerns. Groups which have a low rank in (i) but have a high value in (ii) are groups which are tightly coupled to only a few groups, and they are not scattered enough to be considered crosscutting. Finally, groups which minimize (i) and minimize (ii) must be ignored. Figure 7 shows a diagram where the *x* axis represents average of references (i.e., how tangled a group is, -ii-), while the *y* axis represents the different groups that have references (i.e., how scattered they are, -i-).

The dispersion of groups from the banking application according to number of groups that have references and the general average of references is very particular. Since the banking application has only 3 groups and each group has references from the other 2 groups, all states are in the same *y* value, but as they have a different *general average of references*, they are spread along the *x* axis. In this situation, it is easy to conclude a linear order, because all the groups have the same number of different groups that refer to them, and we only need to sort them according to average of references. Therefore, the first candidate to be considered a crosscutting concern is the group with the highest number of average of references, that is, number 1. Then, it is followed by number 2, and finally by number 3. In this example, the diagram is not necessary, and it can be deduced directly from Table 6. Table 7 shows the final order.

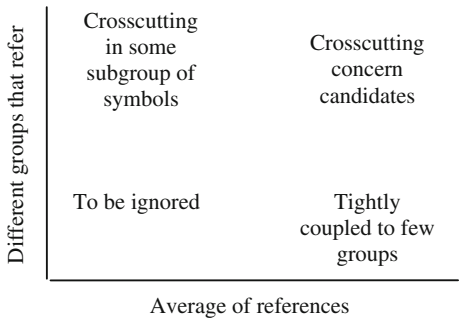

Fig. 7 Level of scattering and coupling and its implication for the identification of crosscutting concerns

4.4 Final analysis

An analysis must be performed after groups and reference counting have been carried out. It consists in analyzing and determining which symbols could alter and bias the ranking of groups. This analysis must be done manually because the requirements engineer is the one who knows the application domain and can decide which part will be designed as core and which as crosscutting concern. The strategy proposes ranking the groups (and symbols) in order to help the requirements engineer make the decision. Some issues are obvious crosscutting concerns, such as quality attributes, but there are others like functional crosscutting concerns that must be detected in the application domain.

The requirements engineer has to pay attention to the main symbols (objects, subjects, and verbs) in the application language, which will not be modularized as crosscutting concerns, because they constitute the core of the application. Therefore, if a symbol has very high values and it causes the group to have very high values because of its presence, the group must not be considered. However, groups with homogenous values in all of its symbols and with high ranks in both variables must be considered crosscutting concerns. Summing up, the steps to perform final analysis are as follows:

Table 7 Final order of the groups from the banking application

		Number of groups that have references	General average of references
1	Signed	2	6.2
2	Blocked	2	2.6
3	Activated	2	2.4

Table 8 References between different groups of the banking application with columns showing core symbols

		Signed					Blocked					Activated				
		Signed	Bank	Client	Account	Open an account	Blocked	Log	Activate	Record	Verify	Activated	Operate	Deposit	Withdraw	Consult balance
Signed	Signed						1									
	Bank							1		1	1		1			
	Client													1	1	1
	Account													1	1	1
	Open an account															
Blocked	Blocked		1	1	2							1	1			
	Log		2									2				1
	Activate		1		1											
	Record		1													
	Verify		1	1												
Activated	Activated			1	1											
	Operate			1	1											
	Deposit		3		2			1		1	1					
	Withdraw		4		3			1		1	1					
	Consult balance		3		1			1		1	1					

Table 9 Reference counting for the banking application without considering core symbols' references

		Number of groups that have references	General average of references	Average of references to subjects	Average of references to objects	Average of references to verbs
1	Signed	2	1.4	4.0	0	0
2	Blocked	2	2.6	0	4.0	2.7
3	Activated	2	2.5	0	0	2.5

- Inspect symbols (mainly with high ranks) in every group to determine if they are core symbols.
- Not count the core symbols' references.
- Recalculate the ranking of groups.

The core symbols with high ranks in the banking application are *bank* and *account*. There are other core symbols such as *deposit*, *withdraw*, and *consult balance*. Table 8 shows the core symbols shaded in gray. These core symbols are shaded only in the columns because the references they receive must be excluded from the counting of the group. Table 9 calculates again the reference counting without considering the references the core symbols receive. With this new result, the groups are reordered into *blocked*, *activated*, and *signed*.

Table 10 summarizes the analysis which determines that only the *blocked* group must be considered a crosscutting concern.

5 Case studies

We have used our approach in two real applications. One application is part of the tax system in Argentina [3], and the other is a proprietary system of a company which publishes news from different Latin American countries in a Web portal [3]. The applications have been implemented, and they are in a maintenance period, and neither of them has been designed with aspects for different reasons. The antitax evasion system was developed while the team was not acquainted with aspect technology; moreover, the tools used do not support this technology. In relation to the Web portal application, when its development started, requirements were very poor and aspectual functionality was not detected. Afterward, the application was extended in a disordered way and aspects were detected, but they were never factorized as such.

Table 10 Detailed analysis of each group of the banking application

Id	State	Description
1	Signed	The *bank* and *account* symbols are the main subject and object respectively in the application language and they alter the values. Therefore, this group must not be considered a candidate
2	Blocked	This group has only objects and verbs, but references are balanced among all of them. Therefore, this group must be considered a candidate
3	Activated	This group has only verbs, and references are quite balanced among all of them. But these verbs are considered main symbols in the language application. Therefore, this group must not be considered a candidate

Table 11 Measures of the application LELs and Use Case Points from the case studies

	Use cases	Unadjusted use case points	Symbols	References
Tax	84	1,125	63	160
News	87	915	54	121

One of the co-authors of this article has been working on both applications for several years, and he knows them very well. He has worked for more than 10 years with the first application and for more than 3 years with the second one. Although he built some partial LEL during development of both applications, both these LELs needed to be completed and checked. The fact that there were versions of LELs previous to the development of this technique (although they were incomplete) shows that the experiment was not biased.

It is important to mention that although both applications used in these case studies are part of bigger systems, the size of the application is big enough to show the approach is applicable and scalable to larger systems. The size of the applications calculated in unadjusted use case points [17] is 1125 for the Tax application and 915 for the News application. Both of them have more than 80 use cases. The size of the LELs can be measured according to the symbols described and the references between them. The Tax application has 63 symbols and 160 references, while the News application has 54 symbols and 121 references. This information is summarized in Table 11. LEL construction and calculation were performed using basic and general purpose tools such as word processors and spreadsheets, but note that we are currently working on a specific tool [4] to assist in the application of the strategy.

We can justify the suitability and effectiveness of our approach in 3 ways. First, the approach is described and also justified in the previous section. Then, in this section, effectiveness is justified in two more ways. At the end of each case of study, we compare the results obtained by our approach with the expert opinions of different people involved with both systems. One of the experts is one of the co-authors who were involved with both applications. He gives his opinion from his experience as designer and developer of the first application, and as requirements engineer of the second one. Nevertheless, although he has a qualified opinion, it is still subjective. So, as yet another means to prove effectiveness, we present other expert opinions which were gathered through structured interviews conducted with members of the development team, in order to contrast the results from our approach with the objective opinion of other people involved with the application. The following questions were asked: (i) What role did you play during development? (ii) how many years of experience do you have? (iii) what are your academic studies? (iv) which functionality do you consider to be crosscutting concerns? (The interviewees were given LEL groups and were asked to identify crosscutting concerns from the groups.) and (v) justify your choice in the previous question.

The first application is rich in situations and actions that are carried out during the prosecution phase. The second application relies on technology, because many actions which are developed make sense only within the computer world. So, although we emphasize how important it is to identify crosscutting concerns in analysis stage, we show with the second case study that we can use the approach to identify crosscutting concerns in source code too. Section 5.1 describes the analysis performed in the antitax evasion application, while Sect. 5.2 describes the analysis of the Web portal.

5.1 The antitax evasion case study

When a citizen does not comply with his duty to pay debts, the petitioner performs the necessary actions to get the debt paid. The normal course of action involves the following: (i) establishment of the debt; (ii) internal procedural measures; (iii) active procedural measures; (iv) pending procedural measures; and (v) liquidation of the debt.

The five steps indicated above are complex activities that include the following tasks:

(i) Establishment of the debt: Situation where a taxpayer has incurred a debt and he has dismissed the opportunities to pay it.
(ii) Internal procedural measures: At this stage, a cost-benefit analysis must be done, in order to determine if the effort of claiming the debt is profitable. If it is profitable, it is also necessary to determine if it is a

critical debt (of a large sum of money, for example) so as to take preventive actions.

(iii) Active procedural measures: They consist of 4 steps, namely: bring a prosecution against the debtor in court, obligate him to pay, adopt preventive measures, and finally obtain the sentence from the judge. Since preventive measures could be taken in stage (ii), the sequence of steps is not unique.

(iv) Pending procedural measures: They basically consist in waiting for the debtor to voluntarily pay the debt. It is expected that he will do it because his actions are legally restrained by the preventive measures. Preventive measures expire, so they must be renewed if necessary. Nevertheless, if the debtor has belongings (a car or a house), it is possible to perform an auction to liquidate the debt if certain conditions are fulfilled.

(v) Liquidation of the debt: At this stage, the prosecution phase is finished, the debt is recovered, and the preventive measures are lifted.

There are some situations that can alter the normal flow of events:

(vi) Debt claim with payment: It occurs when a debt is claimed, but it does not exist, because an administrative mistake has occurred and a claim is made by mistake. Although the debt does not exist, the taxpayer will have to pay administrative costs, because he could have clarified this situation in previous steps.

(vii) Moratorium: It is a mechanism that allows the taxpayer to pay the debt in monthly payments.

(viii) Auction: Situation where the debtor has goods (a car or house) which are auctioned to recover the debt.

(ix) Administrative preventive measure: Situation where it is not profitable to file a lawsuit to claim the debt. So, an administrative preventive measure is adopted in order to legally restrain the actions of the debtor and force him to pay the debt.

When analyzing the antitax evasion application, we defined the nine groups mentioned previously in order to construct LEL and we performed the references counting showed in Table 12. Figure 8 shows dispersion of groups according to the number of groups that have references and the general average of references.

In general, groups are located in the main diagonal across Fig. 8, so it is easy to find a linear sequence to the groups. Groups 2 and 1 are definitively in first and second place, as their location are confirmed on the diagram of Fig. 8 and the values of Table 12. Then, group 3 must be considered, because it has approximately the same value of average of references from groups 5, 6, and 7, but group 3 has a high value of different groups that refer to it. Then, groups 7, 5, and 6 must be located. Groups 5 and 6 have the same values of average of references and different groups that refer to them, but group 5 is placed before group 6 because this group is only related to verbs, while group 5 is related to subjects and verbs; this variety makes this group the first. Then, group 8 must be located, because although it is smaller than group 9 regarding average of references, group 8 has a larger number of different groups that refer to it. Group 8 has 2 different groups, while group 9 has only 1, which is why it is not sorted in a different way. Finally, groups 9 and 4 must be located.

Although Fig. 8 shows the dispersion of groups, it is necessary to perform a detailed analysis for every group, because a group could have a main symbol which may alter the ranking values. Table 13 shows the detailed analysis, and it finally identifies the crosscutting concerns.

Summing up, the groups that must be considered crosscutting concern candidates (sorted out from the most likely to the less) are *Internal procedural measures*, *Moratorium*, *Liquidation of the debt*, and *Administrative preventive measure*.

One of the co-authors, who were involved in the development, agreed that the four candidates identified by the approach are candidates for crosscutting concerns. In particular, he identifies in the first place *moratorium* and *liquidation of the debt* as the most likely candidates, because whatever the state the tax evasion trial is in, it could continue with *moratorium* if the debt is real or it could continue with *liquidation of the debt* if the debt is not real. Therefore, the application needs to have functionality to allow trials to be moved from some states to either of these states. Then, after a more detailed analysis, the co-author concluded that *internal procedural measures* and *administrative preventive measure* are candidates for crosscutting concerns. *Internal procedural measures* reaches this status because this group has an object which is frequently referred to by other groups. That means that in the software application, there will be a resource (file, data structure, or an object) that will be used from many different places. *Administrative preventive measure* is in a similar situation because it has verbs which are frequently referred to, that is, in the software application that there will be invocation of methods or functions from several places. So, just as it occurs in the banking application where each operation has to be enriched with login and authorization functionality, in this application many of the states in which a trial can be must be enriched with functionality to move the trial to *moratorium* and *liquidation of the debt*. Moving means not only changing the state, but also adjusting some things related to this change of state. Then, as many groups refer to objects or verbs of *internal*

Table 12 Reference counting for the groups of the tax evasion application

		Number of groups that have references	General average of references	Average of references to subjects	Average of references to objects	Average of references to verbs
1	Establishment of the debt	7	3.7	10	2.3	1.2
2	Internal procedural measures	7	4.3	6	4.7	0
3	Active procedural measures	8	2.0	2	3.3	0.5
4	Pending procedural measures	1	0.7	0	0	0
5	Liquidation of the debt	4	2.0	2	0	1.3
6	Administrative preventive measure	4	2.0	0	0	2.2
7	Moratorium	4	2.3	0	0	1.7
8	Auction	2	1.0	0	0	2.0
9	Debt claim with payment	1	3.0	0	0	2.0

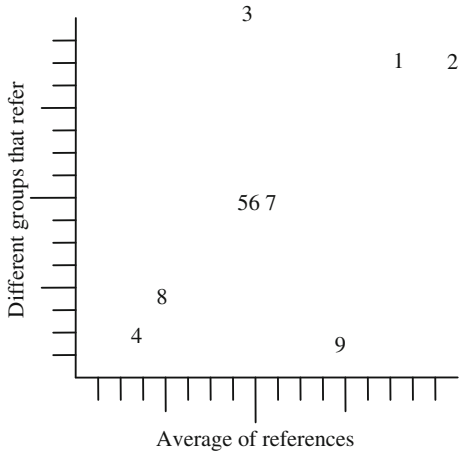

Fig. 8 Dispersion of groups from the tax evasion application

procedural measures and *administrative preventive measure*, many states must be enriched with this functionality.

Apart from the co-author's opinion, two people were interviewed who were involved with the development of the application and are acquainted with aspects technology. One of them has 28 years of experience in the field and fulfilled the roles of requirements engineer, designer, and architect in the development of the application; the other has 25 years of experience, and she was the project leader and also fulfilled the role of requirements engineer in the development of the application. Both of them have a degree in computer science.

The first interviewee considered the following functionality to be crosscutting concerns: *Moratorium*, *Liquidation of the debt*, and *Administrative preventive measure*. He based his opinion on the fact that trials follow a linear sequence of steps; nevertheless, there are some steps that are transversal to the linear steps. *Moratorium* is one of them. At the beginning of the development, there was only one point where a trial could get into moratorium, but at the end of the development, it became an option that could be reached no matter where the trial was. *Administrative preventive measure* was similar to moratorium. Then, *liquidation of the debt* was the final state of the events in the trial, and it could only be reached from the previous step, but as moratorium and administrative preventive measure, liquidation could be reached from various states.

The second interviewee only considered *moratorium* a crosscutting concern. She claimed that *moratorium* is a privilege that can be accessed from any other situation in which the trial is. Therefore, she considered it a crosscutting concern because *moratorium* is extra functionality that must be added in any situation in which the trial could be.

Although there were only two interviewees, they are people with great experience, and it is meaningful that the functionality they have identified as crosscutting concerns was also identified by our approach. Our approach identified 4 candidates; one of the interviewees identified 3 of them, and the other identified 1 of them. The group that our approach considered most likely to be a crosscutting concern was not selected by any of the two interviewees, but the group ranked in second position was considered a crosscutting concern by both interviewees.

After the analysis, both interviewees were asked why they did not choose *internal procedural measures* as a crosscutting concerns candidate. Both said that clearly, it is

Table 13 Detailed analysis of each group from the tax evasion application

Id	State	Description
2	Internal procedural measures	Symbol *petitioner* is a main subject in the language application and it alters the values. Nevertheless, the average of references to subjects only reaches a medium level, while there is an object *goods* which has many references and must be considered. Therefore, this group must be considered a candidate
1	Establishment of the debt	This group has the *debtor* symbol, which is a main subject in the language application, so it alters the values. *Debtor* has 32 references while the general average is 3.7. If *debtor* did not belong to this group, the general average would not be so high. So, this group must not be considered a candidate
3	Active procedural measures	Symbol *prosecution* is a main object in the language application. This symbol has 10 references, while the general average of the group is 2. So this symbol alters the values, and this group must not be considered a candidate
7	Moratorium	The number of groups that have references to this group and the general average is ranked as medium, so this group is quite scattered and it is quite coupled. This group has a high average of references to verbs. Therefore, this group corresponds with traditionally identified crosscutting concerns, so it must be considered a candidate
5	Liquidation of the debt	The number of groups that have references to this group and the general average of references are ranked as medium, so this group is quite scattered and it is quite coupled. This group has a medium average of references to verbs. Therefore, this group corresponds with traditionally identified crosscutting concerns. So it must be considered a candidate
6	Administrative preventive measure	The number of groups that have references to this group and the general average of references are ranked as medium, so this group is quite scattered and it is quite coupled. This group has a high average of references to verbs. Therefore, this group corresponds with traditionally identified crosscutting concerns, so it must be considered a candidate
8	Auction	The number of different groups that refer to this group as well as the general average of references is low, so this group must not be considered a candidate
9	Debt claim with payment	Although this group has a high general average of references, the number of different groups that refer to this group is low. This group is coupled with few groups, in fact it is coupled with only 1 group. So this group must not be considered a candidate
4	Pending procedural measures	The number of different groups that refer to this group as well as the general average of references is low, so this group must not be considered a candidate

not aspectual functionality, but after analyzing in detail the symbols of this group, they admitted that it could probably be considered a crosscutting concern.

5.2 The Web portal case study

The approach was also used to identify crosscutting concerns in an application about a Web portal that presents daily news. News items are developed by writers; editors review them and decide whether to publish them or not. The publication of news involves creating an HTML page and some other formats to show on mobile phones and other devices. Apart from different formats, some kind of categorization must be done to news, too. The construction of news items moves through different states until they are finally published. These states are as follows: written, accepted, HTML built, auxiliary versions built, categorized, tagged, included in calendar, and published.

The first state is *written*. From that state, a news item can go to a *rejected* state, which is a final state. But if the news item is accepted, it reaches an *accepted* state; from there, it goes to an *html built* state and after that to an *auxiliary versions built* state. From this state, news can go to none, one or all of the following states: *categorized*, *tagged*, and *included in calendar*. Finally, the *published* state can be reached.

The following table shows the nine groups related to the nine states with reference counting for each state (Table 14).

Group 3 is definitively the first one, since it has the highest values in average of references and different groups that refer to it. Then, groups 1 and 9 have the same value in average of references, but group 1 has a high value in different groups that refer to it, so the second place is for group 1. Then, group 9 has a very low value in different groups that refer to it (only 1), so the third position is for group 4. Then, come all the groups with value 1 in different groups that refer to them. Table 15 shows groups in a linear order according to the analysis.

Table 16 shows the detailed analysis, and it finally identifies the crosscutting concerns. In conclusion, according to the analysis in Table 16, the only group that must be considered a crosscutting concerns candidate is *html built*.

Using our approach, it is easy to identify group *html built* as a crosscutting candidate, because it is one of the

Table 14 Reference counting for groups from the news portal application

		Number of groups that have references	General average of references	Average of references to subjects	Average of references to objects	Average of references to verbs
1	Written	5	2.5	2	6.3	0
2	Rejected	1	0.5	0	0	0
3	Accepted	5	13.4	32	0	1.0
4	Html built	3	0.9	0	0.3	1.4
5	Auxiliary versions built	1	0.2	0	0	0.2
6	Categorized	1	0.6	0	0	0.2
7	Included in calendar	1	1.3	0	0	1.0
8	Tagged	1	1.0	0	0	0.5
9	Published	1	2.5	0	0	1.0

Table 15 Final order of groups from the news portal application

		Number of groups that have references	General average of references
3	Accepted	5	13.4
1	Written	5	2.5
4	Html built	3	0.9
9	Published	1	2.5
7	Included in calendar	1	1.3
8	Tagged	1	1.0
6	Categorized	1	0.6
2	Rejected	1	0.5
5	Auxiliary versions built	1	0.2

three groups that has more than 1 group with references to it. But one of the co-authors was involved in this development, and it was difficult for him to identify it without a detailed analysis. At the beginning of the development, the application did not have these requirements, so we did not identify this functionality. But with the information used to build this LEL and after analyzing it in detail, it can be concluded that group *html built* has many verbs, in fact, methods, or functions in the software application that will be implemented in many other groups. So, the fact that this group has some functionality that is widespread and used by other groups justifies the essence of being crosscutting concerns.

Apart from this opinion, an interview was conducted with 7 members of the development team. According to work experience, we can arrange the 7 members into three groups. The most experienced group contains only one member, the project leader/architect/designer/requirements engineer of the team, a graduate in computer science with 12 years of experience. Then, there is a middle-experienced group of 4 members with 3–5 years of experience; 1 of them is a graduate in computer science, and the other 3 are students. All of them were designers and/or developers. Finally, in the least experienced group are the last two members of the team, students with less than 2 years of experience who had the role of developers.

The only group which was identified as crosscutting concerns was *html built*. The majority of the members identified it (4 out of 7). The most experienced member of the team agreed with this view. He said: "Although I thought that aspects were *Auxiliary versions built*, *Categorized*, *Included in calendar* and *Tagged*, after analyzing the symbols in the *html built* group, I am completely sure that this group must be the crosscutting concerns, because it contains functionality which was added gradually and in a disordered way within the application. Only when we were advanced into the development of the application did we realize that it would have been convenient to design this functionality as crosscutting concerns." Then, from the middle-experienced group of people, 2 out of 4 agreed on *html built* as crosscutting concerns. One argumentation in favor said: "The methods which are grouped into *html built* are clearly methods shared by the whole application and, above all, they are methods which have suffered many changes during software development." There is another argumentation in favor of *html built,* but with some doubts: "Although it is complex to understand the design of the application through groups of states, analyzing methods which are located in each of the groups contributes to a better understanding. Some of the methods defined in *html built* could be located in another group, and I am not convinced if it is necessary to factorize them as aspects. If this is done, several aspects must be identified and not only one." Finally, one argumentation against the identification also claims that some kind of factorization is necessary, but they are not sure if it must be done using aspect technology.

Table 16 Detailed analysis of each group from the news portal application

Id	State	Description
3	Accepted	This group has the *publisher* subject symbol, which is a main subject in the language application, so it alters the values. Publisher has 63 references while the general average is 13.4. If *publisher* did not belong to this group, the general average would not be so high. So this group must not be considered a candidate
1	Written	Symbol *content* has 12 references and a general average of 2.5. But *content* is a main object in the language application, so this group must not be considered a candidate
4	Html built	This group has several verbs which are all referred to in the same way by three other groups. So this group must be considered a candidate
9	Published	Although this group has a high value in general average of references, there is only 1 group with references to it. So this group must not be considered a candidate
7	Included in calendar	Although this group has a high value in general average of references, there is only 1 group with references to it. So this group must not be considered a candidate
8	Tagged	Although this group has a high value in general average of references, there is only 1 group with references to it. So this group must not be considered a candidate
6	Categorized	This group has a low value in general average of references, and there is only 1 group with references to it. So this group must not be considered a candidate
2	Rejected	This group has a low value in general average of references, and there is only 1 group with references to it. So this group must not be considered a candidate
5	Auxiliary versions built	This group has a low value in general average of references, and there is only 1 group with references to it. So this group must not be considered a candidate

In this case, opinions about identification of crosscutting concerns are divided. Although the majority agreed on their views, it is not a great majority. Moreover, some of the people that identified the same crosscutting concerns are not really convinced of that. So, it is important to mention that the approach identifies crosscutting concern candidates that experts must otherwise analyze in detail to determine whether they are or not crosscutting concerns. It is worth pointing out that it does not matter if *html built* is a crosscutting concern or not. What is important is that our approach suggests crosscutting concern candidates, and even if team members do not agree on that, they agree that if there are crosscutting concerns, it must be *html built.*

6 Conclusions

In this paper, we have described an approach for identifying crosscutting concerns in the application language captured by the LEL. We have provided three real world examples to illustrate how LEL can be analyzed in order to identify crosscutting concern candidates. We have shown that in essence our approach has similarities with many other techniques. In spite of these similarities, our approach has the unique advantage of being very easy to use, since it begins with a description of the application language, a task which can be carried out by any stakeholder without any kind of expertise and with little effort. As simple as it is, the description produced in LEL is very rich, and it can contain knowledge which may not be present in other products of software development. Taking into account that LEL is a model of the application language, our approach brings up discussion of crosscutting modularization at a very early stage in the development process. This strategic information at hand is helpful to make critical architectural decisions. It is a great benefit to have such important data to build the best design possible at the beginning of the construction. Making changes in the middle of the development process can be very difficult, and sometimes, the necessary changes in the application are so enormous that they are considered impossible.

Our approach identifies crosscutting concerns as early as possible in the software development process. We have already proved that it is possible to perform this analysis while learning the application language. We have also shown that performing this identification early brings benefits to the development process. If we identify crosscutting concerns while capturing and understanding the application language, we will have more information available to write requirements, to define the architecture, and to design the software. This being said, the analysis proposed can be done at almost any stage of the software development, whether in requirements or source code, too. The key element that allows for this is LEL. We use LEL to synthesize the knowledge of the application language. LEL organizes the knowledge into symbols which have connections to other symbols, and theses connections build a net (a hypertext). In this net, symbols are grouped according to states which arise from the application language. We then apply our approach to this model. But such a model can also be built from requirements or source code. These products capture the knowledge relevant to the application, and we can use them to construct LEL. Of course, the language elicited from domain experts will be richer than that captured from the source code. Moreover, the knowledge elicited from the application domain will be more abstract than the knowledge extracted from the source code. Thus, the strategy proposed will make a better

use of the language captured from the application domain because it is designed to work with high-level abstraction artifacts. This means the results will be also abstract, that is, coarse-grained. Late steps in software development such as codification make possible to apply techniques which identify crosscutting concerns more accurately, but early stages have high-level abstraction artifacts, so the analysis cannot produce a result different from a coarse-grained one. Therefore, although LEL can be built from requirements or source code, we do recommend building it directly from the application language, so as to apply the approach early on and make good use of the available knowledge, avoiding reworking of items.

LEL, the technique used to model the language application, is a very convenient tool for experts with no technical skills, although people with such skills will obtain more profit from its use. The convenience of LEL as a tool arises from 3 significant characteristics: It is easy to learn, it is easy to use, and it has good expressiveness. There are several publications which use LEL in complex domains and validate these claims. Gil et al. [16] state that "building a LEL in an application completely unknown to the requirements engineer and with highly complex language can be considered a successful experience, since users stated that requirement engineers have developed a great knowledge about the application." In another work, Cysneiros and Leite [15] state that "the use of LEL was very well accepted and understood by the stakeholders. As these stakeholders were nontechnical experts from a specific and complex domain, the authors believe that LEL can be suitable to carry out in many other domains." These are examples of real uses of LEL which confirm our claims, but we can justify its characteristics, too. LEL is easy to learn because it uses natural language, and the expert has to write sentences according to standard grammar rules. The expert must only be trained in the template that guides the description of every symbol. Then, it is also easy to use because the expert only has to describe symbols trying to maintain a similar level of abstraction in each description. Obviously, the identification of links between symbols is a time-consuming task, but although the expert must have in mind a cyclic definition of symbols, the LEL description must be assisted by a tool which defines the links between symbols. Finally, LEL expressiveness is anchored on the fact that it contains colloquial descriptions. Descriptions can be short and simple, or they can be complex and longer, but they can be phrased in a colloquial way. Therefore, these three characteristics allow the expert to pay attention to the knowledge he has to organize and model with LEL, and he does not have to worry about how to describe this knowledge.

While traditional approaches rely on actions to identify crosscutting concerns, our approach considers not only actions but also other semantic categories. LEL organizes its symbols into four basic categories: subjects, objects, verbs, and states. We use all the categories to identify crosscutting concerns. Verb symbols best help to identify crosscutting concerns; nevertheless, subjects, objects, and states also help to confirm the crosscutting concerns identified by the verbs. What is more, they may identify crosscutting concerns which were not identified by verbs. In LEL description, the same action may be described by different verbs, i.e., synonyms, but they might not be identified. If the action is related to the same object, the fact that the object is the same one will identify a crosscutting concern, even if the action will not identify it.

Although our approach has significant characteristics which make it suitable to use in most cases, there are still some issues which we must continue working on. These will be pointed out in the following paragraphs.

Our approach relies on identifying a state machine in the language application, in order to identify which states can be considered crosscutting concerns. We have to continue working on more applications in order to determine if it is possible to have applications with no state machines or with more than one. In the first case, our approach cannot be used. In the second case, if there is more than one state machine, we have to analyze if all of them have the same level of importance. If they do, we must find a way to adapt our approach to that situation. Basically, we have to consider how state machines interact with each other and how to organize LEL symbols according to these state machines in order to perform the counting.

The detail level used to describe LEL is also important. Symbols must be identified uniformly, and they must be described using the same level of abstraction because both these elements have an influence on the results. Our approach involves a counting scheme between the behavioral responses of symbols from one state to another. If one state has many symbols which are not correct, this state could in fact have many more references and it might become a crosscutting concern when it should not be. We use an average of references instead of absolute frequency of references to avoid this defect. Nevertheless, it is worth highlighting that in LEL construction, only symbols from the language application that are really relevant to the system must be considered. At the same time, descriptions of behavioral responses affect our approach in the same way. If one symbol has an extensive description with a great level of detail, it will produce many references that will affect the counting schemes. This extensive description may refer to many symbols from one or various states, and it would cause the position of the states in the ranking to be higher than the real rank. Therefore, the identification of symbols is important as well as the description of behavioral responses. We will continue working on

determining some conditions (maybe metrics) that LEL must fulfill in order to guarantee our results.

In relation to the previous topic, there is an issue connected with the use of abstract and concrete symbols. During the description of a language application, a symbol can be identified which generalizes the essence of many other symbols. In this situation, the symbols are organized in a kind of "is a" hierarchy. That means concrete symbols are in fact symbols with the same essence as abstract symbols. Therefore, references to the concrete symbols can be made to the abstract symbols as well. As a result, counting will be different when only concrete symbols are referred to and when only abstract symbols are referred to. This situation must be analyzed in detail.

As mentioned earlier, we are working on an empirical evaluation in order to show the validity of our approach that takes into account the work described in [8]. We are going to perform an experiment where a group of people will work with a well-established approach and another group will work with our approach. At the end, we are going to compare the results from both approaches. People with knowledge of aspect-oriented programming will be selected, but these people will not have experience in the approach they are going to use in order to identify crosscutting concerns. We are going to present them with an application domain which will be unknown to them. Some people will have to apply the Theme/Doc Approach [6], and others will have to apply our approach. In order to accomplish the objective of validation, our approach must identify at least all the crosscutting concerns identified by the Theme/Doc. We do not expect to detect exactly the same crosscutting concerns, but we do expect that the crosscutting concerns identified by Theme/Doc will also be identified by our approach (this is known as recall). We have chosen the Theme/Doc approach to compare with, because it is an accepted and widely recognized approach which has been book published. If our approach identifies the same crosscutting concerns detected by Theme/Doc (and also enrich them), we will have shown validity. Both approaches work with different input; Theme/Doc needs requirements, and our approach needs LEL. We are going to provide this material in the experiment. Explanations about the two approaches will be given, along with some documentation on how to apply them. After this, people will be left to work at their own pace. After a week, the results will be compared. We do not intend to analyze learning or application time; the sole objective is comparing the crosscutting concerns identified. Our approach represents an addition to LEL modeling; it is not our intention to provide an approach in order to replace other approaches. Our aim is to provide a tool for people that work with LEL to identify crosscutting concerns. If we can verify that our approach identifies the same crosscutting concerns as Theme/Doc and more, we can consider the experiment a success [24, 31].

Finally, the use of a tool is clearly necessary to apply our approach. There are two important reasons to consider a tool. First, it is essential to manage LEL symbols and its references. Second, it is necessary to automate the counting schemes. That being said, there is still an activity that cannot be assisted by a tool: the final decision about whether a group is a crosscutting concern or not. This final decision must be decided manually because the requirements engineer or designer is the one who organizes the concerns determining which are core or crosscutting. This decision must be taken considering the knowledge about the application domain and using the information provided by the strategy about the concerns which are spread all over the system.

In relation to the management of symbols, a word processor could be used to write LEL. In fact, at the beginning of the construction of LEL, the use of a word processor could be suitable in order to transform a narrative description into a set of symbols with its descriptions. However, the most time-consuming task is not LEL writing but link construction. It demands great effort to analyze whether every word written in a description is a link or not. Furthermore, if a new symbol is added or an old one is removed, all the symbols must be updated (or at least reviewed) to reflect this change. This task is more complex because words can have different representations (i.e., verbs and its conjugations, singular and plural nouns, etc.), which means it is not easy to implement this functionality. Nevertheless, there is a tool named C&L to assist in this process [13, 19].

In relation to the counting schemes, this functionality must be clearly automated. If the calculation is done manually, many errors may occur. Furthermore, if LEL is managed in an application, this application will have all the information needed for the counting which is well established and straightforward. Thus, we are working on a specific tool [4] to assist in the application of the strategy, consuming the LEL produced by C&L.

In conclusion, our approach presents a number of advantages: (i) It can be applied in different stages of software production; (ii) it relies on construction of LEL, a technique which has been proven to be very effective with experts who do not necessarily belong to the informatics field; and (iii) traditional techniques generally rely on actions in order to identify crosscutting concerns, while our technique relies on subjects, objects, and states too.

There are still some issues that we have to continue working on: (i) the technique depends on the organization of the language application; (ii) our approach relies on LEL and the organization of its descriptions; (iii) a tool must be implemented to assist in some part of the approach; (iv)

experimentation has to be conducted to revalidate the results achieved so far, mainly by comparison with other approaches; (v) a LEL must be applied to a different application of the same domain to evaluate its reusability, (vi) LEL should be adapted in a software product line context to take advantage of its reusability potential, and (vii) we are also working on text mining in order to help the requirements engineer decide about core and crosscutting concern symbols.

Acknowledgments The authors want to thank Alejandro Oliveros for their support and guidance. His help was invaluable through their research on LEL.

References

1. Antonelli L, Rossi G, Leite JCSP (2010) Early identification of crosscutting concerns in the domain model guided by states, In: Proceedings of the 2010 ACM symposium on applied computing, Sierre, Switzerland, ISBN: 978-1-60558-639-7
2. Antonelli L, Rossi G, Leite JCSP, Oliveros A (2012) Deriving requirements specifications from the application domain language captured by Language Extended Lexicon. In: Proceedings of the 2012 workshop in requirements engineering (WER), Buenos Aires, Argentina, April, pp 24–27
3. Antonelli L (2012) Early identification of crosscutting concerns in the application language captured by Language Extended Lexicon, doctoral thesis (in spanish), Facultad de Informática, Universidad Nacional de La Plata. Available at: http://sedici.unlp.edu.ar/handle/10915/18109
4. Aramayo A, Rossi J (2013) TICCWL: a tool to identify crosscutting concerns through the LEL approach, technical report. Available at: http://www.lifia.info.unlp.edu.ar/papers/2013/TICCWL.pdf
5. Araújo J, Whittle J, Kim DK (2004) Modeling and composing scenario-based requirements with aspects. In: International conference in requirements engineering (ICSE). IEEE Computer Society, pp 58–67
6. Baniassad E, Clarke S (2004) Theme: an approach for aspect-oriented analysis and design. In: International conference in software engineering (ICSE). IEEE Computer Society, pp 158–167
7. Baniassad E, Clements PC, Araujo J, Moreira A, Rashid A, Tekinerdogan B (2006) Discovering early aspects. In: IEEE software, ISSN: 0740-7459, vol 23, Issue 1, January, pp 61–70
8. Basili VR (1993) Experimental software engineering issues: critical assessment and future directions. Lect Notes Comput Sci 706(1993):1–12. doi:10.1007/3-540-57092-6_91
9. Boehm BW (1997) Software engineering. Computer society Press, IEEE, Silver Spring
10. Bounour N, Ghoul S, Atil F (2006) A comparative classification of aspect mining approaches. J Comput Sci 2(4), ISSN: 1549-3636, 2005 Science publication, pp 322–325
11. Breitman KK, Leite JCSP (2003) Ontology as a requirements engineering product, In: Proceedings of the 11th IEEE international conference on requirements engineering (RE). IEEE Computer Society, Monterey Bay, California, USA, ISBN: 0-7695-1980-6
12. Brooks F (1995) The mythical man-month: essays on software engineering, 2nd edn. Addison-Wesley Professional, Reading
13. C&L Tool (2009). Available at http://pes.inf.puc-rio.br/cel/. Accessed in October, 2009
14. Chitchyan R, Rashid A, Rayson P, Waters RW (2007) Semantics-based composition for aspect-oriented requirements engineering. In: International conference aspect-oriented software development (AOSD). ACM
15. Cysneiros LM, Leite JCSP (2001) Using the Language Extended Lexicon to support non-functional requirements elicitation. In: Proceedings of the Workshops de Engenharia de Requisitos, Wer'01, Buenos Aires, Argentina
16. Gil GD, Figueroa DA, Oliveros A (2000) Producción del LEL en un Dominio Técnico.Informe de un caso. In: Proceedings of the Workshops de Engenharia de Requisitos, Wer'00, Rio de Janeiro, Brazil
17. Karner G (1993) Metrics for objectory. Master Thesis, Linkoping University (LiTH-IDA-. Ex-9344:21), Linkoping, Sweden
18. Leite JCSP, Franco APM (1993) A strategy for conceptual model acquisition. In: Proceedings of the first IEEE international symposium on requirements engineering. IEEE Computer Society Press, San Diego, California, pp 243–246
19. Leite JCSP, Silva LF, Breitman KK (2005) C&L: Uma Ferramenta de Apoio à Engenharia de Requisitos. In: RITA 122.: ISSN: 0103-4308, Revista de Informática Teórica e Aplicada (RITA), vol XII, Número 1, Junho 2005, pp 23–46
20. Mahoney M (2005) Modeling crosscutting concerns in reactive systems with aspect-orientation. In: Models 2005, doctoral symposium
21. Mahoney M, Elrad T (2007) Generating code from scenario and state based models to address crosscutting concerns. In: Proceedings of the sixth international workshop on scenarios and state machines (SCESM'07). IEEE
22. Mizuno Y (1983) Software quality improvement. IEEE Comput 16(3):66–72
23. Nuseibeh B (2004) Crosscutting requirements. In: Proceedings of the 3rd international conference on aspect-oriented software development, ISBN: 1-58113-842-3, Lancaster, UK, pp 3–4
24. Pfleeger SL (1995) Experimental design and analysis in software engineering: types of experimental design. In: ACM SIGSOFT software engineering notes. ACM, New York, NY, USA, vol 20, Issue 2, April 1995
25. Rago, A, Abait, E, Marcos, C, Pace, AD, (2009) Early aspect identification from use cases using NLP and WSD techniques. In EA '09: Proceedings of the 15th workshop on Early aspects, pp 19–24
26. Rashid A, Moreira A, Araújo J (2003) Modularisation and composition of aspectual requirements. In: Proceedings of the 2nd international conference on aspect-oriented software development, ISBN: 1-58113-660-9, Boston, Massachusetts, pp 11–20
27. Rashid A, Moreira A (2006) Domain models are NOT aspect free. In: Proceedings of MoDELS/UML, Springer, Lecture Notes in Computer Society, pp 155–169
28. Sampaio A, Chitchyan R, Rashid A, Rayson P (2005) EA-Miner: a tool for automating aspect-oriented requirements identification. In: Proceeding of ASE 05. ACM, California, USA, pp 352–355
29. Sampaio A, Loughran N, Rashid A, Rayson P(2005) Mining aspects in requirements. In: Proceeding of the workshop on early aspects (held with AOSD 2005), Illinois, Chicago, USA
30. Shepherd D, Pollock L, Tourwé T (2005) Using languages clues to discovery crosscutting concerns. In: Proceeding of the international conference on software engineering, ISBN: 1-59593-119-8, workshop on modeling and analysis of concerns in software, St. Louis, Missouri, pp 1–6
31. Shull F, Singer J, Sjøberg DIK, Guide to advanced empirical software engineering, 1st edn. Springer, ISBN: 978-1848000438
32. Van Den Berg K, Conejero JM (2005) A conceptual formalization of crosscutting in AOSD. In: Proceeding of Desarrollo de Software Orientado a Aspectos, Granada, España
33. Yu Y, Leite JCSP, Mylopoulos J (2004) From goals to aspects: discovering aspects from requirements goal models. In: International requirements engineering conference, pp 38–47