Identifying Crosscutting Concerns in Requirements Specifications

Lars Rosenhainer
Department of Computer Science
Chemnitz University of Technology
09107 Chemnitz, Germany
lars.rosenhainer@informatik.tu-chemnitz.de

Abstract

Identifying and documenting early crosscutting concerns, i.e. requirements-level crosscutting concerns, is critical. It improves traceability among requirements as well as between requirements and downstream artifacts, facilitates easier assessment of change impact, supports requirements evolution, enables the application of aspect-orientation from the very start of the software lifecycle and prevents easy oversight of crosscutting influences. However, the identification of requirements-level crosscutting concerns has been neglected in countless software projects but may—with the growing dissemination of aspect-oriented approaches—retroactively become relevant for some or even many of them. This paper, therefore, describes an approach to the identification problem from an aspect-mining point of view, i.e. to identify crosscutting elements in pre-existing requirements specifications, and gives reasons why such an approach is highly desirable. Two techniques for aspect mining from these documents are suggested. Further, some preliminary results of a case study conducted to confirm the feasibility of the approach, are presented.

1 Introduction

Developing software for industries and organizations is typically a highly complex undertaking since the software to be developed is itself of a highly complex nature, often comprising several million lines of code. A large number of needs, wishes, requirements—often from different viewpoints and oftentimes conflicting with each other—must be harmonized to find a solution respecting the diverse interests. Software development, therefore, has to deal with a large number of concerns. Some concerns are related to the product (the software) to be developed, e.g. functionality and performance. Other concerns are related to the development process itself, e.g. development cost and time. Sutton Jr. and Rouvellou [15] have thus proposed a definition of the term *concern* as being ”any matter of interest in a software system”—a definition that will be followed in this paper.

Separation of concerns [7] (SoC) is a basic principle of software engineering. Derived from common sense, it essentially means that dealing successfully with complex problems is only possible by dividing the complexity into sub-problems that can be handled and solved separately from each other. The partial solutions can then be combined to a complete solution. One incarnation of the SoC principle is modularization: each module is responsible for certain concerns of a software system, all modules together realize all the concerns of the software system.

Whereas—by means of conventional techniques of modularization or object-orientation—one sort of concerns may smoothly be encapsulated within building blocks such as modules, classes, and operations on design or code level, the same is not possible for another sort of concerns. They crosscut the design or implementation of several or even many building blocks and are therefore called *crosscutting concerns*. Typical examples for crosscutting concerns include logging, synchronization and distribution. Due to their very nature, crosscutting concerns entail two main problems for software development. First, their design or implementation is scattered over many building blocks (the *scattering* problem) and second, one building block often comprises the design or implementation of more than one concern (the *tangling* problem). Scattering and tangling have a number of well-known negative implications for the software affected by them. However, aspect-oriented software development (AOSD) aims at alleviating these problems by modularizing crosscutting concerns. They are encapsulated by modular artifacts called *aspects* and thus banned to separate locations. AOSD is therefore an approach to further expanding the SoC principle in software development.

In addition, it does not suffice to pursue aspect-oriented approaches during design and downstream activities alone since crosscutting concerns are not confined to the artifacts dealt with there. Rather, it is beneficial to support aspect-orientation from the very beginning of the software lifecycle. To substantiate this claim the next section introduces the notion of ”crosscutting requirement”, a special kind of crosscutting concern, and section 3 explains then why it is desirable to identify and document them as well as their influences on other requirements. In section 4, some stages in software development and maintenance are suggested when identification and documentation activities are conceivable. Since we cannot expect that crosscutting requirements are always identified early, i.e. during requirements analysis, section 5 explains the usefulness of approaching the identification problem from an aspect-mining point of view, i.e. to search for crosscutting requirements in already existing requirements specifications, and suggests two techniques for accomplishing this task. Section 6 presents some preliminary results of a case study conducted to confirm the feasibility of this approach. Section 7 then discusses related work, and finally, section 8 describes conclusions and future work.

2 What are crosscutting requirements?

In the context of requirements engineering (RE) the concerns of central interest are *requirements*. A requirement is a special kind of concern. A well-written requirements specification is characterized by the fact that each requirements statement does not mix several requirements but represents exactly one requirement, i.e. exactly one concern. In a similar vein, for example Robertson and Robertson [12, p. 157f] recommend to express each requirement in one sentence only and to avoid complex sentences since they might contain more than one requirement.

In addition, good traceability—being a critical quality factor—is expected from well-written requirements specifications (cf. e.g. ANSI/IEEE Std 830-1984 [2]). Among others, it is an important precondition for making explicit the many semantic relationships that normally exist between different requirements. Examples of such relationships include interdependencies, elaborations and part/whole relationships (cf. e.g. [9]).

A special type of relationship among requirements is one that may be qualified as crosscutting. A requirement A crosscuts requirement B if a software decomposition has been chosen in which B cannot be satisfied without taking A into account. Thus design elements or code can only satisfy B if they provide for A as well. Requirements that crosscut others are referred to as being *crosscutting requirements* (A in this case). Further, the expression *crosscutting influence* is used as a synonym for the relationship between two requirements which is established by one crosscutting the other. If it is said that A has crosscutting influence *on* B it additionally denotes the relationship’s direction, namely that A crosscuts B.

A crosscutting influence is a certain dependency among requirements. However, not all requirements dependencies are of a crosscutting nature. To relate the notion of “crosscutting influence” to traditional types of dependencies, one of several existing taxonomies devised for the classification of dependency types has been used. It is Pohl’s [8] taxonomy. Altogether, it contains eighteen different dependency types of which there is only one, though, that seems to cover crosscutting influences, namely “constraint”. Pohl does not provide a definition of what exactly he understands by a constraint dependency type but gives an example that suggests the type’s basic meaning as one requirement imposing some restriction on another one. Since such restrictions always have an impact on the manner the affected requirement may be satisfied, the constraint notion fits well with the definition of “crosscutting” given earlier. The following three examples show cases where one requirement is constraining the other and which are therefore considered as crosscutting:

- The response-time requirement “all system responses shall occur within 10 seconds” constraints the interoperability requirement “the system shall be interoperable with the XYZ platform.”

- The functional requirement “when requested, the system shall display a user’s name, date of birth, and address” is constrained by the security requirement “only authenticated users are allowed to view personal data.”

- The functional requirement “when requested, the system shall download component updates from respective ftp mirror sites” is constrained by the functional requirement “if a connection to ftp mirror sites cannot be established within a specified timeout period the system shall cancel the download process.”

In all these examples, constraint dependency will eventually become visible in system behavior satisfying both requirements at the same time. This is a sure sign for this dependency type’s crosscutting nature.

An important characteristics of crosscutting influences is that they may be obvious but need not be. They are obvious if clear references to other requirements (for example as numbers or textual references) exist and thus make the respective interdependencies explicit. Crosscutting influences are more subtle if clear references do not exist and it can therefore not easily be concluded from reading a crosscutting requirement that it is crosscutting or, in case it is known that it is, on which other requirements it has influence. In these cases crosscutting influences are not (or not explicitly enough) documented on the requirements’ representation level (which is normally a natural language text possibly augmented by some semi-formal or formal definitions). For example, if a requirement demands the system's conformance to ISO 4217 we may infer its crosscutting nature either from background knowledge since we know that the requirement is constraining how other requirements may be implemented, or from such textual cues as "conform" or "ISO". However, our conclusion is not based on a clear reference. Furthermore, we cannot easily see on which requirements the requirement has influence since the individual dependencies are hidden behind a general statement. All we might deduce is that it is related to requirements having something to do with currency symbols.

Thus even if we do not find obvious tangling on the requirements' representation level there may, and in fact there does exist crosscutting influences beyond the representation level. These are dangerous since they may be overlooked in subsequent development stages. Moreover, they are a likely origin for tangling on the design or code levels. Such hidden influences are especially true for many non-functional requirements since these are normally stated in a global manner.

3 Why identify crosscutting requirements?

Ideally, a requirements specification would document all crosscutting influences as well as all other semantic relationships among requirements. Reality, however, is quite different. Even if each requirements statement does address only one concern and is traceable as well, not all semantic relationships between requirements are identified and explicitly recorded in most cases. The main reason for this is the fact that there normally is not enough time in a software project to transform a sub-optimal specification into an optimal one characterized by complete traceability. However, if some of the relationships among requirements (such as crosscutting influences) are obscure, developers run the risk of forgetting about them. At best they detect these influences in later development phases and are able to react accordingly. At worst some influences remain hidden until after delivery of the software system and are then indirectly discovered by users because some requirements are not totally fulfilled by the software. In both cases increased costs will be the result of the requirements deficiencies (the later the detection the higher the costs). The second case may in addition give rise to other troubles such as loss of good faith in the developers or even the disastrous failure of critical system components.

Hence, to stress the importance of identification and documentation of crosscutting requirements and influences we will briefly have a look at three exemplary requirements issues.

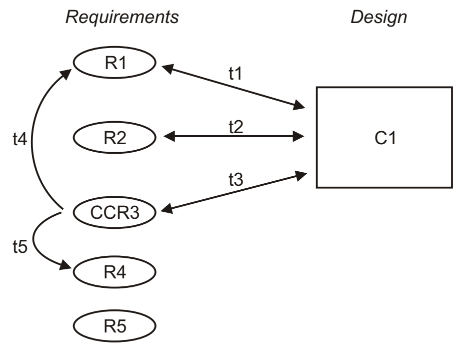

Figure 1. Assessing the impact of change

3.1 Impact of change

An example will illustrate how documentation of crosscutting requirements and influences makes it easier to assess the impact of change. On the left-hand side of Fig. 1 there are five requirements R1, R2, CCR3, R4, and R5. CCR3 is a crosscutting requirement. It crosscuts R1 and R4, which is denoted by the arrows t4 and t5, respectively. On the right-hand side there is a design component C1 satisfying the first three of the requirements, which is denoted by the three arrows t1, t2, and t3.

Now, C1 is to be altered. Thus it must be checked whether the first three requirements (and possibly any code which is not depicted in the figure) are affected by the change. This check is possible due to the traceability links t1 to t3. It then emerges that C1's change should be reflected in tightening CCR3. If the latter's crosscutting influences are documented (as depicted in the figure by t4 and t5), then it is easy to find the requirements that are impacted by CCR3, namely R1 and R4. It is then possible to find out what, if any, consequences a change of CCR3 will have for them. If, however, t4 and t5 are unknown an analyst must first check any other requirement (i.e. R1, R2, R4, and R5) whether or not it is affected by a change of CCR3. Such work is frustrating and involves unnecessary costs since thought processes have to be repeated that were already thought out previously while first modeling the requirements and designing C1.

3.2 Requirements evolution

As may be expected, being capable of assessing the impact of change among requirements in a sufficient manner is an important pre-requisite for requirements evolution. Thus comprehensive documentation of crosscutting requirements and influences also facilitates easier requirements evolution. To demonstrate this, an example will be used again.

A requirements specification might, among others, state the following requirements for some CAD-based system:

FR-1.1: The system shall generate raw parts from DIN 4000 norms.
FR-1.2: A system message shall confirm the performance of the operation described in FR-1.1.
PR-2: All system responses on user interactions shall occur within 30 seconds.

The first two requirements (FR-1.1 and FR-1.2) are functional whereas the third (PR-2) is a performance requirement. Although it can be said with confidence that each of these three requirements statements represents exactly one concern, there is the problem that PR-2 is a global requirement crosscutting all other requirements dealing with system responses (such as FR-1.2). Moreover, the crosscutting influences of PR-2 are obscure since it contains no clear respective references to the influenced requirements. (FR-1.2 is also crosscutting but contains a clear reference to the influenced requirement FR-1.1. Therefore, for the sake of brevity, we will not consider FR-1.2's crosscutting nature any further.) Furthermore, an inspection performed on the specification reveals that FR-1.2 is ambiguous. It is not clear when the message shall occur: at the beginning, in the middle, or at the end of the generation process? Therefore, a requirements engineer may want to clarify this and think about changing FR-1.2 in the following way:

FR-1.2: A system message shall confirm the completion of the operation described in FR-1.1.

Though this wording solves the ambiguity problem, it equally brings up an even greater problem. Since raw part generation is normally a computing-time intensive process, FR-1.2 now (very probably) conflicts with PR-2. Thus the planned hardware/software solution can (very probably) not satisfy both requirements at the same time. If the influences PR-2 has on FR-1.2 are not documented, this inconsistency may go undetected for a longer time and finally lead to the sort of troubles described earlier. If, however, PR-2's influence on FR-1.2 is documented, it is relatively easy for the requirements engineer to realize the potential conflict between them. Therefore, the engineer's final version of FR-1.2, being precise enough and no longer in conflict with PR-2, might read:

FR-1.2: A system message shall inform the user about the progress of the operation described in FR-1.1 from start to end.

3.3 No overlooked crosscutting influences

If done early enough, identification and documentation of crosscutting requirements and their impact on other requirements (which may, of course, be crosscutting themselves) brings about another important advantage: not to overlook them in subsequent phases. Provided that the crosscutting influences on a certain requirement are documented, it is easily possible to take them into account when design or code artifacts are derived from it or some corresponding maintenance task is performed. Otherwise, especially if the requirements specification is voluminous it is fairly easy for an architect, programmer or maintainer to unintentionally overlook some crosscutting influences and ignore them in their work.

Crosscutting influences that are documented also allow to manually, semi-automatically or—with a sufficient degree of formalization—even automatically derive corresponding test cases from the requirements influenced or from the crosscutting requirements themselves. It would for example be possible to semi-automatically generate performance test cases for requirement PR-2 (cf. section 3.2). In contrast, it is easily possible to forget to provide sufficient test cases for crosscutting requirements if crosscutting influences are not explicitly recorded. And this becomes even easier the closer a deadline approaches!

4 When to identify crosscutting requirements

There are several stages of the development and maintenance processes when identification and documentation of crosscutting requirements and influences are conceivable. These will briefly be discussed in turn:

- *while requirements modeling*: After requirements have been elicited they should be modeled. Modeling is a highly analytic activity whose goals are for example to structure the requirements (normally top-down), to identify and model dependencies and other relations among them, to identify and eliminate inconsistencies and to identify and clarify ambiguities and vagueness. It would of course also be natural and highly desirable to identify and model crosscutting requirements and influences already at this stage.

- *while writing the requirements specification*: Writing a specification normally follows requirements modeling. However, it is often the case that a specification is written directly after requirements have been elicited thus bypassing the modeling stage. In this case writing the specification is a highly analytic process as well. Obviously, it should then also include identifying and documenting crosscutting requirements and influences.

- *after writing the requirements specification*: Although it is best to identify and document crosscutting requirements and influences when they arise, i.e. during requirements analysis, we have to face the fact that countless requirements specifications exist, written without the aspect of crosscutting requirements in mind. With hindsight however, it may become desirable for developers to identify them in such documents. It then becomes necessary to "mine" (see section 5) crosscutting requirements and influences from them.

- *during downstream activities*: The crosscutting nature of some requirements and their influences may and will of course also be detected during activities later in development or maintenance and should then be documented.

5 Aspect mining from requirements specifications

From the stages mentioned in the previous section the third one, namely "after writing the requirements specification", is of central interest to us. The research work described here therefore focuses on the identification of crosscutting requirements and influences from an aspect-mining [4] point of view. Aspect mining is a term coined for the identification of aspects (which encapsulate crosscutting concerns) in some pre-existing requirements or design documents, code and other development artifacts. Aspects are normally not on the surface of these artifacts but must be "mined" from them. The artifacts we are interested in are requirements specifications.

Aspect mining from requirements specifications is reasonable since countless ones already exist, written without any consideration of crosscutting concerns. Furthermore, it seems to be realistic not to expect aspect-orientation in RE to become a widely-accepted approach in the near future in order so as to be applied from the early stages of software projects. Therefore many more documents with these shortcomings will be created in the future as well. For these reasons, aspect mining is already needed and—with the increasing dissemination of aspect-orientation in software industry—will even much more be needed to open the way to finding and documenting hidden crosscutting concerns in already existing specifications and thus enhance their quality and usefulness.

For the purpose of mining crosscutting requirements and influences from requirements specifications, two conceivable techniques will now be suggested, the first being manual, the second being semi-automatic.

- *identification through inspection*: After a requirements specification (or some part of it) has been written it should be inspected to identify and document any sorts of deficiencies (e.g. omissions, inconsistencies, typos, etc.). It would therefore be straightforward to read the specification with the particular aspect of identifying crosscutting requirements and influences in mind. From our own experience it is for example recommendable to start from a given non-functional (and probably crosscutting) requirement, say response time, and then search in the text for statements mentioning some system response. If such a statement has been found the inspector notes it down. After the inspection has been finished the collected statements must be analyzed whether they reflect requirements constrained (see section 2) by the response-time requirement. If so, a crosscutting influence has been located and will be documented. If at least one crosscutting influence has been found the response-time requirement is then marked as being a crosscutting requirement.

- *identification supported by information retrieval techniques*: It is also possible to support the identification of crosscutting requirements and influences by means of information retrieval (IR) techniques. A program may for example find statements that seem to be affected by a requirement the analyst assumes to be crosscutting. In a similar manner as above, the analyst must then examine each statement and document when a crosscutting influence has been found. Again, recording at least one crosscutting influence leads to the requirement's characterization as being crosscutting.

The IR-based technique is especially promising for voluminous documents. It may be used standalone if, for example, time is short for performing a full-blown inspection or in combination with an inspection to detect further crosscutting influences not identified so far. On the one hand, inspecting a substantial specification is hard work where many interesting details (e.g. crosscutting influences) may be overlooked because of lack of time, fatigue, or monotony. On the other hand, compared to an IR approach, inspection has its main strengths in dealing with the peculiarities of natural language such as ambiguity and vagueness, and may therefore far easier achieve completeness, i.e. detect all requirements influenced. However, although completeness is highly desirable even inspection cannot fully guarantee that completeness has been accomplished at some stage since the approach is based on subjective and retroactive judgments.[1] Both techniques also appear to be suitable for an agile approach to identifying crosscutting influences, i.e. to only search for influences when it is actually required by some development or maintenance task.

[1]Most traceability approaches—both manual and (semi-) automatic ones—must cope with a degree of uncertainty with respect to completeness. Further, accomplishing completeness it is not always necessary. E.g., in the example given in 3.2 it is not necessary to know of all the requirements influenced by PR-2. Instead, knowledge of its crosscutting influence on FR-1.2 is sufficient.

Property	Number
pages	134
complete use cases	35
incomplete use cases	5
complete scenarios	94
incomplete scenarios	9
preconditions, scenario steps, and postconditions	686
quality requirements	110
constraints	10

Table 1. General statistics of RS-Ecom (e-commerce application)

Property	Number
pages	52
higher-level functional requirements	46
higher-level non-functional requirements	12

Table 2. General statistics of RS-Rent (rental system)

quality requirement category	RS-Ecom	RS-Rent
accuracy	●	–
authentication	●	–
authorization	●	–
data recovery	–	●
documentation	–	●
identification	●	–
immunity	●	–
integrity	●	–
intrusion detection	●	–
latency	○	–
legislative	–	●
non-repudiation	●	–
portability	●	●
precision	●	–
privacy	○	–
response time	●	●
robustness	●	–
security	–	●
system maintenance security	●	–
timeliness	●	–
usability	○	●

Table 3. Crosscutting (CC) quality requirement categories in example RSs. (● = surely or most probably CC; ○ = potentially CC; – = irrelevant for the given RS)

6 Case study

Some examples of crosscutting requirements and influences which have been identified in two requirements specifications (RSs) shall be mentioned here to suggest that the proposed aspect-mining approach is feasible. The study mainly concentrated on exploring interactions between non-functional and functional requirements. The specifications studied by and large conform to general recommendations on how to write requirements specifications. The semi-automatic procedure which will be introduced in this section works for locating crosscutting influences of response-time requirements in these specifications and most probably in others of similar quality. However, currently it cannot be told definitely whether similar methods work for other non-functional requirements as well. This will be a matter of future research.

The first RS (RS-Ecom) studied is use-case based, comprises 134 pages and describes an e-commerce application that allows selling and buying any goods via the internet. The second one (RS-Rent) is of a more traditional structure (no use cases), comprises 52 pages, and describes a rental system allowing renting of arbitrary equipment. Tab. 1 and 2 provide some statistical detail on both documents.

6.1 Results

It is relatively easy to identify crosscutting non-functional requirements if they are mentioned in a separate section especially devoted to non-functional (or "quality") requirements. That is the case with the example documents. The section in RS-Ecom contains 40 different subsections each describing a certain category of quality requirement. Nine of the categories are either irrelevant for the e-commerce application or are open issues or are not (fully) defined yet, i.e. the respective section mostly contains a TBD ("to be determined"). Of the remaining 31 categories, there are 14 that are surely or most probably of crosscutting nature (e.g. response time, authentication, cf. Tab. 3), 3 that are potentially crosscutting (e.g. privacy, usability, cf. Tab. 3) and further 9 (e.g. reliability, auditability) that may be crosscutting but have so far not been studied sufficiently to arrive at a definite conclusion. The respective section in RS-Rent contains 8 subsections each describing one category of non-functional requirement. There are at least 7 of the categories with crosscutting character (cf. Tab. 3).

It is not as easy to identify crosscutting functionality not explicitly mentioned in a subsection. However, study of RS-Ecom revealed that there is for example some timeout concern affecting at least three use cases. In this case, the influence is even observable in the RS's contents.

Finding the requirements which are crosscut by other ones is possible by studying the RS with this aspect in mind. As part of the case study, the functional requirements affected by global response-time requirements in both RSs were to be identified. The response-time requirements read as follows:

RT1: All system responses shall occur within 30 seconds. (*RS-Ecom*)
RT2: The System shall respond to any given command in an average time of 2 seconds of a user request. (*RS-Rent*)

Analyzing crosscutting influences on functional requirements only, the search could be constrained to sections on functional requirements. Since both RSs are of different types (one is use-cased based, the other not) and therefore of at least partially different terminology it is uniformly referred to their most detailed elements as *requirement leaves* (RLs). If an RL is crosscut by some other requirement its parents (i.e. superior requirements elements) are of course affected as well. Whereas an RL relevant for the study is either a scenario step, a precondition, or a postcondition in RS-Ecom, it is either a requirements description, a precondition, a postcondition, or a "side effect" in RS-Rent.

Inspection as well as IR-based techniques were applied for finding RLs crosscut by response time in both specifications. Some preliminary experiments showed that the following simple semi-automatic IR-based procedure is relatively effective for this task:

1. Use regular expressions according to the patterns <`SYSTEM NAME`> <`WORD STARTING WITH LOW LETTER`> and <`SYSTEM NAME`> `shall` <`WORD STARTING WITH LOW LETTER`> to find all respective matchings (e.g. *"System responds"* or *"System shall generate"*).
2. Sort the matchings and remove all duplicates.
3. Manually remove all matchings regarded as irrelevant (i.e. those not related to response time).
4. Find all lines and their contexts by means of regular expressions formed from the remaining matchings.
5. From the results returned, manually find all the lines containing requirements statements affected by the response time requirement (e.g. *"... When requested by a user, the System shall generate a report containing ... "*) and sort out all irrelevant lines.

In steps 3 and 5 it may be necessary to consult the requirements text if a line or its context does not contain sufficient information for making the right decision. From our own experience, this was needed more often in step 3 than in step 5. Altogether, however, text consultation was only rarely necessary.

The results of a comparison between inspection and the preliminary semi-automatic procedure are summarized in Tab. 4. For RS-Ecom, inspection was first applied which identified 178 RLs affected by RT1 in about 161 minutes. Then by employing the above procedure 179 RLs were identified, however in only 61 minutes. A review of the IR-based results revealed that of the found RLs, 178 were identical to those identified by inspection and one had been wrongly classified as affected by RT1. Taking the inspection results as a baseline, the procedure thus yielded maximal recall at a precision of 99.4% and time savings of about 62%.

Since much information is repeated in different sections of RS-Rent, it was decided to study two sections separately from each other. This time starting with the IR-based procedure, 24 RLs influenced by RT2 were identified in the first section within 19 minutes. Then inspecting the section found 27 influenced RLs in 21 minutes. For the second section again starting with the procedure, 50 influenced RLs were found in 16 minutes. Then by means of inspection 52 RLs could be identified in 24 minutes. Analysis showed that all RLs found in both sections by the IR approach had been classified correctly. Again taking the inspection results as a baseline, the IR-based procedure thus yielded a recall of 88.9% and 96.2%, respectively, both times at a precision of 100%. However, only for the second section, it brought significant time savings of about 33.3%.

6.2 Discussion

The results of the case study, in which a relatively large and two smaller portions of requirements specifications were analyzed, give rise to the following remarks and preliminary insights.

Crosscutting requirements: The study confirms the observations of others that not only is it possible to identify crosscutting requirements in requirements specifications but also that these documents are "significantly aspectual" [14]. At least 17 of 31 (54.8%) non-functional requirements categories in RS-Ecom and 7 of 8 (87.5%) in RS-Rent contain clearly or potentially crosscutting requirements. Further, some other crosscutting functionality was found which, however, was not considered in detail.

Time savings: The application of an IR-based technique seems to be more efficient for large than for small specifications. For the large RS-Ecom specification part, it achieved time savings of 100 minutes, which is more than 62%. Time savings for the small specification parts of RS-Rent were significantly lower. It may be assumed that the reason for this result lies in the density of influenced RLs. Whereas in RS-Ecom there are 178 of 686 (25.9%) RLs affected by response time, in the 1st and 2nd sections of RS-Rent there are only 27 of 51 (52.9%) and 52 of 118 (44.1%) RLs affected, respectively. The examples seem to suggest a correlation between density of RLs and time savings–the lower the density, the higher the time savings. In specifications with low density of influenced RLs, an IR-based tool can obviously more intensively function as a filter, saving an analyst from time-consuming unnecessary reading and leading to more focused work. For specifications with high density of influ-

Document	# pages analyzed	# RLs identified			time needed		comparison of IRT to Insp		
		Insp	IRT	IRT errors	Insp	IRT	recall	precision	time savings
RS-Ecom	110	178	179	1	161 min	61 min	100.0%	99.4%	62.1%
RS-Rent: 1st section	8	27	24	0	21 min	19 min	88.9%	100.0%	9.5%
RS-Rent: 2nd section	21	52	50	0	24 min	16 min	96.2%	100.0%	33.3%

Table 4. Crosscutting influences of response time requirements in example RSs (Insp = Inspection; IRT = IR-based technique; RLs = requirement leaves).

enced RLs, however, inspection might be more appropriate.

Prior knowledge: The IR-based procedure's high time savings and recall for RS-Ecom may in part lie in the prior knowledge and understanding collected by inspecting the specification first. Thus it is probable that some decision during the semi-automatic procedure (to exclude something from reading or not, an RL is relevant or not, an RL is influenced or not) was made faster than during inspection and also more accurate than without prior knowledge. However, every analyst who has written a certain specification, possesses prior knowledge which will help with the identification of crosscutting influences even though he/she has not specifically paid attention to them before. Moreover, to get a better impression of how effective and efficient the IR-based procedure is, the author wanted to exclude the possibility of being influenced by prior inspection results while semi-automatically analyzing the RS-Rent sections. Thus the author applied the procedure before inspection and tried not to use additional knowledge won by inspection of the first section for the semi-automatic analysis of the second. Whereas in doing so, no maximum recall could be achieved (see reasons below), time could always be saved.

Errors during IR-based procedure: Altogether, six errors (one superfluous RL and five influenced RLs not detected) have been made during IR-based identification. Analysis has shown that four of the errors have been made because of some human mistake (e.g. decision too hasty, unintentional deletion of relevant RL). Such mistakes could have been avoided since enough information for making a right decision had been available. The other two errors are consequences of line-based searching, which can be avoided using a better (e.g. sentence-based) approach.

7 Related work

During the last three to five years there has been considerable work around the application of aspect-oriented approaches in RE. This has, for example, been embodied in the notion of "early aspects" (e.g. [11]), explicit modeling of concerns (of which some are crosscutting) [15] and the interaction of use cases and aspects (e.g. [5, 3]). Although the necessity of identifying requirements-level crosscutting concerns has been stated, only few attempts have been made so far as to how to systematically identify them (e.g. in [5, 10, 11]). There is also at least one other paper ([14]) that deals with the manual derivation of crosscutting concerns from requirements documents. An important particularity of all these approaches is the precondition that requirements are modeled from scratch. While modeling the requirements (as viewpoints, concerns, or use cases) crosscutting concerns are identified and modeled as well.

In contrast, our work focuses on the identification of crosscutting concerns and influences in already existing requirements documents and augmenting them (or corresponding models) a posteriori. Since it cannot be expected that aspect-oriented techniques are applied in every project right from the start and there is the heritage of countless specifications written without consideration of crosscutting concerns, it is useful to offer an approach that may be considered as some sort of aspect mining [4] from these documents. Its advantage is the chance to uncover and document hidden crosscutting concerns in them and thus enhance their quality and usefulness. Thus a retroactive introduction of aspect-oriented RE approaches into projects that have even left the requirements phase, becomes feasible. Of course, it is also possible to employ this approach in projects in which requirements models and specifications have so far only partially been completed. Further requirements (also crosscutting ones) may then additionally be modeled using a more direct approach like the ones mentioned in the previous paragraph.

An approach similar to our work is the Theme/Doc [1] approach. One of its major goals is to semi-automatically identify crosscutting behaviors in pre-existing requirements specifications. After an analyst has specified a list of key actions, which are based on important phrases from the requirements document, the Theme/Doc tool takes the list as input and lexically analyzes the document. The analysis results are then presented in a graphical view containing the actions and links between those actions that occur in the same requirements sentence. Since it is assumed that linked actions are an indication of crosscutting and tangling, it is then the analyst's task to determine which of these links really indicate crosscutting relationships.

The main difference between Theme/Doc and the IR-based procedure introduced in section 6.1 consists in different preparatory work needed for running semi-automatic analyses. Theme/Doc needs setting up a keyword list which is done by an analyst and only possible by fully reading a requirements document. In contrast, using the IR-based procedure an analyst does not need to read the whole document before starting the analysis. All his/her preparatory work consists in providing the system's name and corresponding placeholders (e.g. "system") as well as in possibly determining the document's sections which will be searched. During analysis, selecting relevant words also becomes necessary, but not until a certain number of irrelevant passages has been filtered out. Hence, it may be expected that time is saved in comparison to applying the Theme/Doc approach. However, if it is the analyst's goal to find all crosscutting relationships in a requirements document, the Theme/Doc approach is certainly to be preferred. If, instead, an analyst is only interested in quickly finding the functional requirements crosscut by some response-time requirements he/she might prefer the IR-based procedure.

Our approach can also be seen as strongly related and contributing to the emerging field of Requirements Interaction Management (RIM) which is defined as "the set of activities directed toward the discovery, management, and disposition of critical relationships among sets of requirements" [13]. One of the main goals of RIM is to "detect and resolve requirements conflict" [13]. Moreover, of course, the semi-automatic technique proposed, heavily draws on the field of information retrieval.

8 Conclusions and future work

Crosscutting requirements are a special kind of requirement which influence the way how other requirements are satisfied by design or code artifacts. Identifying and documenting them as early as possible is critical and useful since it improves traceability among requirements as well as among requirements and downstream artifacts, facilitates easier assessment of change impact, supports requirements evolution, enables the application of aspect-orientation from the very start of the software lifecycle, and prevents easy oversight of crosscutting influences during development and maintenance activities.

In this paper, the identification problem is approached from an aspect-mining point of view, i.e. to identify crosscutting requirements and influences in pre-existing requirements specifications. Hence, two techniques for mining the former from the latter have been suggested. Some preliminary results of a case study confirm the approach's feasibility as to the mining of crosscutting influences of response-time requirements. The paper also mentioned that identified elements need to be documented but did not elaborate on that. As similarly suggested by [10], it would for example be possible to use a matrix relating crosscutting requirements with influenced requirements.

Our future work will mainly concentrate on the application of information retrieval (IR) techniques to the identification of crosscutting requirements and influences. For this reason, more requirements specifications will be studied to manually identify (hopefully) all crosscutting influences contained in them. Hereafter these results will then be compared with the results obtained by an IR-based semi-automatic tool which is currently being developed. Thus it will become possible to assess the effectiveness and efficiency of this technique. It is also interesting to find out how efficient the other suggested identification technique, namely inspection, is compared to the IR-based one. From what may be supposed however, using inspection or a tool is no either/or question. Instead, they may be used in combination and will then probably yield the best results.

Acknowledgements

The author wishes to thank the anonymous reviewers for their helpful remarks on an earlier version of this paper.

References

[1] E. Baniassad and S. Clarke. Finding Aspects in Requirements with Theme/Doc. In *Proceedings of Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design, Lancaster, UK*, 22 Mar. 2004.

[2] M. Dorfman and R. Thayer, editors. *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press, Los Alamitos, California, 1990.

[3] I. Jacobson. The Case for Aspects. *Software Development*, pages 32–37, Oct. 2003.

[4] N. Loughran and A. Rashid. Mining Aspects. In *Proceedings of Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop, April 22, Enschede, Holland*, 2002.

[5] A. Moreira, J. Araújo, and I. Brito. Crosscutting Quality Attributes for Requirements Engineering. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 167–174. ACM Press, 2002.

[6] B. Nuseibeh. Crosscutting Requirements. In *Proceedings of the 3rd International Conference on Aspect-Oriented Software Development*, pages 3–4. ACM Press, 2004.

[7] D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[8] K. Pohl. *A Process Centered Requirements Engineering Environment*. PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät, RWTH Aachen, Aachen, Germany, 1995.

[9] B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, Jan. 2001.

[10] A. Rashid, A. Moreira, and J. Araújo. Modularisation and Composition of Aspectual Requirements. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, pages 11–20. ACM Press, 2003.

[11] A. Rashid, P. Sawyer, A. Moreira, and J. Araújo. Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In *Proceedings of IEEE Joint International Conference on Requirements Engineering (RE 2002)*, pages 199–202. IEEE Computer Society, 2002.

[12] S. Robertson and J. Robertson. *Mastering the Requirements Process*. ACM Press books. Addison-Wesley, Harlow et.al., 1999.

[13] W. N. Robinson, S. D. Pawlowski, and V. Volkov. Requirements Interaction Management. *ACM Computing Surveys*, 35(2):132–190, 2003.

[14] S. M. Sutton Jr. Concerns in a Requirements Model - A Small Case Study. In *Proceedings of Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop, March 17, Boston, USA*, 17 Mar. 2003.

[15] S. M. Sutton Jr and I. Rouvellou. Modeling of Software Concerns in Cosmos. In *Proceedings of the 1st International Conference on Aspect-Oriented Software Development*, pages 127–133. ACM Press, 2002.