Requirements Eng (2007) 12:103–120
DOI 10.1007/s00766-007-0045-1

ORIGINAL ARTICLE

Automated classification of non-functional requirements

Jane Cleland-Huang · Raffaella Settimi · Xuchang Zou · Peter Solc

Received: 3 November 2006 / Accepted: 22 February 2007 / Published online: 23 March 2007
© Springer-Verlag London Limited 2007

Abstract This paper describes a technique for automating the detection and classification of non-functional requirements related to properties such as security, performance, and usability. Early detection of non-functional requirements enables them to be incorporated into the initial architectural design instead of being refactored in at a later date. The approach is used to detect and classify stakeholders' quality concerns across requirements specifications containing scattered and non-categorized requirements, and also across freeform documents such as meeting minutes, interview notes, and memos. This paper first describes the classification algorithm and then evaluates its effectiveness through reporting a series of experiments based on 30 requirements specifications developed as term projects by MS students at DePaul University. A new and iterative approach is then introduced for training or retraining a classifier to detect and classify non-functional requirements (NFR) in datasets dissimilar to the initial training sets. This approach is evaluated against a large free-form requirements document obtained from Siemens Logistics and Automotive Organization. Although to the NFR classifier is unable to detect all of the NFRs, it is useful for supporting an analyst in the error-prone task of manually discovering NFRs, and furthermore can be used to quickly analyse large and complex documents in order to search for NFRs.

J. Cleland-Huang (✉) · R. Settimi · X. Zou · P. Solc
Center for Requirements Engineering,
School of Computer Science, Telecommunications,
and Information Systems, DePaul University,
243 S. Wabash Avenue, Chicago, IL 60604, USA
e-mail: jhuang@cs.depaul.edu

R. Settimi
e-mail: rsettimi@cs.depaul.edu

X. Zou
e-mail: xzou@cs.depaul.edu

P. Solc
e-mail: petersolc@hotmail.com

Keywords Non-functional requirements · Quality requirements · Classification · Early aspects

1 Introduction

Non-functional requirements (NFR) describe important constraints upon the development and behaviour of a software system. They specify a broad range of qualities such as security, performance, availability, extensibility, and portability. As these qualities play a critical role in driving architectural design [1] they should be considered and specified as early as possible during system analysis. Unfortunately NFRs are often discovered in an ad-hoc fashion relatively late in the development process. In a recent review we conducted of 15 publicly available software requirements specifications (SRS) [2], we found an almost universal lack of any requirements describing non-functional qualities. This suggests that developers may fail to appreciate the importance of specifying NFRs or may falsely assume them to be implicitly understood and agreed upon by all stakeholders.

Despite a lack of emphasis on NFRs, stakeholders' quality concerns are often collected as a by-product of the requirements elicitation process and documented across a range of artefacts including memos, interview notes, and meeting minutes. Resulting requirements specifications tend to be organized by functionality, with non-functional requirements scattered widely across multiple documents.

This can lead to a number of problems such as important conflicts going undetected, architectural solutions that fail to take into account critical quality constraints, and development of products that fall short of meeting the stakeholders' real needs. This paper introduces and describes an NFR-classifier, for retrieving and classifying NFRs scattered across both structured and unstructured documents [3]. Although the classifier is trained to detect and categorize general NFRs; the use of domain specific terminology, specific writing styles, or use of pre-defined standards across disparate projects and organizations, means that ongoing training is needed. The paper therefore describes the initial training phase and related experiments that were conducted, and then introduces an iterative approach for retraining the classifier so that it can be utilized across a broad spectrum of document types and domains.

The NFR-classifier can also be used to detect and classify early aspects. An early aspect is a concern that cuts across the dominant decomposition of a system [4] and is found in the requirements specification or other early design documents. Many early aspects are identical to high-level NFRs such as security, performance, portability, and usability. Intermediate level aspects include concerns such as logging and authentication, while lower-level concerns focus on program level concepts such as buffering and caching. The classification method described in this paper is applicable to the high and intermediate level concerns that are described in the requirements specification or other early documents. Lower level concerns that belong to the solution domain at the code or design level are not discussed further. Early discovery of aspects is significant not only for purposes of architectural design, but also so that candidate aspects can be evaluated and modelled in the design and code of the system, thereby minimizing the need to 'mine' and refactor aspects from the code at a later date.

The NFR-classifier uses information retrieval methods to find and identify NFRs. The method assumes that different types of NFR are characterized by the use of relatively distinct keywords that we call 'indicator terms'. When those indicator terms are learned for a specific NFR type, they can be used to detect requirements, sentences, or phrases, related to that type. The process, which is depicted in Fig. 1, includes the three primary phases of *training*, *classification*, and *application*. During the training phase indicator terms are mined from existing requirements specifications in which NFRs have been manually categorized by type. These terms are then used during the retrieval phase to detect and classify other NFRs. Finally during the application phase, the classified requirements are used to support more advanced software engineering activities such as requirements negotiation or architectural design. An additional phase, labelled in the diagram as *iterative training* augments the previously learned indicator terms through feedback obtained from the analyst as he or she reviews the classification results for a new document. Iterative training can also be used to train a classifier from scratch.

This paper extends our previous work presented at the International Requirements Engineering Conference [3]. Section 2 surveys existing methods for eliciting and discovering non-functional requirements and early aspects. This section also describes a preliminary experiment we conducted using a fixed key-term approach. Section 3 describes the NFR-classifier, and the process of mining terms and using them to classify NFRs. Section 4 presents an initial experiment we conducted that was previously reported in [3] to evaluate the effectiveness of the approach using 15 different requirements specifications constructed as term projects by MS students at DePaul University. In Sect. 5, we report additional results from extending the size of the dataset to include 30 projects. Section 6 then describes an industrial case study which was previously reported in [3] and is based on a large user requirement document developed for a project at Siemens Logistics and Automation plant. The classification results obtained from reusing previously mined indicator terms are compared to results obtained from retraining the tool. In Sect. 7 we introduce a new technique and associated algorithms for iteratively training (or re-training) the classifier when it is needed in an entirely new domain or organizational context. The iterative process is critical for use of the tool in an industrial setting. Section 8 then concludes with a discussion of the application of this approach to the requirements analysis process, and suggestions for future work. Finally, this paper corrects the precision results reported in our previous paper. This correction is discussed in greater detail in Sect. 4.2.

2 Existing NFR classification methods

There are two primary approaches that are currently used to classify NFRs. These include elicitation methods that support stakeholders as they reason about, identify, negotiate, and model NFRs; and also detection methods for semi-automated or manual extraction of NFRs from a variety of existing documents.

2.1 Elicitation techniques

Elicitation methods frequently rely upon creative brainstorming or the use of checklists and templates to trigger stakeholders' input. For example Win–Win methods [5] provide generic checklists and require stakeholders to contribute, prioritize, and negotiate requirements that are

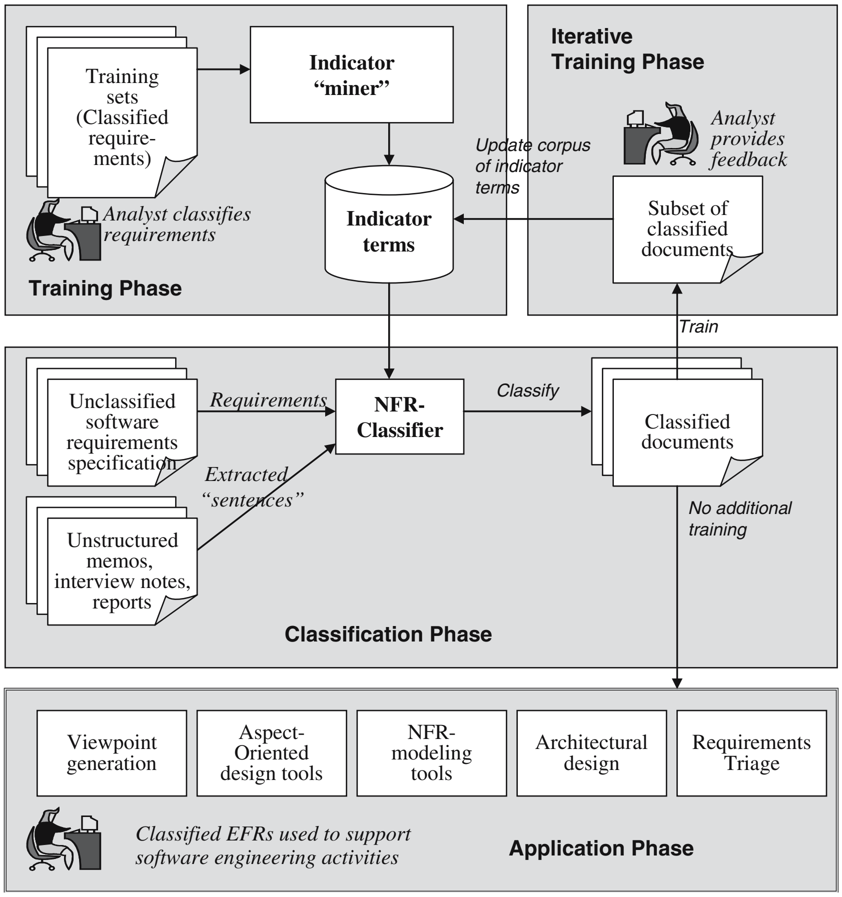

Training sets (Classified requirements)
Indicator "miner"
Indicator terms
Analyst classifies requirements
Training Phase
Iterative Training Phase
Analyst provides feedback
Update corpus of indicator terms
Subset of classified documents
Train
Unclassified software requirements specification
Requirements
NFR-Classifier
Classify
Classified documents
Extracted "sentences"
Unstructured memos, interview notes, reports
No additional training
Classification Phase
Viewpoint generation
Aspect-Oriented design tools
NFR-modeling tools
Architectural design
Requirements Triage
Classified EFRs used to support software engineering activities
Application Phase

Fig. 1 The process for classifying non-functional requirements

perceived to be important to the success of the system. Cysneiros et al. [6] proposed a technique for eliciting, analysing, and tracing NFRs based on the use of a language extended lexicon (LEL). Their approach required stakeholders to build a common vocabulary to support the concurrent specification of a functional and non-functional model. An NFR knowledge base was then used to determine if any quality concerns or constraints were related to any of the LEL symbols. This information was used to construct the NFR model. Dőrr et al. [7] proposed an NFR elicitation technique that first constructed functional use cases and then utilized a checklist to identify and associate NFRs to constrain the use cases. Kaiya et al. [8] proposed a similar use case approach but used the goal-question-metric (GQM) model to explore NFRs and their interdependencies.

In addition to NFR elicitation methods, several NFR modelling and analysis techniques have been proposed. The architectural assessment method (ATAM) models NFRs using utility trees in which stakeholders describe quality requirements within a hierarchical abstraction of high-level goals [9]. The NFR framework provides catalogues to help analysts define NFR quality goals, potential implementation solutions, and also to identify conflicts [10]. In addition to the NFR framework, other goal oriented techniques such as i* [11], and Kaos [12, 13] provide a notation and environment in which analysts can model non-functional requirements and evaluate their constraints and tradeoffs. These techniques provide a structured approach for brainstorming and documenting NFR needs and for producing a set of categorized NFRs, however there are no guarantees that important stakeholders' concerns will be considered and included in the goal models. The NFR-classifier can support the process of gathering non-functional concerns by helping to ensure that NFR related stakeholders' comments are not inadvertently ignored or missed during the elicitation process. Work towards a speech-based version of the NFR-classifier that would further augment this process is described in [14].

2.2 Detection techniques

With the increasing popularity of aspect-oriented programming (AOP), several researchers have developed techniques for detecting low-level aspects in the design and code and 'early-aspects' from requirements specifications [15]. At the program level, aspects can be 'mined' using clone detection techniques such as pattern matching against

Table 1 Keywords for security and performance requirements

NFR type	Keywords extracted from a SIG catalogue [9]	Recall	Precision	Specificity
Security	Confidentiality, integrity, completeness, accuracy, perturbation, virus, access, authorization, rule, validation, audit, biometrics, card, key, password, alarm, encryption, noise	0.702	0.315	0.773
Performance	Space, time, memory, storage, response, throughput, peak, mean, index, compress, uncompress, runtime, perform, execute, dynamic, offset, reduce, fixing, early, late	0.438	0.214	0.874

the abstract syntax tree, or through analyzing the system's meta-model [16]. Runtime methods such as the analysis of execution traces [17] can also be used at the code level. Unfortunately, none of these approaches are applicable to the detection of early aspects, including NFRs, which are less formally expressed and exist prior to the system becoming executable.

Several semi-automated techniques have been proposed for mining early aspects. Rosenhainer proposed a basic information retrieval (IR) method that required an analyst to manually search through the requirements looking for candidate aspects [18]. If a candidate aspect were found, then it was used as the basis of an IR style query to find related requirements. Several researchers have described effective methods for implementing such artefact based searches [19–23]. However Rosenhainer's approach is labour intensive as it requires manual inspection to discover a 'starting point' for the analysis, and it also depends on finding a good initial requirement that contains similar terms to other aspect-related requirements. Maarek et al. [24] described a similar approach for mining software repositories and finding appropriate reusable components. However this approach requires the analyst to provide an initial query, which is something that the NFR-classifier generates automatically through use of a training set.

The Theme/Doc method [25] also provides semi-automated support for early aspect mining. An analyst parses the requirements specification to identify keywords, which are then used by the tool to generate a visual representation of the relationships between behaviours. This view is used by the analyst to identify candidate aspects. Again, the approach is rather labour intensive as the analyst needs to perform a preliminary manual search for keywords in addition to a later analysis of the candidate concerns. It also presupposes that the requirements specification is grammatically structured in a certain way. However Theme/Doc provides an opportunity to discover aspect types that are unique to a specific project in addition to commonly occurring types.

Sampaio et al. proposed a method that uses natural language processing to first identify viewpoints based on nouns occurring in the specification and then to find actions (i.e. verbs) that occur across multiple viewpoints. This approach has potential for identifying unique aspect types, but it requires significant user feedback to evaluate viewpoints and assess the feasibility of the candidate aspects. It may also miss aspects if different action verbs are used to represent the same concern across different viewpoints [26].

Although the techniques for early aspect identification are applicable to the problem of NFR classification, they are significantly more labor intensive than the NFR-classifier, which requires initial training and is then able to automatically retrieve a candidate list of NFRs.

2.3 Keyword classification method

As a precursor to our work on the NFR-classifier we investigated whether a pre-defined fixed set of keywords could be used to classify each type of NFR. This simpler approach would avoid the need to develop and use a training set. A small experiment was conducted in which a set of keywords, listed in Table 1, were extracted from catalogues of operationalization methods for security and performance softgoal interdependency graphs (SIGs) [10]. These catalogues represent extensive bodies of knowledge related to goals and potential solutions for each of these NFRs and so provided a standardized set of keywords. The keywords were used to retrieve NFRs from a set of 15 requirements specifications developed by DePaul MS students as term projects for a course in Requirements Engineering. Approximately 80% of the students in this course work in the software industry as professionals. Out of the 45 students in the class the top 15 projects were selected for this experiment, based primarily upon grades assigned by the course instructor. Project topics included a wide range of domains including a claims dispute manager, vehicle parts finder, meetings scheduler, battleships game, and an enterprise level service bus for use by the public works department in a major US city. Requirements were specified using the Volere template (http://www.systemsguild.com/), which has specific subsections assigned for several NFR types, including those analysed in this paper. Classification of NFRs in the training set was therefore performed by the students themselves as part of the requirements writing process. Although this process introduces the possibility that the requirements writing was biased by examples used in class and in the textbook, these datasets provided the opportunity for evaluating and comparing classification methods that would have been very time consuming to establish from larger industrial datasets. However, this threat was not present in the industrial case studies reported at the end of this paper, that were designed to test the results obtained from the initial student-based study.

In the keyword retrieval method, any requirement containing one or more of these keywords was classified as a candidate security or performance requirement respectively. A multiple categorization approach was taken in which a requirement containing both a security keyword and a performance keyword was classified into both categories.

Results were evaluated for each NFR type using the standard metrics of recall, precision [27], and specificity [28] where recall measures the percentage of NFRs that were correctly retrieved and categorized and is defined as follows

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}.$$

Precision measures the total number of correctly retrieved NFRs in respect to the total number of retrieved NFRs and is defined as:

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}.$$

Finally, specificity in binary classification is defined as the proportion of true negatives to all negatives and is computed as

$$\text{specificity} = \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}}.$$

The results of this experiment are shown in Table 1 and indicate that the security keywords returned a recall of 70.2% and precision of 31.5%, while the performance keywords returned a recall of 43.8 and 21.4% precision. Observation showed that many of the security keywords were in fact shared by other types of NFRs, and that several of the target NFRs, especially performance ones, did not contain any of the keywords and so were not retrieved. One of the primary stumbling blocks of this approach (and the reason that we did not evaluate more NFRs using this method) was the difficulty of finding accepted and standardized catalogues for the other NFR types specified in our data sets.

3 The NFR-classifier

The NFR-classifier addresses this problem by using a training set to discover a set of weighted indicator terms for each NFR type. This approach means that the NFR-classifier is limited to recognizing and retrieving NFR types for which it has been trained. However this is not overly limiting. Although more than 150 NFR types and numerous lower level aspects have been documented [10], in practice a much smaller subset of common ones are generally of interest during the system design process. The approach also has several benefits over the standard keyword method. Indicator terms can be automatically learned from existing pre-categorized requirement specifications, and therefore customized for an organization in accordance with their own standard terminologies and policies.

The NFR-classifier method consists of two stages. During the first stage, a set of indicator terms is identified for each NFR category. This step assumes the existence of a set of correctly pre-classified requirements that can be used for training. The requirements in the training set are used to compute a probabilistic weight for each potential indicator term in respect to each NFR type. The weight measures how strongly an indicator term represents a requirement type. For example, terms such as ‘‘authenticate’’ and ‘‘access’’ that occur frequently in security requirements and infrequently in other types of requirements, represent strong indicator terms for security NFRs, while other terms such as ‘ensure’ that occur less frequently in security requirements or are found in several different requirement types, represent much weaker indicators.

Once indicator terms are mined and weighted, they can be used in a second step to classify additional requirements and statements. A probability value that represents the likelihood that the new requirement belongs to a certain NFR type is computed as a function of the occurrence of indicator terms of that type in the requirement. A requirement is then classified according to a certain NFR type if it contains several indicator terms representative of that type. Requirements receiving classification scores above a certain threshold for a given NFR type will be classified into that type, and all unclassified requirements will be assumed to be functional requirements. Because classification results can only be considered successful if a high percentage of the target NFRs are detected for a specific type, in all of the experiments described in this paper the threshold was established with the objective of achieving high recall results.

Prior to classification, the requirements must be pre-processed and reduced to a set of keywords [27]. The pre-processing step first eliminates all common ‘‘stop’’ words that do not provide any relevant information on the document’s lexical content (for example conjunctions and prepositions). The remaining words are then reduced to their stemmed form using Porter’s stemming algorithm, to eliminate plurals, past tenses, and other suffixes. The following sections more formally describe the two steps of mining and classification.

3.1 Indicator terms mining

Let Q be a given requirement type. Indicator terms of quality type Q are found by considering the set S_Q of all type Q NFRs in the training set. The cardinality of S_Q is defined as N_Q. Each term t is assigned a weight score $Pr_Q(t)$ that measures how well the term helps identify a requirement of quality type Q. The weight score $Pr_Q(t)$ corresponds to the probability that a particular term t identifies a document (i.e. an individual requirement) as belonging to a type Q based on the standard information retrieval assumption that terms indicating relevance for a certain NFR type must be present in the document to be classified. The frequency freq(d_Q,t) of occurrence of term t in document d_Q, is computed for each document in S_Q. The expression for the probability value is computed as follows:

$$Pr_Q(t) = \frac{1}{N_Q} \sum_{d_Q \in S_Q} \frac{\text{freq}(d_Q, t)}{|d_Q|} \cdot \frac{N_Q(t)}{N(t)} \cdot \frac{NP_Q(t)}{NP_Q} \tag{1}$$

The first factor $\frac{1}{N_Q}\sum_{d_Q \in S_Q} \frac{\text{freq}(d_Q,t)}{|d_Q|}$ in expression (1) represents the term frequency component that is standard in Information Retrieval [29] and shows that the weight score $Pr_Q(t)$ increases if term t occurs frequently in NFR documents of type Q. It is computed as the average term frequency of term t in type Q NFR documents d_Q rescaled by the documents size $|d_Q|$.

The remaining component of the expression in (1) measure inverse document frequency [29] and penalizes the weight score if the term occurs in several quality types. The second factor $\frac{N_Q(t)}{N(t)}$ is the percentage of Q type documents in S_Q containing t with respect to all requirements in the training set containing t, whose number is denoted by $N(t)$. This factor decreases if the indicator term t is used broadly throughout the requirements specification. If the term is only used in Q type requirements, it will evaluate to 1 for that type. The third factor $\frac{NP_Q(t)}{NP_Q}$ is the ratio between the number $NP_Q(t)$ of system projects containing type Q documents with term t and the number NP_Q of all projects in the training set with type Q NFRs. The purpose of this rescaling factor with values ranging between zero and one is to decrease the weight $Pr_Q(t)$ for terms that are project specific. It is equal to one only if a term appears in all the projects containing type Q documents.

A probability score $Pr_Q(t)$ is computed for each term t and terms are then ranked by decreasing order according to $Pr_Q(t)$. We considered two alternative methods to determine which terms should be used as indicator terms for each type Q. One method selected the top K terms as the indicator terms, and the second method selected all terms with a non-zero weight. Experiments to evaluate the two methods and select the best value for K are presented in Sect. 4.

3.2 NFR classification

The NFR classification algorithm is defined by computing a probability score $Pr_Q(R)$ that evaluates the probability that a certain NFR R belongs to type Q. This probability score depends on the lexical content of requirement R, as we assume that type Q NFRs are more likely to contain indicator terms for that type.

Let I_Q be the set of indicator terms for a quality type Q. We assume that the weighted indicator terms in I_Q are identified and their weights computed from a training set that contains correctly pre-categorized NFRs. The indicator terms are mined using the expression in (1).

The classification score that an unclassified requirement R belongs to a type Q is defined as follows:

$$Pr_Q(R) = \frac{\sum_{t \in R \cap I_Q} Pr_Q(t)}{\sum_{t \in I_Q} Pr_Q(t)} \tag{2}$$

The numerator is computed as the sum of the term weights of all type Q indicator terms that are contained in R, and the denominator is the sum of the term weights for all type Q indicator terms. The probabilistic classifier for a given type Q will assign higher score $\text{Pr}_Q(\text{R})$ to an NFR R that contains several strong indicator terms for Q. Results on the application of our classifier are reported below.

4 Evaluating the classifier model

The training set used in the experiments again consisted of the 15 requirements specifications developed as term projects by MS students at DePaul University. These specifications contained a total of 326 NFRs and 358 functional requirements. NFR types included availability, look-and-feel, legal, maintainability, operational, performance,

Table 2 Counts of requirement specification by project and quality type

Quality type	Project number															Total
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Availability (A)	1	1	1	0	2	1	0	5	1	1	1	1	1	1	1	**18**
Look and feel (LF)	1	2	0	1	3	2	0	6	0	7	2	2	4	3	2	**35**
Legal (L)	0	0	0	3	3	0	1	3	0	0	0	0	0	0	0	**10**
Maintainability (M)	0	0	0	0	0	3	0	2	1	0	1	3	2	2	2	**16**
Operational (O)	0	0	6	6	10	15	3	9	2	0	0	2	2	3	3	**61**
Performance (P)	2	3	1	2	4	1	2	17	4	4	1	5	0	1	1	**48**
Scalability (SC)	0	1	3	0	3	4	0	4	0	0	0	1	2	0	0	**18**
Security (SE)	1	3	6	6	7	5	2	15	0	1	3	3	2	2	2	**58**
Usability (US)	3	5	4	4	5	13	0	10	0	2	2	3	6	4	1	**62**
Total NFRs	8	15	21	21	37	44	8	71	8	15	10	20	19	16	12	326
Functional	*20*	*11*	*47*	*25*	*36*	*26*	*15*	*20*	*16*	*38*	*22*	*13*	*3*	*51*	*15*	*358*
Total	28	26	68	47	73	70	23	91	24	53	32	33	22	67	127	684

Table 3 Top 15 indicator terms learned from the training set

Rank	Availability	Legal	Look and feel	Maintainability	Operational	Performance	Scalability	Security	Usability
1	avail	compli	appear	updat	interfac	second	simultan	onli	us
2	achiev	regul	interfac	mainten	environ	respons	handl	access	easi
3	dai	standard	profession	releas	server	time	year	author	user
4	time	sarban	appeal	new	oper	longer	capabl	user	train
5	hour	oxlei	colour	chang	product	fast	support	inform	product
6	pm	php	look	dure	system	minut	expect	ensur	abl
7	year	pear	simul	promot	databas	take	concurr	data	understand
8	technic	legal	product	product	browser	process	abl	authent	successfulli
9	downtim	law	compli	addit	window	user	number	secur	intuit
10	long	estimat	scheme	everi	web	system	user	system	learn
11	system	regard	logo	budget	comput	let	launch	malici	system
12	product	complianc	sound	develop	applic	maximum	process	prevent	click
13	seven	rule	brand	season	us	complet	next	incorrect	minut
14	defect	requir	feel	integr	internet	flow	product	product	self
15	asid	izognmovi	sea	oper	abl	everi	connect	ar	explanatori

scalability, security, and usability. As there were insufficient portability and process requirements, these NFR types were not included in the study. Counts for each requirement type are displayed in Table 2. The high ratio of NFRs to functional requirements reflects the time limitations that inhibited the writing of a more complete set of requirements during the allocated time. Table 3 lists the top 15 indicator terms that were mined from each NFR type. Performance indicator terms are amongst the most intuitive, and include stemmed terms such as *second*, *respons*, *time*, *longer*, *fast*, *minut*, *take*, *process* etc. In certain cases, domain specific indicator terms, such as *izognmovi* (as the weakest term shown for legal), or *sea* (as the weakest indicator term for look-and-feel) appear amongst the 15 top ranked terms. In fact, each of these terms occurred strongly in a single dataset, and although the project component of the mining formula decreased the terms' importance, in each case they were still ranked as the 15th term. Nevertheless, as depicted in Fig. 2, the weighting assigned to these terms was relatively low, and so their impact on NFR detection was relatively insignificant.

Another interesting phenomenon is that several terms, such as *product*, appeared across multiple NFR types. In fact this term appeared as an indicator term in seven out of the ten NFR categories. In this particular case, the term was found repeatedly in several NFR types but very rarely in any of the functional requirements. For this particular experiment, the situation was caused by the tendency of the

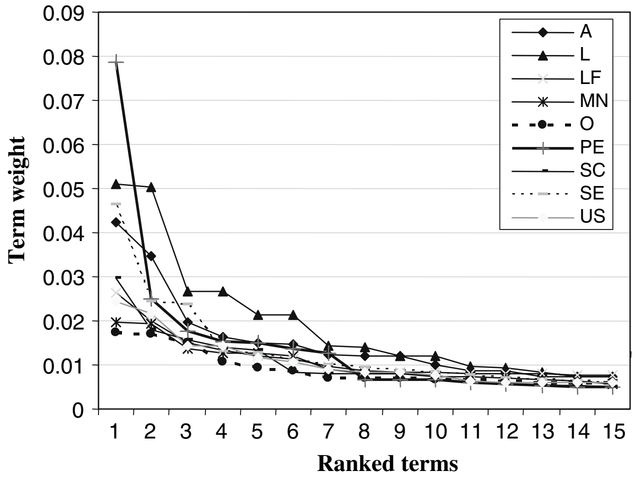


Fig. 2 Term weightings

students in the experiment to specify NFRs using phrases starting with ''The product shall.....'' The impact of writing styles and standards on the classifier is discussed more fully in Sect. 6.2.

4.1 Classifying the NFRs

To evaluate the effectiveness of the NFR-classifier, a leave-one-out cross validation technique was applied against the 15 SRS. Fifteen iterations were conducted. During each iteration indicator terms were extracted from 14 SRS's that constituted the training set, and term weights for each NFR type were calculated based on the function in (1). Two alternate methods were evaluated for selecting indicator terms from the training set:

1. **Top K terms**, where *K* is a positive number. For each NFR type, the *K* terms with the highest weights were selected as the indicator terms.
2. **All terms**, where every term with a non-zero weight with regard to a specific NFR type was selected as an indicator term for that type.

The extracted indicator terms were then used to classify requirements in the remaining SRS using the function in (2). A multiple classification scheme was used so that for any given NFR type, all requirements that scored higher than a certain threshold value were classified as that NFR type. This meant that a single requirement could be classified into more than one NFR type. Although normally a single requirement will only belong to one type of NFR, there are instances in which a requirement represents more than one NFR type. For example ''The railway gate must close at least 30 seconds before a train enters the crossing'' represents both a safety and a performance requirement.

As a side note, an additional experiment was conducted to compare the efficacy of multiple classification versus a ''pick-top'' method in which each requirement was assigned to only the NFR type for which it received the highest classification value. When the pick-top method was used, recall was problematic for almost all categories. For example, recall dropped to 33% for legal, 9% for look-and-feel, and 41% for maintainability. The average recall obtained using this method was only 52% as opposed to 76% using the multi-category method. For this particular problem (i.e. attempting to retrieve and classify all NFRs into their correct categories), recall is significantly more important than precision, because it is a much simpler task for an analyst to evaluate a set of candidate NFRs and reject the unwanted ones, than it is to browse through the entire document looking for entirely missed ones. For this reason the multiple-classification approach, which favours higher recall over precision, was adopted.

The goodness of the classifiers was evaluated by the three metrics of recall, precision and specificity for each NFR type in each single iteration of the experiment. Overall results were calculated by combining the results of the 15 iterations. As each SRS was subjected to classification in only one experiment, the combined results therefore represented the classification of each individual requirement only one time.

4.2 Selection of indicator terms

Three different values for *K* were used in our experiments: $K = 5$, $K = 10$ and $K = 15$. Table 4 shows the overall recall and precision of the classification using Top-5, Top-10, Top-15, and 'all' indicator terms respectively. The results in Tables 4 and 6 correct the precision values previously reported in [3], that were erroneously computed by excluding the functional requirements from the count of the classified requirements. A classification threshold of 0.04 was maintained for all four of the experiments, meaning that only requirements that were given scores greater than 0.04 for a particular NFR type were classified.

The results indicated that among the Top-*K* methods, Top-5 returned the worst recall, about 10% lower than the other two methods. The classification accuracy showed no significant difference between the Top-10 and Top-15 methods. Precision remained at about 14% while recall

Table 4 Comparisons of top-*k* versus ''All'' terms classification methods

Indicator selection method	Recall	Precision
Top-5	0.6564	0.1851
Top-10	0.7423	0.1372
Top-15	0.7669	0.1416
All	0.7392	0.1603

improved slightly from 74.23 to 76.69% when changing the selection method from using Top-10 to Top-15 terms. The results also indicated that if all terms were retained as indicators then the recall was reduced by 2.7%, while precision was increased by 2.4% compared to the Top-15 method. As recall and precision tend to trade-off against each other, this difference was again considered relatively insignificant. If the threshold had been slightly raised to remove some of the extra 'background noise' caused by the additional less significant terms, then recall would have likely decreased and precision increased to levels similar to Top-15. There was therefore no significant difference between the use of Top-10, Top-15, and ''all'' terms approach.

4.3 Classification results

A confusion matrix is a useful instrument for analyzing classification results [30], as it has the ability to depict true and false positives as well as true and false negatives. Table 5 illustrates the confusion matrix for the classification results for Top-15.

The correct classifications (true positives) are depicted on the diagonal, and have been highlighted in the diagram. For example, the matrix shows that 16 availability NFRs were correctly categorized. By looking in the column labelled ''A'' for availability we also see that eight maintainability requirements were incorrectly classified as availability. Additionally operational, performance, scalability, performance, and usability requirements were also similarly incorrectly classified. Looking across the row labelled ''Availability'' we also see that availability requirements were incorrectly classified under several NFR types including three as 'look-and-feel', and six as 'maintainability'. Although there are only 18 actual availability requirements, the multiple classification approach means that many of them will be classified more than once.

Table 5 Confusion matrix showing classification results

Actual	Total#	Classified as								
		A	L	LF	MN	O	PE	SC	SE	US
Availability	18	**16**	0	3	6	10	10	6	4	10
Legal	10	0	**7**	2	0	3	1	0	1	5
Look-and-feel	35	0	7	**18**	9	20	0	1	9	22
Maintainability	17	8	0	5	**15**	11	4	5	3	10
Operational	61	10	0	32	10	**44**	3	11	17	42
Performance	48	20	1	7	7	27	**30**	12	20	35
Scalability	18	11	0	5	4	12	3	**13**	6	11
Security	57	6	3	10	12	38	5	8	**46**	38
Usability	62	13	4	26	7	34	14	24	35	**61**
Functional	**359**	**60**	**21**	**46**	**68**	**188**	**40**	**37**	**109**	**189**

Table 6 Results using top-15 terms at classification threshold value of 0.04

EFR type	Recall	Precision	Specificity
Availability	0.8889	0.1111	0.7792
Legal	0.7000	0.1628	0.9525
Look-and-feel	0.5143	0.1169	0.6907
Maintainability	0.8824	0.1087	0.8220
Operational	0.7213	0.1137	0.4151
Performance	0.6250	0.2727	0.8561
Scalability	0.7222	0.1111	0.7825
Security	0.8070	0.1840	0.6468
Usability	0.9839	0.1442	0.3447

Table 6 reports the three metrics of recall, precision, and specificity that were derived from the confusion matrix for each NFR type. The precision values were erroneously computed in [3] by excluding the functional requirements from the count of the classified requirements. Table 6 reports the correct values. Here the specificity for a specific NFR type, say *availability*, is computed as the proportion of all non-*availability* NFR types that are not misclassified as *availability* to the total number of non-*availability* NFRs. A high specificity of an NFR type indicates the ability of the classifier to correctly differentiate this NFR from others.

As depicted in Table 6, different NFR types responded differently to the classification method. Usability had the highest recall of 98.39%, and the confusion matrix shows that only 1 out of the 62 usability NFRs were misclassified or unclassified. However, usability NFR also had the lowest specificity of 34.47%, meaning it was quite difficult to differentiate non-usability NFRs from usability ones. As shown in the last column of the confusion matrix, there are a total of 173 non-usability NFRs that have been misclassified as usability. In contrast look-and-feel achieves the lowest recall of 51.43% and a specificity of 69.07%.

In general these results suggest that the NFR-classifier can effectively detect several different types of NFRs, but that additional work is needed to improve results for certain NFR types such as 'look-and-feel.' It was observed for this NFR type, that categories of words such as colours tended to occur across multiple requirements, and future work will therefore investigate the possibility of using categories of indicator terms or extended training to improve these retrieval results.

4.4 Comparison to standard approaches

Following the publication of our initial paper on the NFR-classifier [3], we made the dataset containing the 15 student projects available to the PROMISE repository [31]. As part of Dr. Tim Menzies data mining course at the University of West Virginia, Jalaji et al. conducted an extensive set of experiments utilizing WEKA to compare standard classification techniques against the performance of the NFR-classifier algorithms. The results from naïve bayes classifier, standard decision tree algorithm (J48), feature subset selection (FSS), correlation-based feature subset selection (CFS), and various combinations of the above were evaluated [32]; however they were unable to globally improve on the results obtained by the NFR classifier. The combined approach with decision trees and feature subset selection procedure in Jalaili et al. produced the best recall and precision values among the standard machine learning tools used in the paper. Its recall value was still lower than the one achieved by the NFR classifier, while the precision values were similar. The complete report of their findings is available [32].

5 Training set size

To examine whether the number of NFRs in the training set affects the classification results positively, an experiment was conducted in which 15 new student term projects were added to the training set. Like the original projects, the new ones also utilized the Volere template and described a broad range of projects such as prescription management and online appointment scheduling. The number of NFRs contained in each project ranged from 8 to 28.

In order to compare results with those from the previous experiment, this experiment was designed to reclassify only the previous 15 projects. The original 15 projects were randomly divided into five groups of three. Each group in turn was used as the test set, while the remaining 27 projects (12 from the original set plus the 15 new ones) were used as the training set. In this way, after five iterations of the experiment, each of the previous projects was classified once and only once. The only significant difference in this experiment from the previous ones described in Sect. 4 was therefore the larger training set size.

The classification results, as displayed in Table 7, only partially supported our hypothesis that the classifier performs better with a larger training set. The recall for all NFRs increased only from 76.7% to 81.3% while precision slightly decreased to 12.4%. Although improvements were seen in availability, look-and-feel and operational requirements, other requirements such as scalability and security showed significant declines. The decreased recall observed in the scalability requirements was accounted for by the fact that several scalability requirements in two of the projects had previously gone unclassified because several critical indicator terms only appeared in two of the projects, and these project were both randomly selected into the same test set group. The classifier was therefore unable to learn the terms for classifying these two scalability requirements.

Table 7 Impact of training set size

NFR type	Training with 15 projects			Training with 30 projects		
	#	Recall	Precision	#	Recall	Precision
Availability	18	0.8889	0.1111	34	0.9444	10828
Legal	10	0.7000	0.1628	27	0.8	0.5333
Look and feel	35	0.5143	0.1169	62	0.7714	0.1004
Maintainability	17	0.8824	0.1087	48	0.9412	0.0510
Operational	61	0.7213	0.1137	93	0.8361	0.1275
Performance	48	0.6250	0.2727	68	0.6042	0.2589
Scalability	18	0.7222	0.1111	25	0.5556	0.0769
Security	57	0.8070	0.184	101	0.7895	0.1573
Usability	62	0.9839	0.1442	101	1	0.1387
Overall	326	**0.7669**	0.1416	559	**0.8129**	0.1244

6 Industrial case study

As an initial proof-of-concept the indicator terms mined from the 15 projects were used to detect and classify candidate NFRs from a Microsoft Word document describing the customer requirements for an integrated engineering toolset (IET) under development at Siemens Logistics and Automation plant. IET is a system for planning and constructing production lines, and represents a domain that is entirely unrelated to any of the 30 student projects. To the best of our knowledge, Siemens' employees who were involved in constructing the IET requirements had not previously been exposed to the Volere template (used by the students) and so were not biased towards writing requirements in a style similar to the student projects. The IET document was a well written document (scoring 10.3 on the Flesch-Kincaid reading level) organized entirely by functionality. It contained 137 pages, 2,250 paragraphs, and 30,374 words.

To classify the NFRs in the document, it was first saved as a text file, parsed to remove unwanted characters, and then deconstructed into 2,064 ''sentences.'' These sentences were not necessarily grammatically complete, as they included bullet points, and text extracted from tables etc. Some sentences corresponded to actual requirements in the text and others to less structured narrative. The data was treated to remove stop words and reduce terms to their stemmed forms, and was then parsed by the NFR classifier. In addition to automated classification, all of the sentences were manually classified into NFR types by the DePaul University faculty investigators in order to create an 'answer' set against which classification results could be compared. The manual classification took approximately 10 h of a single person's time. Validating the results took an additional 10 h. The counts for each NFR type are depicted in Table 8.

6.1 Fixed keywords

In the first experiment, the fixed keywords shown in Table 1 were used to classify security and performance NFRs. Security NFRs were retrieved with recall of 58%, precision of 9%, and specificity of 83%, while performance NFRs were retrieved with recall of 35%, precision of 6%, and specificity of 92%. These results strengthened our earlier conjecture that the use of this fixed set of keywords did not consistently produce good classification results.

6.2 Using prior indicator terms

In the second experiment, the terms extracted from the 15 original SRS's were used. Both ''Top-15'' and ''all'' indicator term approaches were evaluated, however no significant differences were observed in recall and precision metrics. Results from the ''all'' indicator approach which showed slightly higher recall, are shown in Table 8. Results using these previously mined indicator terms were relatively good for availability, security, and usability, which all had recall values around 80%, but were disappointing for several other NFR types. For example look-and-feel and performance NFRs were retrieved only at recall levels of approximately 33%, while legal, operational, and scalability requirements also returned relatively low recall values.

An analysis of the targeted NFRs in the user requirements document revealed a mismatch of terms between the MS Projects and the IET data. For example, performance NFRs in the high-level IET document tended to use more general terms such as ''fast'' and ''quickly'', compared to the more precise use of terms such as ''per second'' in the MS project training sets. Furthermore, different organizations follow different practices for structuring requirements, use of jargon, use of predefined templates, company and organization standards, and legal requirements; all of which may significantly impact the use of terminology within a requirements specification. This variation provides a feasible explanation for why indicator terms mined from the original training sets were not effective in classifying certain types of NFR in the new data set. It further underlines the importance of the NFR-classifier's ability to be trained to recognize NFRs and aspects within a specific organization or project.

6.3 Retraining the classifier

Because of the poor classification results from the first two experiments we decided to conduct a third study in which the NFR-classifier was retrained using a training set composed of one third of the sentences in the IET document. The training set was used to mine new indicator terms which were then used to classify NFRs in the remaining two thirds of the document. The fraction of ''one third'' was selected to provide sufficient NFRs in each type for training purposes. One third of the requirements from each requirement type, including functional requirements were randomly selected for the training set. Results obtained using the new indicator terms were generally much improved. All of the availability requirements were recalled; operational, security, and usability NFRs were recalled at relatively high values ranging from 73 to 87%; and only the NFR type of look-and-feel performed badly with a recall of 40%. In fact even this was a significant improvement from the previous recall value of 13%. There were insufficient

Table 8 Results from retrieving and classifying NFRs from IET requirements document

NFR type	Using ''all'' indicator terms mined from 15 MS projects				Using indicator terms mined from 30% of IET data			
	NFR count by type	Recall	Precision	Specificity	NFR count by type	Recall	Precision	Specificity
Availability	18	0.889	0.190	0.967	12	1.000	0.112	0.860
Legal	9	0.222	0.033	0.971	6	0.667	0.061	0.948
Look and feel	15	0.133	0.017	0.943	10	0.400	0.103	0.926
Maintainance	33	0.667	0.222	0.962	22	0.636	0.2	0.763
Operability	73	0.329	0.122	0.914	48	0.813	0.184	0.801
Performance	23	0.130	0.051	0.973	15	0.600	0.321	0.958
Scalability	2	0.5	0.008	0.939	2	Insufficient data to mine terms		
Security	29	0.828	0.071	0.847	19	0.737	0.21	0.885
Usability	183	0.803	0.262	0.779	122	0.877	0.277	0.500
All NFRs		**0.626**	**0.147**			**0.799**	**0.207**	

scalability requirements in the IET document to train the tool to recognize this type of NFR. Overall, recall rose from 62.6 to 79.9% and precision also increased from 14.7 to 20.7%. The results from this initial study supported the generally accepted IR wisdom that a training set that is ‘closer’ to the data being classified significantly improves the retrieval results.

7 An iterative approach

The previous experiment using the IET datasets has shown the importance of retraining the classifier before using it on datasets that are dissimilar from the ones on which it was initially trained. Unfortunately, training can be time consuming, and must therefore be designed to minimize human effort. This section therefore describes a new and iterative technique that sequentially creates a training set through user’s feedback on a subset of selected requirements. An existing corpus of indicator terms can be used as a starting point and augmented through additional training based on requirements selected from the new dataset. Alternately the process can be started from scratch without the benefit of an existing corpus. An iterative approach requires less human effort and enables the analyst to stop training when little additional improvement is predicted. When an existing corpus of terms is used, the amount of training is expected to primarily depend on the similarity between the new data sets and the initial training set.

The iterative approach builds a corpus of indicator terms in a sequential fashion. During each iteration a subset of requirements are displayed to an analyst who reviews each requirement and either accepts the classification result or reclassifies it into the correct category based on his or her expert judgment. The requirements that are manually evaluated and classified by the user are incrementally added to the training set, and the corpus of indicator terms is updated accordingly through the addition of new terms and the modification of existing term weights. This iterative step incorporates indicator terms from requirements that belong to the same domain and have been written within a similar context of practices and standards as the NFRs that are to be classified. The ability of the classifier to identify and categorize the NFRs is therefore expected to improve.

7.1 Selection criteria

The level of human effort required to train the classifier can be minimized by carefully selecting requirements to present to the user for feedback. These requirements should maximize the potential for discovering new indicator terms and strengthening existing ones as early as possible. Through a series of informal experiments and subjective observations, we determined that a significant number of previously unidentified indicator terms occurred in requirements that were ranked just below the classifier threshold values. Based on this observation, we decided that a large percentage of requirements presented to the analyst should consist of these ‘near misses’. This hypothesis was informally tested through including a random selection of unclassified requirements in earlier experiments; however this approach proved to be less efficient because fewer requirements resulted in useful feedback and training therefore took longer.

For experimental purposes the requirements selected for each iteration therefore included five randomly selected NFR type documents (as classified by the tool), plus 15 functional requirements (unclassified by the tool) selected among the top unclassified requirements based on the ranking of their classification scores. We also specified that a single requirement could be selected only once across all iterations of the training process.

In our experiments, the iterative learning approach was conducted for a fixed number of iterations to evaluate the performance of the classifier, and to study appropriate stopping criteria. In practice, the iterative learning approach should stop either when the user chooses to discontinue the training process, or when little improvement is anticipated through further training. One possible stopping criterion is determined through comparing recall values from two ‘sliding windows’, where a window consists of the recall classification results obtained from n iterations. At the end of a given iteration i, Window2 is composed of iterations $i-n+1$ to i, and Window1 is composed of iterations $i-2n+1$ to $i-n$. For example, if n is set to three, then following iteration 6, iterations 1–3 would be considered as Window1, and iterations 4–6 as Window2. A stop condition occurs when Recall(Window2) – Recall(Window1) < t, where t represents a predefined threshold value. Other possible stopping criteria involve using different measures for evaluating the classification process, for instance by considering the precision of functional requirements it is possible to indirectly measure the ability of the classifier to detect and classify the NFR types correctly.

7.2 Learning the indicator terms

The indicator term weights are updated according to the user’s feedback using the expressions reported below. Let D be an NFR that has been classified as quality type Q^* by the analyst.

For any term w in D, the probability $Pr_{Q^*}(w, i)$ at the ith iteration will be computed using equation (1) in Sect. 3.1. The change in the probability values due to the addition of the new NFR D to the training set can be evaluated by writing the expression of $Pr_{Q^*}(w, i)$ as follows:

$$Pr_{Q*}(w,i) = \frac{1}{N_{Q*}(i-1)+1} \times \left(\sum_{d \in S_{Q*}} \mathrm{freq}(d,w)/|d| + \mathrm{freq}(D,w)/|D| \right) \times \frac{N_{Q*}(w,i-1)+1}{N(w,i-1)+1} \frac{NP_{Q*}(w,i)}{NP_{Q*}(i)}$$

where S_{Q*} is the set of NFR documents of type $Q*$ and the following terms are computed at each i-th iteration from the training set:

- $N_Q(i)$ is the number of type Q NFRs
- $N_Q(w,i)$ is the number of type Q NFRs containing term w
- $N(w,i)$ is the number of requirements containing w
- $NP_Q(i)$ is the number of projects containing type Q NFRs
- $NP_Q(w,i)$ is the number of projects containing type Q NFRs with terms w.

In practice this formula simply represents an iterative version of the formula presented in Sect. 3.1. Notice that the number of projects $NP_Q(i)$ containing type Q documents is increased by one only if the document added to the training set belongs to a type Q that was not previously discovered in that project.

An analysis of the expression above shows that if we add a document D of type $Q*$ to the training set, the weight value $Pr_{Q*}(w, i)$ of each indicator term w in D will increase, and the increment amount will be larger if the training set contains fewer type $Q*$ documents, and/or if the term w belongs to fewer than average documents. As expected, adding a document that either belongs to a quality type that is underrepresented in the training set and/or that contains rarer terms will have a stronger impact on the corpus of indicator terms learned from the training set.

It is also interesting to observe the effect of adding a document D of type $Q*$ on the weight values $Pr_{Q\wedge}(w,i)$ for any quality type $Q\wedge$ different from $Q*$. The probability $Pr_{Q\wedge}(w,i)$ of the indicator term w in D is computed as follows:

$$Pr_{Q^\wedge}(w,i) = \frac{N(w,i-1)}{N(w,i-1)+1} Pr_{Q^\wedge}(w,i-1)$$

where $N(w,i-1)$ is the number of requirements containing the term w at the $(i-1)$th step. The probability of the indicator term $Pr_{Q\wedge}(w,i)$ decreases with a factor equal to $1/(N(w,i-1)+1)$, which is inversely proportional to the number of documents containing term w. Thus the impact on the indicator term weights will be stronger if w is a rare term in respect to the training set.

7.3 Simulation results

Two simulation experiments were conducted to test the effectiveness of the iterative learning approach for the Siemens IET dataset. In Experiment 1 the indicator terms previously learned from the original 15 student projects were used as an initial corpus of terms, and in the second experiment, Experiment 2, the training started with no initial corpus of terms. Results from the two experiments allowed us to evaluate the effectiveness of using an initial corpus of terms to classify a new dataset.

In both experiments, the collection of requirements was divided into a learning set and a testing set. The documents for the learning and the testing set were randomly selected using a stratified sampling approach to maintain the original size ratio of the different NFR types in both sets. The learning set was constructed by selecting 60% of the documents from each requirement type, and the testing set contained the remaining 40% of the documents. The learning set was used to sequentially construct the training set for the classifier while the testing set was used to measure the performance of the NFR classifier during the incremental training procedure.

A total of ten runs were executed for each simulation experiment and a learning set and a testing set were created at each run. For each run, the sequential learning approach was iterated 30 times by randomly selecting five requirements classified as NFRs and 15 requirements classified as functional (currently unclassified) from the learning set. As described above, the 15 currently unclassified requirements were selected from the top 15 unclassified documents in the learning set, ranked accordingly to their classification scores. For the first experiment, all of the indicator terms learned during the training phase of the 15 students projects were used to form an initial corpus of indicator terms. In Experiment 2, as no initial indicator terms were available to classify the learning set during iteration one, 20 requirements were randomly selected from the unclassified learning set. For both experiments the classification threshold was fixed at 0.04, and user feedback was simulated through the use of the previously defined ''answer set''. During each iteration, the selected requirements classified using the answer set to simulate user's feedback were added to the training set and used to update the corpus of indicator terms. The new corpus was then used to reclassify the remaining portions of the learning set, and to classify the documents in the testing set. After a certain number of iterations, the classifier was unable to detect additional NFRs in the testing set and no further improvement in recall and precision values were achieved.

At the end of the experiments, the performance of the classifier was measured by computing the average recall and precision values for the documents in the testing set

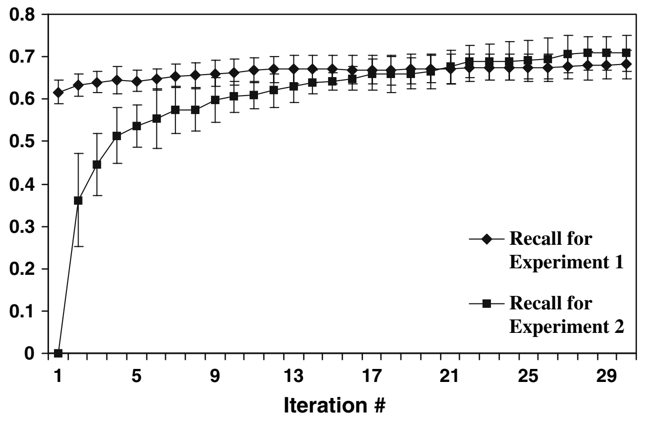


Fig. 3 Improvement in overall recall using the iterative tool. Experiment 1 uses initial corpus of indicator terms. Experiment 2 starts with no prior knowledge

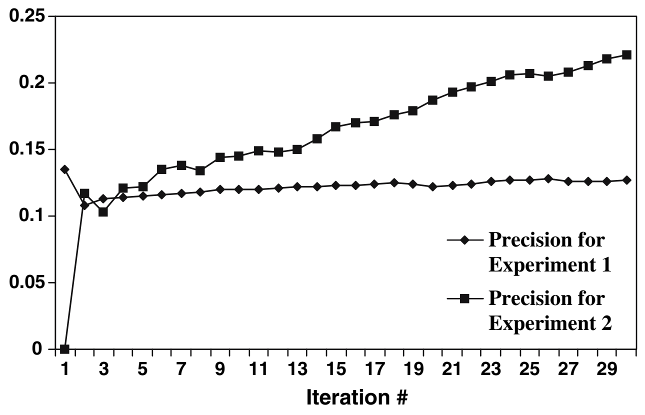


Fig. 4 Improvement in overall precision using the iterative tool for Experiments 1 and 2

based on the ten simulation runs. Figure 3 shows the average recall values and their error bars computed as two times their standard deviation for each iterative step. For Experiment 1, recall improved gradually until the 20th iteration, with an increase from the initial 61.6 to 68.2% after 30 iterations. In Experiment 1, the Availability, Security, Usability and Maintenance requirements had consistently high recall values of over 70% as shown in Figs. 4 and 5, while the most significant increase in recall due to the sequential training approach was seen for the look-and-feel and operational requirements. The recall of legal requirements worsened following initial training and only gradually improved over the remaining iterations. The improvement in recall for look-and-feel and operational requirements suggests that many quality type indicator terms missing in the initial corpus were sequentially learned during the iterative steps.

For Experiment 2, in which no prior knowledge of indicator terms was assumed, the improvement in recall was significantly faster during the early iterations, starting from 36.1% recall after the first iteration, and ending at 70.8% recall after 30 iterations. Iterative training was very effective for learning the indicator terms for most quality types as shown by the consistent upward trend in recall values displayed in Fig. 6 for the first 15–20 iterations. Important indicator terms were discovered at early iterations, and produced high average recall values for several NFR types, such as Availability and Operational.

The initial corpus of indicator terms was very useful and effective in discovering and classifying Maintenance and Security requirements. For these two types, the recall values of the classifier used in Experiment 1 were consistently higher than the recall values for Experiment 2 as displayed in Fig. 7. However the classifier using no prior indicator terms performed significantly better in identifying and categorizing Performance, Operational and Legal NFRs, and the sequential training with no prior knowledge was more effective. The overall performance of the two classifiers was very similar for Availability, Usability and Look & Feel requirements indicating that the initial corpus of indicator terms contained terms that were appropriate to discover and classify documents of those types even though the domain was different. For both experiments, increase in recall values seems to level off after 20 iterations.

A very interesting result is that precision in Experiment 1 showed no significant change, while the precision for Experiment 2 improved consistently during each iteration, as shown in Fig. 4. For Experiment 2, the precision values increased from 11.7 to 22.1% after 30 iterations. The classifier with no prior indicator terms therefore yielded better recall and better precision than the classifier with the initial corpus of indicator terms. The significantly higher precision values for Experiment 2 show that the choice of the initial corpus of indicator terms can have a significant effect on the classifier's performance. In particular it suggests that a disparate corpus of terms is a source of possible classification error, as it contains terms from different domains or specific specification styles that may be used differently in the new dataset. However if no sequential training can be conducted, an initial corpus of terms can be very helpful to train the classifier as suggested by the results of Experiment 1 and the results of our previous experiments in Sect. 6.

7.4 Analysis of indicator terms

A detailed analysis of the indicator terms for each NFR type learned from the two experiments at the end of the 30th iteration showed some interesting results. The set of indicator terms for each experiment were constructed through combining the indicator terms learned from all ten simulation runs for the Experiments 1 and 2, respectively.

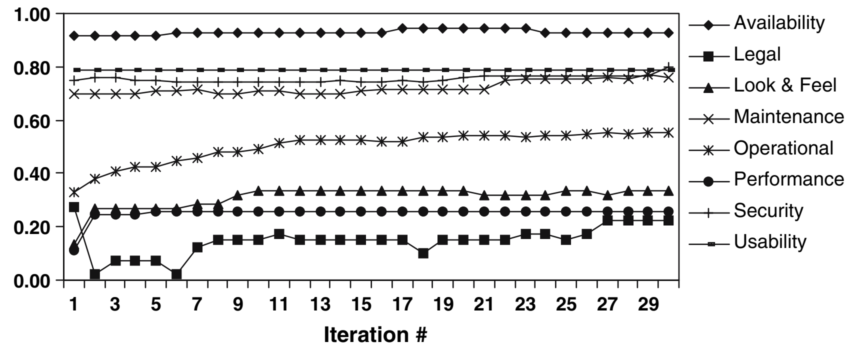


Fig. 5 Average recall for NFRs in the testing set for Experiment 1, utilizing an existing corpus of indicator terms

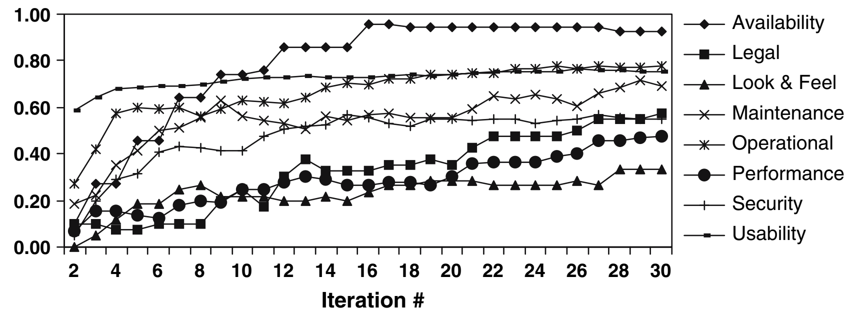


Fig. 6 Average recall for NFRs in the testing set for Experiment 2 based on no prior corpus of indicator terms

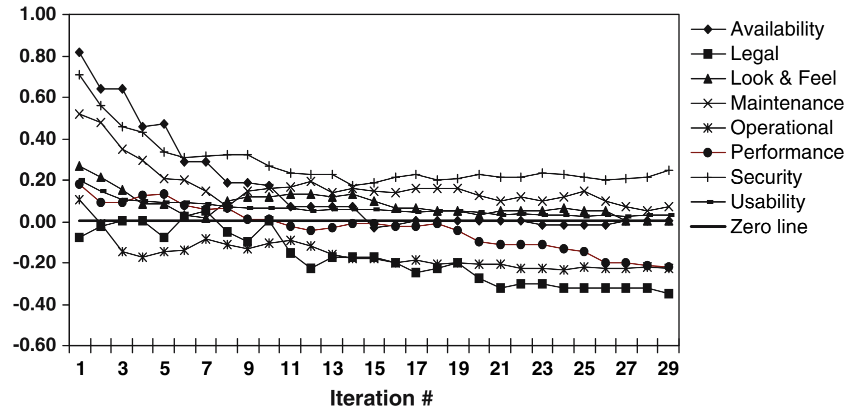


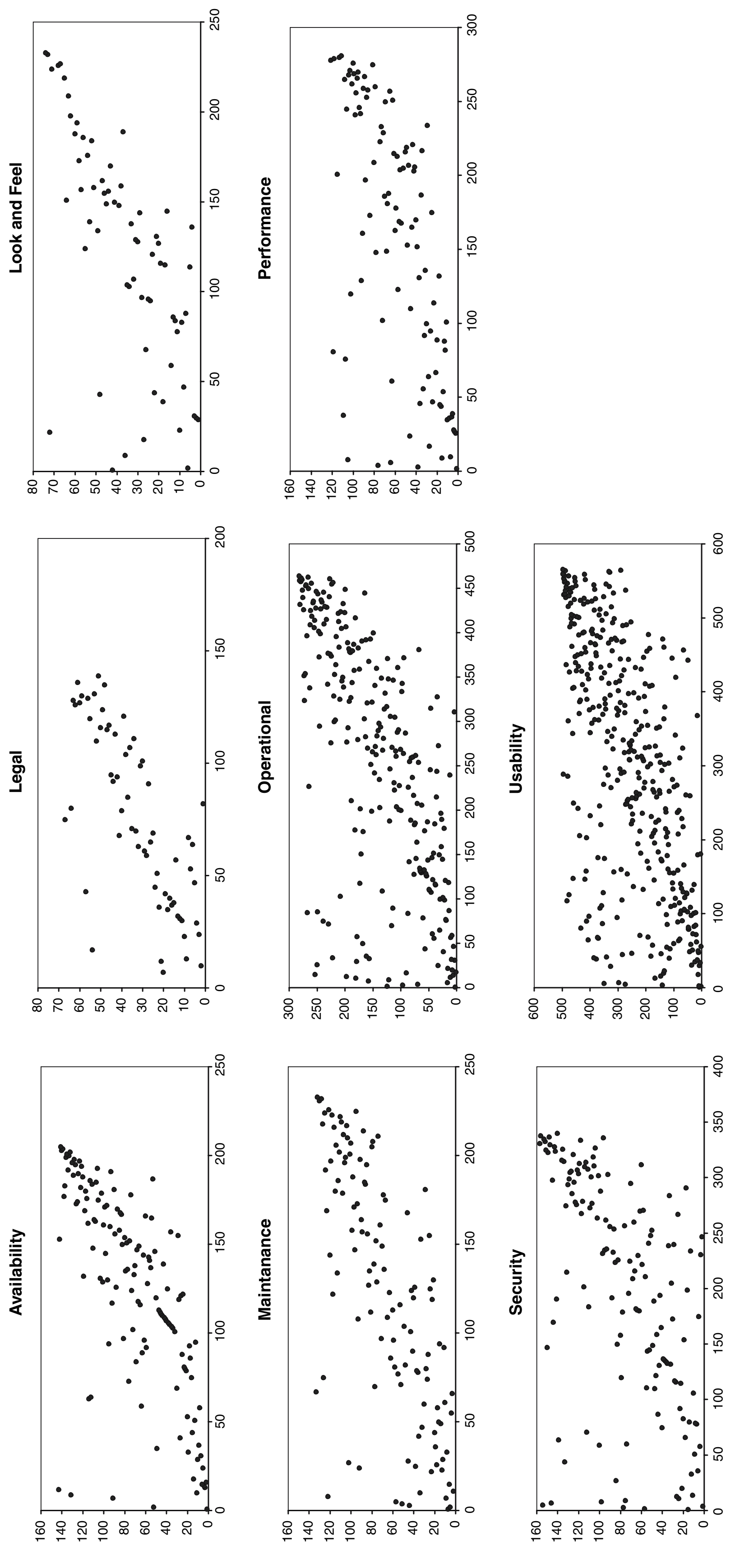
Fig. 7 Difference in average recall values between Experiments 1 and 2 for each NFR type

Each indicator term in the two sets was assigned a weight score obtained as the weight score average in the ten runs.

Our results showed that the top indicator terms for NFR types learned from the two experiments varied significantly. The scatter plots in Fig. 8 display the difference in ranking for the sets of indicator terms that appear in both the training set for Experiment 1 built using an initial corpus of indicator terms and the one for Experiment 2 assuming no prior knowledge. Several terms that are highly ranked in Experiment 1 were assigned significantly lower weights in Experiment 2. Those are the terms that appear more often in the initial corpus of indicator terms but that are not representative of the content of the NFR documents in the dataset to be classified. For instance the term ''system'' that is ranked fifth as indicator term for Performance and third for Operational in Experiment 1 has a much lower rank in Experiment 2. It is only 64th for Performance and 96th for Operational. For Performance requirements, the terms ''response'' and ''time'' among the top ten in Experiment 1, are ranked only 38th and 76th by the second classifier.

We also noticed that on average the top ranked terms for the classifier in Experiment 2, were also ranked relatively high in Experiment 1. This is consistent with the fact that highly ranked indicator terms in Experiment 2 are representative of the NFRs content in the project to be classified. Therefore they are likely to be learned during the sequential training process and used to augment the training set in Experiment 1. The weight scores for these terms are expected to still be reasonably high.

Fig. 8 Indicator terms ranking according to the weight scores for each NFR type for Experiments 1 and 2



8 Applications of the NFR-classifier

This paper has introduced a new approach based on information retrieval methods for detecting and classifying non-functional requirements from both structured requirements specifications as well as from free-form text. Although the approach still requires an analyst to evaluate the correctness of candidate NFRs, it requires less effort than previous semi-automated classification methods such as the Theme/Doc method [25]. Despite the fact that our results have suggest that perfect recall is probably unachievable and that 60–90% is a more realistic goal, the NFR-classifier is useful in several different ways.

First, the NFR-classifier can be used to support an analyst in manually detecting and classifying NFRs from a previously uncategorized requirements specification. The NFR-classifier could be used to augment the manual evaluation, which like all human activities has a tendency to be error prone. Secondly, the strength of the NFR-classifier lies in its ability to quickly trawl through large free-form datasets searching for stakeholders' comments and requirements. For example, during the elicitation process, requirements analysts may generate large amounts of unstructured documents in the form of meeting minutes, survey responses, interview notes, and memos etc. The NFR-classifier can parse these documents and extract viewpoints for different NFR qualities of interest. For example, a security analyst could issue a query for all comments and requests related to security issues, or a GUI designer could issue a request to retrieve information about stakeholders' usability or look-and-feel concerns.

Although the recall of the NFR-classifier is typically less than perfect (60–90% in general), it can still retrieve interesting and useful information that might otherwise be entirely overlooked during the requirements analysis process. Furthermore, because the classifier trawls for NFRs from raw stakeholders' data, there is a reasonable likelihood that many of the concerns will have been expressed by more than one stakeholder using different terminology. The NFR-classifier may therefore have more than one opportunity to detect a single concern. The NFR-classifier is useful for supporting many of the NFR elicitation and analysis techniques described in the introduction to this paper. For example the NFR-framework assumes that quality related goals for the system are known and understood, while other techniques such as Cysneiro et al's approach assume that the non-functional goals related to functional areas of the system can be brainstormed in a systematic fashion [6]. Our approach retrieves this information from the data collected during a broader elicitation process, and makes it available during the process of NFR modelling.

This paper has provided an initial validation of the approach and has also introduced a new technique for iteratively training a classifier to work in entirely new domains. Additional work is still needed to compare various potential stopping conditions; to optimize the value of the analysts' efforts by more strategically selecting and presenting statements for review; and finally to improve runtime performance of the tool so that it can be strategically used by a real analyst.

Acknowledgments The work described in this paper was partially funded by NSF grants CCR-0306303 and CCR-0447594. Additional funding and access to project artefacts was provided by Siemens Corporate Research and Siemens Logistics and Automation plant in Grand Rapids, MI. We would like to express our appreciation to Wen Dou for constructing the iterative prototype tool.

References

1. Nuseibeh B (2001) Weaving together requirements and architecture. IEEE Comput 34(3):115–117
2. Zowghi D Resources: collection of software requirements specifications [Online document]. Available at http://www.research.-it.uts.edu.au/re/cgi-bin/resources_srs.cgi
3. Cleland-Huang J, Settimi R, Zou X, Solc P (2006) The detection and classification of non-functional requirements with application to early aspects. In: IEEE international conference on requirements engineering, Minneapolis, MN, pp 39–48
4. Baniassan E, Clements P, Araujo J, Moreira A, Rashid A, Tekinerdogan B (2006) Discovering early aspects. IEEE Softw 23(1):61–70
5. In H, Boehm BW (2001) Using winwin quality requirements management tools: a case study. Ann Softw Eng 11(1):141–174
6. Cysneiros LM, do Prado Leite JCS, de Melo Sabat Neto J (2001) A framework for integrating non-functional requirements into conceptual models. Requirement Eng 6(2):97–115
7. Dőrr J, Kerkow D, Von Knethen A, Paech B (2003) Eliciting efficiency requirements with use cases. In: Ninth international workshop on requirements engineering: foundation for software quality. In conjunction with CAiSE'03
8. Kaiya H, Osada A, Kaijiri K (2004) Identifying stakeholders and their preferences about NFR by comparing use case diagrams of several existing systems. In: Proceedings of 12th IEEE international requirements engineering conference, pp 112–121
9. Kazman R, Klein M, Clements P (2000) ATAM: method for architecture evaluation. CMU/SEI Technical Report, CMU/SEI-2000-TR-004, ADA382629, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA
10. Chung L, Nixon B, Yu E, Mylopoulos J (2000) Non-functional requirements in software engineering. Kluwer, Boston
11. Eric SK Yu, John Mylopoulos (1994) Understanding ''Why'' in software process modelling, analysis, and design. In: Proceedings of 16th international conference on software engineering, pp 159–168
12. van Lamsweerde A (2004) Goal-oriented requirements engineering: a roundtrip from research to practice. In: Proceedings of RE'04, 12th IEEE joint international requirements engineering conference, Kyoto, 4–8 September 2004 (Invited Keynote Paper)
13. Dardenne A, Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program
14. Steele A, Arnold J, Cleland-Huang J (2006) Speech detection of stakeholders' non-functional requirements. In: International

workshop on multimedia requirements engineering—beyond mere descriptions, MERE'06, In conjunction with IEEE RE'06, September, 2006, pp 3

15. Kellens A, Mens K (2005) A survey of aspect mining tools and techniques. INGI Technical Report, 2005–08, UCL, Belgium, Deliverable 6.2a for the workpackage 6 of the IWT project 040116 ''AspectLab''
16. Bruntink M, van Deursen A, Tourwe T, van Engelen R (2004) An evaluation of clone detection techniques for identifying cross-cutting concerns. In: 20th International conference on software maintenance (ICSM'04), pp 200–209
17. Tonella P, Ceccato M (2004) Aspect mining through the formal concept analysis of execution traces. In: 11th working conference on reverse engineering (WCRE'04), pp 112–121
18. Rosenhainer L (2004) Identifying crosscutting concerns in requirements specifications. In: Workshop on early aspects: aspect-oriented requirements engineering and architecture design, Vancouver, Canada, 2004. Available at http://www.trese.cs.utwente.nl/Docs/workshops/oopsla-early-aspects-2004/
19. Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E (2002) Recovering traceability links between code and documentation. IEEE Trans Softw Eng 28(10):970–983
20. Cleland-Huang J, Settimi R, Duan C, Zou X (2005) Utilizing supporting evidence to improve dynamic requirements traceability. In: International requirements engineering conference, Paris, France, pp 135–144
21. Cleland-Huang J, Settimi R, BenKhadra O, Berezhanskaya E, Christina S (2005) Goal-centric traceability for managing non-functional requirements. In: International conference on software engineering, pp 362–371
22. Huffman Hayes J, Dekhtyar A, Sundaram SK (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans Softw Eng 32(1):4–19
23. Settimi R, Cleland-Huang J, BenKhadra O, Mody J, Lukasik W, DePalma C (2004) Supporting change in evolving software systems through dynamic traces to UML. In: IEEE international workshop on principles of software evolution, pp 49–54
24. Maarek Y, Berry DM, Kaiser GE (1994) GURU: information retrieval for reuse. In: Hall P (ed) Landmark contributions in software reuse and reverse engineering. Unicom Seminars Ltd
25. Clarke S, Baniassad E (2005) Aspect-oriented analysis and design. In: The theme approach. Addison Wesley Ltd. ISBN: 0321246748
26. Sampaio A, Loughran N, Rashid A, Rayson P (2005) Mining aspects in requirements. In: Workshop on early aspects
27. Frakes WB, Baeza-Yates R (1992) Information retrieval: data structures and algorithms. Prentice-Hall, Englewood Cliffs
28. Altman DG, Bland JM (1994) Statistics notes: diagnostic tests 1: sensitivity and specificity. Br Med J 308(1552)
29. Salton G, McGill MJ (1983) Introduction to modern information retrieval. McGraw-Hill, New York
30. Fawcett T ROC graphs: notes and practical considerations for researchers. HP Labs Tech Report, HPL-2003–4
31. PROMISE Software Engineering Repository. http://www.promise.site.uottawa.ca/SERepository/
32. Jalaji A, Goff R, Jackson M, Jones N, Menzies T (2006) Making sense of text: identifying non functional requirements early. Submitted to PROMISE 2007, also available on-line as a West Virginia University CSEE technical report, 2006. http://www.stupidchoices.org/cs591o/final_pres/ottawa.ca/SERepository/