10 Managing Large Repositories of Natural Language Requirements

Johan Natt och Dag and Vincenzo Gervasi

Abstract: An increasing number of market and technology driven software development companies face the challenge of managing an enormous amount of requirements written in natural language. As requirements arrive at high pace, the requirements repository easily deteriorates, impeding customer feedback and well-founded decisions for future product releases. In this chapter we introduce a linguistic engineering approach in support of large-scale requirements management. We present three case studies, encompassing different requirements management processes, where our approach has been evaluated. We also discuss the role of natural language requirements and present a survey of research aimed at giving support in the engineering and management of natural language requirements.

Keywords: Large-scale requirements management, Linguistic engineering, Natural language processing, Relationships, Redundancy, Duplicates.

10.1 Introduction

Market and technology driven companies developing increasingly complex software products eventually face the challenge of dealing with huge information flows that may overwhelm their management and analysis capabilities. Requirements are particularly difficult to manage effectively due to their unstructured nature. The requirements also have a potential to grow to such volumes and arrive at such rates that specific information and knowledge management challenges emerge: deterioration of the requirements repository and an increasing difficulty to identify and maintain requirements inter-relationships.

A major reason for these problems is that requirements are communicated in natural language, which induces several problems like imprecision, ambiguity, incompleteness, conflict, and inconsistency, which take time to resolve (see Chap. 11 for a separate discussion on ambiguity). Requirements management processes may be very different in design. Nevertheless, companies that acknowledge both customer involvement and their own innovative potential as rewarding means for discovering successful product services and functionality are faced with a common challenge: analyzing and evaluating every incoming requirement, customer wish and technical suggestion as soon and as thoroughly as possible.

In traditional requirements management [49] there is an implicit focus on isolated monolithic requirements specifications. The new challenge of managing enormous amounts of requirements that continuously must be analyzed, re-analyzed and consolidated is generally left untouched. This is also reflected by current requirements management tools, which do provide the functionality to assign links between requirements, but give no assistance in the actual matching of thousands of incoming requirements with those already analyzed. Requirements management tools could do better than providing simple keyword search facilities to alleviate the manual burden of consolidating large amounts of requirements.

Companies facing these challenges may arrive at a cross-road where the choice is to reduce the flow of incoming requirements or to assign more resources to handle them [24]. However, seen from a business perspective, neither of these approaches is particularly rewarding (and in many situations impossible). Choking the elicitation and invention of new requirements will increase the risk of missing potential business opportunities [28], and adding more people to do the job has been shown to be too costly and at times counter-productive [3, 24].

In this chapter we present a linguistic engineering approach that may give considerable support in the continuous management of large amounts of textual requirements. The approach is based on techniques from information retrieval where similarities between requirements are calculated to indicate the semantic overlap. This gives a possibility for product managers to more quickly find relationships between requirements based on their textual content.

In Sect. 2 we will first discuss the general role that natural language requirements play in large-scale software development. Section 3 provides an in-depth survey of the current research in linguistic engineering applied to requirements engineering and management. Section 4 introduces the idea of calculating similarity between requirements as a means for identifying semantically related requirements. In Sects. 5 through 7 we present three case studies conducted at three software developing companies, while Sect. 8 concludes the chapter.

10.2 The Role of Natural Language Requirements

A recent survey supports our own research experience that requirements to a very large extent are written and communicated in natural language (NL) [34]. Still, after years of rewarding research that has helped us understand and improve the way requirements may be specified and formulated [Chaps 3 & 11], the state of the practice is generally that requirements quality guidelines are rarely applied. There is a large gap between the formal models advocated by many researchers and the informality that dominates in industry. Several reasons can be identified to why requirements are initially specified in natural language and in many cases kept in that form throughout the development process:

- NL is the primary communication language, which is shared by all stakeholders and participants in the development process. Formal languages require specific training, which is unrealistic to expect from every stakeholder and in particular from customers or end-users.
- Requirements engineering (RE) is a social and evolutionary process where requirements are elicited and specified at different levels of abstraction at different points in the development process. NL is universal, meaning that it can be

used to talk about arbitrary domains and at arbitrary levels of abstraction. Many formal languages do not have this strength.

- In large-scale development there are comparatively few of the proposed requirements that are actually selected for implementation (see for example Chap. 4 on prioritization and Chap. 13 on market-driven RE). Since not all requirements are expected to be implemented, there is little motivation for spending time formalizing them. In particular, our experience tells us that companies which value close interaction with their customers and rapid reaction to changing market conditions do not find it cost-beneficial to translate all requirements into formal specifications.
- Many formal methods do not offer any support for the management and analysis of erroneous, incomplete, or partially-specified requirements. In contrast, NL techniques adapt naturally to such situations, which in practice make up a large part of a requirement life cycle.
- While formal languages can improve our ability to check internal consistency and completeness of requirements (a process often referred to as *verification*), they cannot capture external properties of the requirements, e.g. correspondence between the requirements and the actual user intentions. It requires good communication and interaction with the stakeholders to verify such properties (*validation*) – and to this end, NL is a more suited language.

Thus, despite its recognized and infamous deficiencies, there are few incentives to avoid natural language. We should therefore expect that its use cannot be escaped. This is also elucidated by M. Jackson stating that RE is where the informal meets the formal [25]. The gap between the users' needs and a new release of the software system must therefore be bridged using methods and techniques that acknowledge, in some form, communication in natural language.

An increasing number of software development companies move away from isolated contract development projects (also called bespoke software development) towards development for a broader market. This is, for example, also indicated by the growing interest in commercial off the shelf (COTS) development in the RE research community [52]. Companies developing for a broader market face distinct challenges, of which one crucial is to stay ahead of competitors and reduce time-to-market [Chap. 13]. After an initial version of a product has been released, there is a need for a dynamic process of elicitation and prioritization. In this dynamic environment where requirements arrive from many different sources and stakeholders (customers, sales representatives, developers, support personnel), the decision of which requirements are to be included in the next release of the product must in most cases be made based on the NL requirements available in the repository, in addition to the experience and skill of the product manager and certain nonnegotiable requests by key customers.

In essence, these companies face an information overload problem. But, as we already pointed out in the introduction, the most apparent solutions (reducing the inflow of requirements or adding personnel) are not satisfying. Another approach has therefore been examined, which aims at supporting requirements analysis activities through automation. The proposed solution is to use the techniques from *natural language processing* (NLP) [26].

10.3 State of the Research Addressing NL Requirements

As pointed out in the well-referenced paper by Ryan [46], there have been many unrealistic expectations on NLP techniques given the desire for a system that could support the currently expensive activities within RE. These expectations are typically based on misconceptions about what the communication problem in industrial RE really is and to what extent the requirements on a system are available in textual form (e.g. see [51] on linguistic problems with requirements elicitation). Ryan concludes that RE is a social process and that linguistic techniques can succeed only in a supporting role to this process –not by trying to replace it.

A pragmatic approach is suggested by Garigliano, who points out a range of criteria for applied systems dealing with natural language [18]. The criteria elucidate the possible variation points for the usefulness of an NLP-based system. In essence, it is a matter of systematic cost-benefit analysis.

To relate our work to the current body of knowledge, we present here a survey of research aimed at supporting RE activities using linguistic engineering techniques, grouped by three major RE process activities addressed:

- *Domain and requirements understanding*, which is a fundamental success factor in all systems and software development.
- *Requirements verification and validation*, which are carried out to ensure that a specification is internally consistent and to certify that the requirements are a correct representation of the users' intentions [2, Chap. 8].
- *Requirements management,* dealing with storage, change management and traceability issues. This is within the scope of this paper.

We encourage the interested reader to look into the work of each author. In many cases the industrial applicability and scalability is yet to be determined through larger case studies with real data. Also, although most approaches acknowledge ambiguity and inconsistencies, seldom is it reported how any other pollution in the data is treated (e.g. misspellings and non-information carrying characters). A combination of different techniques would likely be the most rewarding and the research surveyed provides an excellent basis for this acquisition.

10.3.1 Domain and Requirements Understanding

A central task in domain and requirements understanding is to identify and understand *domain concepts*, also called *domain abstractions*. Domain abstractions are general concepts that are formed to represent common features of specific instances in the domain. Domain abstractions make communication more efficient within the domain, but developers must nevertheless take into account not only the general concept, but also the specific instances, in order to fully understand the abstractions. Domain abstractions are typically represented in NL through sets of terms (often nouns and noun phrases). Researchers have therefore investigated linguistic engineering techniques to extract these terms, representing the abstractions, from the discourse generated from interview transcripts and customer wishes expressed in natural language. Following is a survey of the major research efforts addressing abstractions.

Goldin and Berry [21] presents an original approach and a prototype tool for suggesting requirement abstractions to the human elicitor. Their method compares sentences using a sliding window approach on a character-by-character basis and extracts matching fragments that are above a certain threshold in length. The approach can properly handle arbitrary lengths, gaps and permutations and avoids some specific weaknesses in confidence and precision when using only parsers or counting isolated words.

Rayson et al. [44] present two experiments in probabilistic NLP using tools they have developed (part-of-speech and semantic taggers integrated into an end-user tool). The results suggest that the tools are effective in helping to identify and analyze domain abstractions. This is further supported by a later study by Sawyer and Cosh [47] where ontology charts of key entities are produced using collocation analysis.

10.3.2 Requirements Verification and Validation

It is generally acknowledged that spending more time in the verification and validation stages and finding errors early is more rewarding than proceeding too soon to coding [1, 9, 10]. Therefore, considerable research effort has been put applying natural language processing to support requirements verification and validation. The two activities are not carried out separately. Checking a set of requirements may reveal internal inconsistencies that may as well be external, which must be resolved with a stakeholder. Therefore, requirements verification and validation are here addressed together.

Gervasi and Nuseibeh [19] treat validation as a decision problem on whether a given software model, generated by parsing the requirements text, satisfies certain properties. Their experiment with the use of lightweight formal methods shows that even subtle errors, not discovered by human inspection, may be identified.

An approach to improve the quality of written requirements is proposed by The Goddard Space Flight Center's Software Assurance Technology Center (SATC) [53]. They have derived seven quality indicators used for measuring the quality of requirements specifications. These have been used to develop a tool which is used by NASA to improve their requirements specifications. Fabbrini et al. [11, 12] also propose a quality model and have implemented a tool to show the quality model's industrial applicability. Fantechi et al. [13] have applied both the tool by Fabbrini et al. and SATC to evaluate the quality of 100 use cases. They conclude that although the techniques may support quality evaluation, they are not sufficient to completely address correctness and consistency. Cybulski and Reed [6, 7] describe an elicitation method and a supporting management tool that help in analyzing and refining requirements A set of NLP components are used to force the requirements engineer to rephrase requirements in order to unify the terminology.

Burg and van de Riet [4] have developed an approach and a supporting environment for specification, verification, and validation of functional requirements. Verification is supported graphically, lexically, and logically, while validation is supported through paraphrasing (transforming models into language readable by the user or customer) and simulation of the dynamic behavior. In several different ways they show how the approach enhances the quality of the specification. Park et al. [42] present an implementation of a requirements analysis supporting system, which may help to identify conflicts, inconsistencies, and ambiguities in requirement. Their approach to combine syntactic parsing with a sliding window method gives more accurate similarity measures than using them separately.

To further adapt the language to formal validation, several researchers have proposed to explicitly restrict the language used in requirements. The suggested advantage is that it may be used by domain specialists that want the benefits from formal languages but who lack the required training. Fuchs and Schwertzel [16] and Macias and Pulman [31, 32] use a subset of English to forbid the expression of ambiguous sentences. Cyre and Takar [8] define a syntax and grammar of restricted English. Somé et al. [50] go one step further and restrict the language and semantics to a scenario style, albeit more understandable by the user than formal specification. Osborne and MacNish [41] suggest using extensions to a parser with a wide-coverage grammar in order to identify and present syntactic and semantic ambiguities to the requirements analyst.

Towards formalization, Fliedl et al. [14] suggest the use of a conceptual predesign model to bridge the gap between the NL representations and enable formal validation. The pre-design model is not as technical as common conceptual modeling languages, while still supporting the general principles behind several different conceptual models (e.g. use cases, state charts, etc.) and the mapping to more formal model.

Nanduri and Rugaber [38] use object modeling technique guidelines and a link grammar parser for transforming high level specifications into object charts. Although their tool produces object diagrams that may help identify omissions, the approach suffers from several common problems when trying to transform natural language requirements into object models: parser limitations, ambiguity, incompleteness and insufficient domain knowledge and transformation rules. A similar approach is taken by Mich and Garigliano [35]. Rolland and Proix [45] describe a prototype that aims at providing support to problem-statement acquisition, elicitation, modeling and validation. It has not been validated but likely also suffers from the common problems listed above. In a recent paper García Flores [17] proposes to use NLP techniques to extract relevant sentences from and identify inconsistencies within large requirements corpora. The approach uses shallow parsing and contextual exploration networks, based on the presence of certain textual markers in the text. It has not yet been evaluated.

10.3.3 Requirements Management

As previously noted, large-scale software systems development involves a considerable flow of requirements. Requirements are elicited and arrive from many different sources and constantly change [49]. When numerous requirements arrive each month, either in bursts of thousands or continuously 3-5 requirements each day, the importance of proper requirements management activities becomes very apparent. Although requirements management is intertwined with the traditional software development process (i.e., where requirements are further analyzed and successively formalized into specifications, ending in executable and tested code), there are requirement management activities that take place before actual development starts [Chap. 13]. But, although it may be clear what must be done, the requirements management process easily becomes overloaded due to the sheer number of requirements. Thus, there is a strong need for more supportive tools.

Current requirements management tools provide facilities for storing and recalling requirements, annotating them with metadata (usually consisting in arbitrary attribute/value pairs, where standard sets of attributes are offered as libraries), and for managing relationships between requirements. Indexing, keyword-based search, and search on metadata are normally provided. Unfortunately, the management of relationships is most often limited to manually establishing links (typically used for traceability) between pairs of requirements. Some link types can be declared as fragile, in that any change in one of the linked requirements marks the link as broken until manually verified and re-established by the user.

Surprisingly, there are, beside the cases presented later on in this chapter, no specific attempts that directly try to tackle the management challenges by using natural language requirements processing. In particular, the following specific hands-on requirements management activities are open for scrutinized research:

- Matching incoming (potentially new) requirements to previously elicited, planned, and already implemented requirements
- Maintaining a separation and finding relationships between customer requests and requirements invented within the organization
- Identifying dependencies and other interrelationships between requirements [5]
- Supporting the extraction of requirements from the repository that fit strategic areas (e.g. invoicing capabilities, decision-making features)

Difficulties in performing these activities are a major obstruction in the efficient management of elicited, invented and implemented requirements. Any technique that may support requirements maintenance and management activities, even if partially, can be expected to be warmly accepted in industry.

10.4 Requirements Similarity

In this section we introduce a fundamental concept in our discussion, that of *requirements similarity*. As we will see in the following, a number of problems in the management of large volumes of requirements can be solved or at least alleviated by using a measure of how similar two requirements are. Naturally, many different notions of similarity can be used. In most problems, what is needed is a notion of *semantic similarity*: a measure of whether two requirements convey the same meaning, and to what extent. However, other notions of similarity can also be used. A few of these are listed in Table 10.1; more measures can easily be obtained by considering other metadata about the requirements (e.g., priority assigned, system version targeted, approval responsibility, implemented status, etc.).

Table 10.1 A listing of some similarity measures

Similarity measures	**Description**
Semantic	Similarity in meaning
Syntactic	Similarity in grammatical structure
Lexical	Similarity in words used
Structural	Similarity in sectional structure
Extensional	Similarity in size
Argumentative	Similarity in rationale
Goal	Similarity in objective
Source	Similarity in the proponent
Function	Similarity in function addressed
Object	Similarity in system parts affected
Temporal	Similarity in time of origin

Whatever measure is chosen, in order to be applicable to the management of large repository it must possess a fundamental property: it has to be computable in a relatively inexpensive way. Any measure requiring significant human intervention will be too costly to be used on large requirement repositories; we are thus forced to focus on similarity measures that can be computed in a totally automatic way. Unfortunately, given the current state of the art in natural language processing and in knowledge representation, it is not feasible to extract meaning in a reliable way from totally unrestricted natural language text as that found in most requirements. We therefore focus on *lexical similarity* as a way of approximating semantic similarity.

On a lexical level, we consider a requirement as a sequence of *words*. The exact definition of what a word is varies with the language and the application. More refined approaches distinguish the various lexical (and at times, morphological) constituents of requirements with more precision, e.g., punctuation (as in ","), contraction markers (as the apostrophe in "can't"), parenthetical structures (as "(") etc. can be considered as words on their own. We refer to the process of separating the lexical constituents of a requirement as *tokenization*, and each word (in this extensive definition) is called a *token*. In the upcoming case studies, a token is regarded as sequence of letters and/or digits. Any other characters are regarded as delimiters and thus discarded.

$$sim(p,r)=\frac{\sum_i w_p(v_i)\cdot w_r(v_i)}{\sqrt{\sum_i w_p(v_i)^2\cdot\sum_i w_r(v_i)^2}}$$

Fig. 10.1 The Cosine measure

Tokens can be further processed in various ways. Most typically, tokens are reduced to their base form, removing morphological inflections (e.g., reducing plural nouns to their singular form, or removing person, mood or aspect information from verbs). This process is called *stemming* and is usually performed with the help of general morphological rules, and a dictionary listing exceptions to those rules. In Case 1 we have used the well-known Porter stemmer [43], but in Cases 2 and 3 we have switched to a newer one, reported to perform better [37].

Another common operation is *stop word removal*. It consists in dropping from the sequence of tokens all those words that have a purely grammatical role. The grammatical information they convey may be stored in some other form (e.g., in parsing trees) before removing the stop words, if so desired. Again, the details of the process depend on the language at hand, and on the kind of analysis that is to be performed on the requirements. In most cases, stop words coincide with so-called *closed class words*, e.g. articles and prepositions. Also in this case, a special-purpose dictionary can list exceptions. In the presented cases we have used a stop word list comprising 425 words derived from the Brown corpus [15].

Further various processing steps are possible, but for the sake of brevity we return now to the problem of measuring requirements similarity. We can formally consider a requirement r taken from a requirement set ▯ as a finite sequence $r=\left\langle v_{i_1}, v_{i_2},\dots, v_{i_m}\right\rangle$ of tokens drawn from a given alphabet $V=\{v_1, v_2, \dots v_n\}$, which includes all the tokens that appear in our requirements database. Using the preprocessing steps described above, V would consequently contain stemmed tokens that do not appear in the stop word list. If order is not considered important, an alternative representation is possible: a requirement r can be considered as a vector $a_r=[w_r(v_1), w_r(v_2), \dots w_r(v_n)]$, where $w_r(v_i)$ denotes the weight, or relative importance, of the token v_i in requirement r. Different weighting schemes are possible. As requirements expressed in feature style are more focused than literary text, we assume that the tokens remaining after the preprocessing step are all equally valuable. In Case 1, we apply the simplest weighting scheme, assuming that weight coincides with frequency. However, as it is considered that the importance of a token is not linearly proportional to the number of times it occurs, in Cases 2 and 3 we also use the well-known weighting formula *1+log₂(term frequency)* [33]. Case 2 explicitly compares the results obtained by using the two schemes.

Once requirements have been encoded as vectors, it becomes possible to apply standard similarity measures. In Case 1 we chose to compare the performance of the Dice, Jaccard, and Cosine measures [33]. Their most significant difference is how they treat different lengths of the compared requirements. In Case 2 and 3, the Cosine measure was selected as it was considered to generally perform better than the other two. This measure got its name from calculating the cosine of the angle between the vectors that represent the requirements in a vector-space model.

Formally, given two requirements, *p* and *r*, we have that the similarity between *p* and *r* is given by the formula in Fig. 10.1 (An example of applying the measure can be found in [39]). The definition assumes that the vector space employed has a Euclidean distance, uniform across all dimensions. This is of course a gross oversimplification: in practice, the presence or absence of certain terms may be much more important and revealing of true semantic similarity than that of other terms. However, since we are mainly interested in techniques that work irrespective of the exact domain and language used, and for the sake of generality, we will accept this simplification, keeping in mind that more refined techniques can be employed in specific domains.

In the following sections we present three case studies in which the technique of calculating similarity between requirements has been evaluated. For the evaluation we utilize the widely adopted measures of recall, precision, and accuracy. Since their usage and interpretation is dependent on the application we leave the definitions and explanations of the measures to each individual case.

10.5 Case 1: Keeping the Repository in Shape

Telelogic AB develops a software development environment for real-time systems called Telelogic Tau, which supports standardized graphical languages and code generation. Telelogic Tau is marketed globally and requirements are collected continuously from several different sources (e.g. marketing, support, development, testing, usability evaluations, and technology forecasting). The requirements are collected into a repository and assigned the status of "New". Each requirement then undergoes a series of evaluations and refinements, such as checking for appropriate detail level, and assignment of cost, impact, and priority. Each requirement has a lifecycle progressing through specific states in the development process. So, for example, when a requirement has been implemented and verified, it is assigned the status "Applied". Thus, all requirements are kept in the repository, which continuously grows.

In its initial state a requirement is checked for three related properties: (1) whether or not the requirement is regarded as a duplicate of another requirement already in the repository, (2) if it is possible to merge the requirement with another requirement, or (3) if the requirement should be split into two or more requirements before further analysis. If a requirement has one of these properties, it is assigned the "Duplicate" status and an appropriate action is taken. When a requirement is merged, all the information is added to the requirement it is merged with. When a requirement is split, the information is distributed over two or more new requirements. When a requirement is a pure duplicate, no further action is taken.

As requirements arrive at an average rate of three per day, and as the requirements repository unendingly increases in size, these activities are causing congestion in the requirements process [24]. Automated support in this situation, using similarity measures to identify duplicates, is suggested to help avoiding deterioration of the repository and enable a quicker way of checking arriving requirements against the ones stored in the repository.

10.5.1 Case Study Requirements Data

A snapshot of the state of the requirements and the repository by the year 2000 is shown in Table 10.2. Of the 1,920 requirements in the repository, 130 had been identified by the analysts as being duplicates, merges, or split sources (i.e. assigned the status 'duplicate'). Example requirements may be found in [40].

Table 10.2 Number of requirements in the database.

New	Assigned	Classified	Implemented	Rejected	Duplicates	**Total**
406	428	601	252	103	130	**1,920**

10.5.2 Evaluation

Of the 130 requirements marked as duplicates, we only consider the 101 that were real duplicates for evaluation purposes, as merges and split sources would match partially and thus bias the results. Moreover, we use the standard measures of *recall, precision* and *accuracy*. Let $sim(r_i,r_j)$ be a function that takes a pair of requirements and gives a similarity measure between 0 and 1, and t be a threshold value, which acts as a selection criteria. If $sim(r_i,r_j) \geq t$ then (r_i,r_j) are considered to be a suspected duplicate pair. Recall is calculated as the percentage of the actual duplicate pairs that fall above the similarity threshold. Precision is calculated as the percentage of actual duplicates above the similarity threshold in relation to all pairs above the similarity threshold. Finally, accuracy is the percentage of all duplicate pairs that fall on the correct side of the threshold (i.e. correctly suggested duplicate pairs and non-duplicate pairs respectively).

The textual information used to represent each requirement was collected from the "Summary" field, which corresponds to a short requirement title, and the "Description" field, which corresponds to a further explanation (see the examples in Table 10.6 and Table 10.7 in the appendix). These fields were then pre-processed according to the steps described in Sect. 10.4. To investigate the impact of different similarity measures we calculated recall, precision, and accuracy curves for the three different measures in Sect. 10.4. The results are presented in Fig. 10.2, which shows that recall decreases from around 80% at threshold level 0+ to just below 20% at threshold level 1. At threshold levels 0+ and 1 the similarity measures perform exactly the same (as expected considering the formula) but between these two extremes the curves differ. The Dice measure gives slightly worse recall compared to the Cosine measure and may thus be discarded. The best choice between the Jaccard and the Cosine measure is not obvious. The Cosine gives higher recall but lower precision than Jaccard. The choice would thus depend on the application.

The low precision at threshold level 0+ may at first seem very discouraging. However to properly evaluate the feasibility of the approach in an industrial setting, a deeper investigation of the requirement pairs is needed. Taking any two suggested pairs, they may or may not involve the same particular requirements. For example, the requirement pairs (*A*, *F*) and (*C*, *F*) share the requirement *F*. If the analyzer assigns similarity values above zero to each of these pairs and a similarity value equal to zero to the pair (*A*, *C*) it would nevertheless be interesting to look at the three involved requirements together. We denote these preferred groupings of requirements as *n-clusters*, where *n* is the number of requirements in the cluster. The two single pairs in the previous example will thus form a 3-cluster. The cluster distribution can be derived by calculating the transitive closure of a graph in which the nodes correspond to requirements and edges correspond to pairs of requirements (r_i, r_j) with $sim(r_i, r_j) \geq t$.

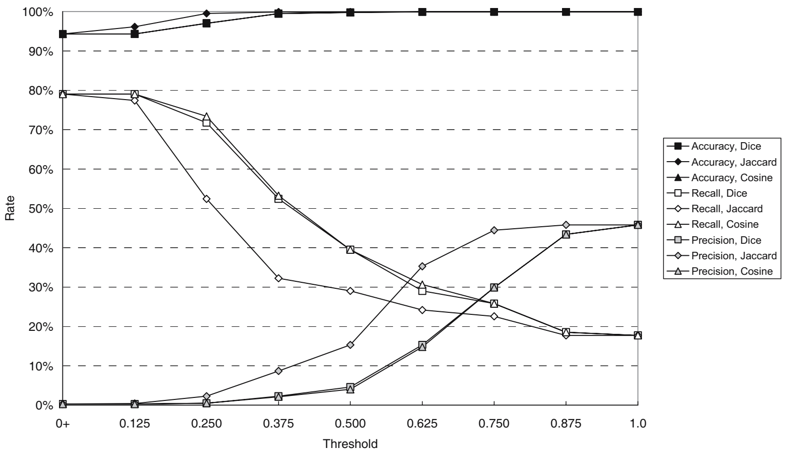


Fig. 10.2 Performance of the similarity measures

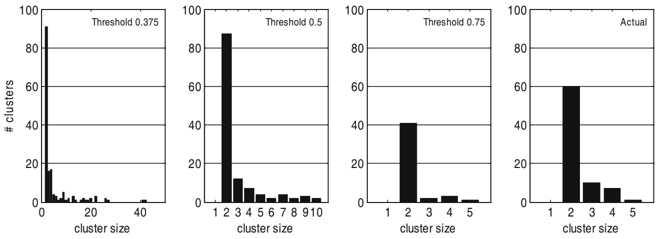


Fig. 10.3 Requirements cluster distribution using the cosine measure on the *Summary* and the *Description* fields. The three leftmost graphs show the number of clusters of different sizes for various thresholds compared to the actual cluster distribution on the right

The cluster distributions at three different threshold levels are shown in Fig. 10.3. The last graph shows the cluster distribution for the actual duplicates found by the experts. The graphs show that with increasing threshold the number of clusters of larger size decreases. For example, at threshold level 0.375 there is one very large cluster involving 42 different requirements.

Noteworthy is that the presented evaluation is made on a snapshot of a reasonably large set of requirements. However, at Telelogic, the requirements arrive continuously, a few at a time. The similarity analysis can thus be made incrementally on a smaller set of requirements, avoiding the need for interpreting the results of similarity analysis of the entire set of requirements at one time. The cluster distribution shows that if we analyze one randomly selected requirements from the database (which may represent a newly submitted requirement), the worst case would be that the analyzer suggests a cluster of 42 requirements to be identical. This is thus the maximum number of requirements the requirement analyst must handle simultaneously. As the number may seem too high for the lower thresholds, it is reasonable to suggest that such large clusters may be ignored as they are probably irrelevant.

Table 10.3 Expert reanalysis of requirements presumed to be incorrectly classified

Relationship	**Count**
Duplicates	28
Similar	13
Related	8
Part of	5
Not related	21

Another interesting issue is whether the automated analyzer reveals duplicate pairs that the experts missed. To explore this we let an expert analyze 75 requirements that were suggested as duplicates using the Cosine measure at threshold level 0.75, but had not been assigned as duplicates by the experts. Table 10.3 shows the surprising result from the analysis. It turned out that 37% of the suggested duplicate pairs had been actually missed by the experts! For that threshold level, recall would increase from 25% to 40%, precision from 30% to 56% and the already high accuracy would become even higher. The analyst did not regard two requirements in a pair as duplicate or similar if they were to be implemented in different parts of the software. The table also shows the additional relationships identified, which thus imply that only 21 of the 75 pairs identified would be completely wrong. These 21 erroneously identified pairs should be put in relation to the several thousand potential suggestions. In an industrial setting it is better to have a few extra suggestions that may be discarded rather than missing any actual duplicates. Stated differently, it is (to a certain extent) of greater interest to increase recall at the expense of precision.

10.6 Case 2: Linking Customer Wishes to Product Requirements

Baan, now part of SSA Global, develops large complex applications aimed for enterprise resource planning, customer relationship management, supply chain management, product lifecycle management, and business intelligence. Continuously, new customer wishes, called *Market Requirements* (MR), and product requirements, called *Business Requirements* (BR), are inserted into the *Baan Requirements Database* (BRD) upon their receipt or creation, respectively. Periodically, the company management decides to start a new release project, and a number of BRs are selected for implementation –preferably, in such a way as to maximize the number of MRs that are satisfied in the new release, compatibly with time and budget constraints. Customers receive informative messages when a MR is accepted in the BRD and when it is satisfied in an upcoming release. Thus, establishing complete and correct MRs-BRs links is paramount to maintaining good relationships with the customers.

MRs and BRs that cover the same underlying functional requirement are linked to each other in a many-to-many relationship; a single MR can span several BRs (e.g., to split a huge work package into manageable pieces), and a BR can satisfy several MRs (e.g., when several customers are requesting the same functionality). MRs are copied into the BRD as-is, i.e. without altering the original text as specified by the customer. Linking MRs to BRs and the other way round is a daily routine for product managers. Each time a new MR is inserted into the BRD, it is first checked by searching whether there are one or more BRs that already include the specified functionality. This process is very time consuming, as the current tool only allows text search in the requirement description. Similarly, when a new BR is created, the corresponding MRs need to be found in the BRD, since the objective is to satisfy as many customers as possible. Finding all MRs that are covered by the BR at hand is virtually impossible, because of the large number of MRs and due to the time-consuming understanding of MR content. Advanced automated assistance to the MRs-BRs linking can improve the quality of the requirements management process and save costly man-hours of the product managers.

Given the favorable lexical features of the requirements, that use mostly terms from a restricted domain, we propose a tool-supported linking process that integrates well with the existing practices and technologies, while at the same time reducing the cost and improving on the effectiveness of manual linking. Based on the similarity calculations, a tool can suggest which requirements already in the BRD could be linked to an incoming MR or BR. The human expert can then decide whether to accept these suggestions or not, or can decide to resort to keywords-based search (as in the original process) for further options. Our expectation is that relevant suggestions will be provided faster this way than if a human would have to select several different search terms and, for each of these, search through the database.

10.6.1 Case Study Requirements Data

The total number of business and market requirements elicited at Baan between 1996 and 2002 and manually linked to each other is found in Table 10.4. Overall, the analyzed corpus contained almost one million words, with MRs contributing approximately two thirds of the total, and BRs constituting the remaining third. Representative examples of each of the two kinds of requirements may be found in [39].

Table 10.4 Requirements elicited and linked at Baan, 1996

Year	Business Requirements		Market Requirements	
	Elicited	Linked	Elicited	Linked
1996	0	0	183	113
1997	5	4	683	262
1998	275	169	1,579	388
1999	709	261	2,028	502
2000	669	167	1,270	397
2001	1,000	153	864	224
2002	1,121	340	1,695	514
Total	**3,779**	**1,094**	**8,302**	**2,400**

In current practice, the association between the two presented requirements would be found by emanating from the submitted BR in Table 10.9, searching for the term *container* among the MRs. Such a search returns 37 hits if searching only in the label field and 318 hits if searching the description field. Experts would then have to browse through all the MRs returned by the search. However, historical data shows that only five MRs were actually linked by the experts (all five were submitted earlier than the BR).
Of these, four could be found by searching for *container*, but the last relevant MR was not returned by the search, and required a new search (for example, on *statistics*, which however adds another 40 hits on the label field and 99 hits on the description field to the already daunting set of candidates to examine). Based on this and similar cases, we estimate that significant time can be saved by replacing the search procedure based on designated keywords with a more sophisticated one based on lexical similarity of the requirements.

A potential hurdle to be overcome is the varying linguistic quality of the text of the requirements. As in Case 1, requirements are often typed in haste, and may contain acronyms, spelling errors, code snippets, colloquial language, etc. We investigated these occurrences for a subset of all terms (those starting with “a”), finding that non-word entities represent around 2–3% of the whole corpus, with spelling errors (the only real threat to lexical matching) only accounting for 0.3%-0.4%. We can therefore assume that the calculation of lexical similarity will not be significantly affected by occasional typing errors in the requirements. The investigation also showed that the two sets of requirements (MRs and BRs) have very similar composition in terms of statistical features. A more detailed comparison can be obtained by considering the two lists of distinct term occurrences, ranked by frequency. The two lists have a 4,660 terms intersection (most of them in the topmost ranking positions); 1,899 terms only occur in BRs, with 8,234 terms only occurring in MRs. Overall, the Spearman rank order correlation coefficient for the two lists is $r_s \cong 0{,}78$, significant at the $p < 0.00003$ level (see [48] and [27] for a discussion on statistics for corpora comparison). The correlation coefficient gives a good indication that a shared lexicon is being used in the two kinds of requirements. This is not surprising, as both MRs and BRs are discussing issues in a restricted domain. In turn, this gives support to our assumption that in this context lexical similarity can be a good approximation for semantic similarity.

10.6.2 Evaluation

In order to evaluate how well the approach presented above performs for identifying correct links, we use the links established manually by the various product managers as the "presumably correct" answer. Our goal is to find out how many of these links the automatic approach can retrieve.

In our industrial setting, we can expect user interaction to consist in the following steps:

1. A new requirement (MR or BR) is submitted to the BRD.
2. A tool computes the similarity score between the new requirement and the preexisting ones of the opposite type (i.e., BRs or MRs, respectively), and ranks all the requirements according to the similarity score.
3. The top-ranking *n* requirements are presented to the user for manual verification and, optionally, for establishing links in the BRD.
4. Optionally, the user can "scroll down" the list, and check the next page of results.

The size of the top list *n* will thereby represent our similarity threshold. A top list size of 7±2 could be a good compromise [36], as such a size would enable the user to quickly spot one or more correctly related requirements, while taking into account that we are not able to reach 100% recall or precision anyway. In this situation it is not critical that a correct suggestion is presented at position 1 but, of course, the higher the position the better. We could then use the ranked recall measure [26], but as we would like to relate the recall to a threshold (i.e. the top list size) we choose to compute recall for different top list sizes. Recall is in this case the proportion of the target items that a system gets right (i.e., true positives divided by the total number of answers returned) and we use the following adapted procedure to compute it:

1. Compute the complete similarity matrix
2. For each requirement of one type, sort the requirements of the other type by similarity
3. Calculate the overall recall for a top list of size *n* as the ratio between the number of correct links identified among the top *n* ones and the total number of correct links

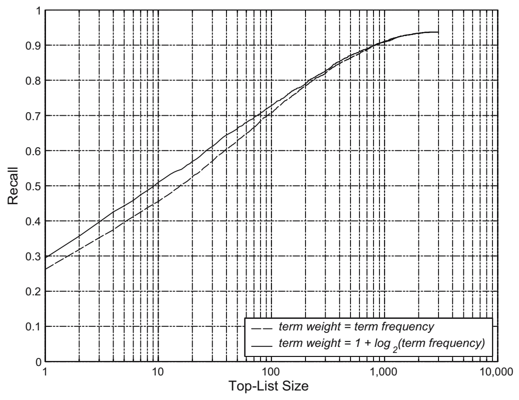


Fig. 10.4 Recall for linking a MR to BRs

The recall curve as a function of the top list size is shown in Fig. 10.4. The figure shows the recall curve for the top lists of suggested BRs for each MR. The dashed line represents the recall curve for calculating the similarity using just the *term frequency* as weight; the solid line represents the recall curve using 1+log$_2$(*term frequency*), which provides slightly better recall.

The figure reveals that we only reach a maximum recall of approximately 94% (though with an unreasonable top list of 3,000 requirements). This is due to 204 requirements that have been linked manually but have no terms in common at all. The 204 links are particularly interesting to look at, as they represent cases where our assumption about the validity of lexical similarity as an approximation of semantic similarity does not hold. In particular, we found that:

- The links comprised 101 BRs and 158 MRs.
- The majority of the requirements were sparingly described, consisting of just a single line of text. In some cases there was no description at all. This is not necessarily wrong in the Baan RM process perspective (an empty BR is allowed to be created and directly linked to an MR). These special cases do, however, affect the results negatively.
- Some requirements were completely written in languages other than English, while the requirements they were linked to were written in English. This should not be allowed without an additional English description, and of course makes automatic matching practically infeasible.
- Some of the linked BRs and MRs seemed to us to describe completely different things. They could have been erroneously linked, or perhaps be related in a way that escaped our understanding. For a more thorough analysis of these cases further work would be required, which is beyond the scope of this analysis.

On the positive side, Fig. 10.4 shows that, for a very reasonable top list size of 10, we reach a recall of 51%, which is good considering the pragmatic approach taken and the impact on the saving of time that could be made in industry.

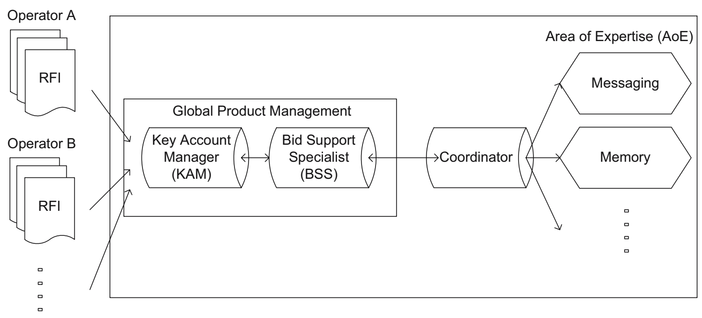
To get an impression of the time that could be saved, we can make a rough estimate based on the statistics presented and on another measure reflecting how many requirements could be completely linked just by browsing a top-10 list. We found that for 690 of the BRs, the recall rate would be 100% using a top list size of 10, i.e. every related MR for each of the BRs would be found within a top-10 list. These 690 BRs are linked to 1,279 MRs, giving an average of 1.85 MRs per BR, but in order not to exaggerate the gain we assume that, in the manual case, one search term would be enough to find all the links for one requirement. Supported by the search hit example in [Sect. 10.6.1], we further assume that a search would return approximately 30 hits. Thus, in the manual setting the average case scenario would be to browse 30 requirements. With a top list size of 10, the worst case scenario with automated support would be to browse 10 requirements. Up to 66% effort could consequently be saved. If we assume that it takes about 15 seconds to read a requirement and either accept or reject it as a link, we find that the overall gain is 57.5 hours.

The critical reader might observe that in a real setting it is not possible to know when to stop perusing the list, as more relevant links could be found by further browsing. The same applies to the manual case: searching for more keywords could yield more links. Nevertheless, the data from our case study show that a similar level of coverage can be reached more efficiently (i.e., with less effort) by applying lexical similarity when compared to keywords search. If so desired, the time saved can be spent in increasing the level of coverage, by examining more candidates, or devoted to other RE activities if the coverage attained is deemed acceptable.

10.7 Case 3: Managing Redundant Customer Requests

Sony Ericsson Mobile Communications AB (SEMC) develops mobile phones for a global market. As such, they must handle requirements from many different sources in the RE process. SEMC's primary customers are the mobile phone operators, who sell the phones to the end user, either directly or through a third party. In order for the operators to acquire knowledge in the technical capabilities of SEMC's phones, so called Requests for Information (RFI) are submitted to SEMC by the operators. Two kinds of RFI's can be identified: *general requests for information* and requests for *statement of compliance* (SoC). SoCs, which are the most common ones, comprise specific requirements and are replied upon using simple standardized statements on whether or not a certain product complies, i.e. whether or not a stated requirement is fulfilled by the product.

The RFI process is depicted in Fig. 10.5. Each year each operator submits a couple of RFIs. The RFIs arrive to the Key Account Managers (KAM), one for each major operator, in different document formats (PDF, Excel, MSWord, etc) and at different times. The main specification technique for the RFI requirements is feature style, i.e. function specification in natural language [30]. The KAM passes the RFI on to a Bid Support Specialist (BSS), who reviews the RFI from a



Operator A
RFI
Operator B
RFI
Area of Expertise (AoE)
Global Product Management
Key Account Manager (KAM)
Bid Support Specialist (BSS)
Coordinator
Messaging
Memory

Fig. 10.5 The request for information process at Sony Ericsson

market point of view and decides which products shall be considered when dealing with the RFI. The BSS then passes the documents on to the coordinator, who analyzes the RFI and accompanying instruction and then distributes relevant parts of the RFI to Areas of Expertise (AoE). An AoE consists of a Function Group (FG) and a Technical Work Group (TWG). The TWG works with roadmaps (i.e. future functions) and the FG works with implementation and testing. When the AoEs have stated the compliance to each requirement, they send the RFI reply back to the coordinator. He reviews the answers and sends the replies on to the Bid Support Specialist, who also checks the answers. If the RFI originates from a major operator, a meeting is held with Global Product Management, the coordinator, and experts from the AoE in order to discuss the answers which are to be submitted back to the operator. The RFI reply is then sent back to the operator by the KAM.

The RFIs play a important role in the operator's strategic planning. The RFIs also provide SEMC with vital business intelligence information as the features prioritized by the operators may be used as a guideline when developing future phones. The operators thus have a great deal of influence on the final requirements for a product and a good relationship with the operators, based on timely and correct replies to the RFI's, is therefore of utmost concern.

The efficiency of the RFI process, in which requirement are analyzed and checked against product features, is however severely impeded. The AoE are concerned with their primary assignment in development and testing and have trouble finding the time required to analyze the RFIs. Furthermore, they get particularly frustrated as they have to state the compliance to the same or very similar requirements over and over again. Large parts of new versions of RFIs arriving from the same operator are typically the same as previous versions. Unfortunately, the revision history of the operators requirements cannot be trusted as there have been cases where requirement IDs have been reused and where requirements have been changed without indication. Current requirements management tools give no automated assistance in merging thousands of requirements. Furthermore, it is often the case that the same and very similar requirements occur in the RFIs from different operators. Consequently, there is much unnecessary redundant work required by the AoE.

As the RFIs are written in natural language, we have investigated the possibility of providing automated support to the RFI process using linguistic engineering techniques in order to find similar and related requirements. When RFIs arrive they are converted into a standardized format, where atomic requirements may be identified using unique identifiers. Of course, there is a desire to get the operators to use a standardized format when submitting their RFIs. The manual conversion step could then be removed and more time could be saved –which is of mutual interest for the operators. The standardized RFI is matched against a database of previous RFIs, which have been analyzed for compliance. For each requirement in the RFI, matches are provided based on a similarity measure. The KAM or the Bid Support Specialist may then mark the new RFI requirements as duplicate or similar, or not at all. The RFIs may then be passed on to the AoEs as before, but this time the AoEs only have to check those that are marked as similar or not marked at all. The hypothesis is that it is quicker to judge how similar two requirements are, than to reanalyze each for compliance. Additional benefits are automatically provided through this process:

- All business intelligence is gathered in one place.
- Similarities between different operator requirements may be identified and maintained.
- Contradictions between different operators requirements may be identified more easily.

At the time of writing, a central repository has been put in place comprising approximately 11,000 previously collected requirements. The goal is now to give support in the RFI process as explained above. Furthermore, it has been suggested that the similarity measuring techniques are used to clean the repository by identifying duplicates as described in Case 1.

10.7.1 Evaluation

The technique of using lexical similarity for matching incoming requirements to those already in the repository is currently undergoing further investigations. A support tool, based on our prototype presented in [39] is being developed. The decision to go further was based on an initial pre-study, which is presented here.

At the time of this evaluation, the envisioned repository was unfortunately not yet in place. This put constraints on the number of requirements that could be used in the evaluations. Furthermore, due to the resource constraints at SEMC, there was no possibility to do a full experimentation with experts. SEMC could not allow the AoE to perform the same compliance check twice on the same set of requirements. Instead, indication on potential time to be saved using the proposed approach was made by comparing the work and performance of experts and non-experts and let the expert's judgment decide if the approach is worthwhile. For the case where requirement only were checked for similarity, an expert performed the

Table 10.5 Evaluation results from Sony Ericsson

Run	RFI	# reqs.	Manual (h)	# identical requirements	Semi-automatic (h)
1	A rev. 2	434	AoE: 20	175 (40%)	Non-expert: 8
	A rev. 1	242			
2	B rev. 2	63	Expert: 3		Expert: 2
	A rev. 2	434			

compliance check both manually and with automated supported, below referred to as the *semi-automatic approach,* in which a simple software script was used to calculate similarity between requirements sets and suggest, for each requirement in one set, the five most similar in the other set. The evaluation results are shown in Table 10.5 and discussed in the following.

Run 1. Two revisions of requirements from Operator A were selected to see to what extent the approach could save time by supporting an early process of sifting out similar requirements in order to reduce the redundant work currently required by the AoE. Revision 1 comprised 242 requirements that previously had been checked for compliance. Revision 2 comprised 434 requirements and was sent to an AoE for compliance check by experts. They were thus checking compliance on all requirements in Revision 2 (the revision history from the operator could not be relied upon). The non-expert used the semi-automatic approach and was within 8 hours able to identify that, compared to revision 1, 209 requirements were new, 50 were changed, and 175 were identical. Only 259 requirements would thus have required attention by the AoE. However, the AoE estimated that 65% of the requirements were identical or very similar to revision 1, implying that the KAM and BSS would likely only send 152 requirements to the AoE for compliance check. Assuming that experts within 8 hours (but likely quicker) would find 282 requirements to be already checked according to revision 1 and thus forward the remaining 152 requirements for compliance check, 5 hours could be saved (i.e. *20–(8+(152*20/434)) h*).

Run 2. Two revisions from two different operators were selected to see how the approach may support the experts to sift out requirements that had already been checked for compliance. Furthermore, the requirements were chosen to see how the approach performs on requirements originating from different sources and which are thus expected to be stated differently. In this case, an expert made both a fully manual comparison between the two requirements sets and a comparison supported by the semi-automatic approach. The evaluation suggests a 33% increase in performance, but as a learning effect is expected the exact figure should not be taken too seriously. However, a discussion with the expert revealed that the semi-automatic approach did give relevant support to be valuable in the process.

Given the full picture from the evaluation, knowledge in the requirements, and an understanding of the non-expert's lacking domain knowledge, it was concluded that it was worthwhile to proceed with further studies, which are now in progress. The lack of a streamlined, user-friendly support tool was identified as a performance killer when using the semi-automatic approach. The next logical step is thus the development of a tool that interactively supports the initial check for similar requirements. It has been estimated that 20% of the time spent on checking the products' compliance to the RFIs may be saved. Even a lower expectation motivates further investigations and improvements.

10.8 Conclusions

An increasing number of market- and technology-driven companies realize that requirements are better managed continuously, and therefore best stored, in larger repositories. Unfortunately, as indicated by the same companies' struggle with their requirements repositories, it seems that pure information management challenges are becoming increasingly apparent in large-scale requirements management. This may be an indication that currently available requirements management tools do not meet the demands. The presented approach of calculating similarity between requirements on a lexical level gives reasonably high accuracy, considering its simplicity. Most importantly, it provides added support to the management of large repositories of natural language requirements. The support is not aimed at replacing the current way of working, but to complement it in order to save time. The simplicity of the technique is a deliberate choice. As such, it is robust and requires no or little maintenance or attention, which is important for acceptance in industry. Minor adaptations may be required to align the techniques with the current tools used and with the requirements process. Still, our experience from the three case studies is that the major obstacle is on the implementation level as there are no ready solutions. Until commercial solutions are available, the cost of adopting the technique would correspond to a general in-house development project.

For research purposes, the presented evaluations acts as a baseline to which further research may be compared. Based on the three case studies we suggest an incremental improvement approach. Additional research must be made at a linguistic level, e.g. understanding how requirements are written and communicated, in order to fully understand the limitations and potential of linguistic engineering support. The current state is that twenty-five years of research in corpus linguistics has just recently and very briefly touched the new corpus of software requirements. The technique, as suggested by the three case studies, is relatively easy to implement and could be incorporated by most tool vendors. One vendor, Focal Points AB, already provide a "Find Similar"-functionality based on our earliest results, and our own recent prototype tool, ReqSimile, is freely available and may be adapted to fit the needs (http://reqsimile.sourceforge.net). In addition to the activities presented in the cases, the following other information intensive activities could be supported by the approach:

- Requirements tracing. For several purposes, a requirements traceability matrix should be maintained. A study by Hayes et al. suggests using similarity measuring techniques for easy "after-the-fact" requirements tracing [22].

- Defect tracking. As new defects are reported, a similarity check can help testers to identify if similar defects have been reported earlier, and avoid spending time on duplicate "bug reports".
- Support issues. Call center personnel browse support issues on a daily basis and could be supported by similarity measuring techniques.

Linguistic engineering techniques are widely used in information intensive support systems; for some reason most CASE tools excluded. The techniques are available and may be successfully adapted and further exploited. With the increase in the amount of information written in natural language that large software development companies need to manage, taking advantage of these techniques is definitely worthwhile.

References

1. Boehm BW (1976) Software engineering. IEEE Transactions on Computers 25:1226–1241
2. Boehm BW (1984) Verifying and validating software requirements and design specifications. IEEE Software 1(1): 75–88
3. Brooks FP, Jr. (1995) The mythical man-month: essays on software engineering, Addison-Wesley, Boston
4. Burg JFM (1997) Linguistic instruments in requirements engineering. Ph.D. thesis, Vrije Universiteit, Amsterdam, the Netherlands
5. Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, Los Alamitos pp.84–91
6. Cybulski JL, Reed K (1998) Computer assisted analysis and refinement of informal software requirements documents. In: Proceedings of 1998 Asia-Pacific Software Engineering Conference, Los Alamitos, pp.128–135
7. Cybulski JL, Reed K (1999) Automating requirements refinement with cross-domain requirements classification. In: Proceedings of the 4th Australian Conf on Requirements Engineering. Macquarie University, Sydney, pp 131–145
8. Cyre WR, Thakar A (1997) Generating validation feedback for automatic interpretation of informal requirements. Formal Methods in System Design 10: 73–92
9. Daly EB (1977) Management of software development. IEEE Transactions on Software Engineering 3:229–242
10. Davis AM, Jordan K, Nakajima T (1997) Elements underlying the specification of requirements. Annals of Software Engineering 3: 63–100
11. Fabbrini F, Fusani M, Gervasi V, Gnesi S, Ruggieri S (1998) On linguistic quality of natural language requirements. In: Proceedings of the 4th Workshop on Requirements Engineering: Foundations for Software Quality, Les Presses Universitaires de Namur, Namur, pp 57–62
12. Fabbrini F, Fusani M, Gnesi S, Lami G (2001) The linguistic approach to the natural language requirements quality: benefits of the use of an automatic tool. In: Proceedings of the NASA Goddard Space Flight Center Software Engineering Workshop, Los Alamitos, pp.97–105

13. Fantechi A, Gnesi S, Lami G, Maccari A (2003) Application of linguistic techniques for use case analysis. Requirements Engineering 8: 161–170
14. Fliedl G, Kop C, Mayr HC (2003) From scenarios to KCPM dynamic schemas: aspects of automatic mapping. In: Proceedings of the 8th International Conference on Applications of Natural Language to Information Systems. Bonn, Germany, pp.91–105
15. Francis WN, Kucera H (1982) Frequency analysis of English usage: lexicon and grammar. Houghton Mifflin, Boston
16. Fuchs NE, Schwertel U (2003) Reasoning in Attempto Controlled English. In: Proceedings of the International Workshop on Principles and Practice of Semantic Web Reasoning, Lecture Notes in Computer Science 2901, pp.174–188
17. Garcia Flores JJ (2004) Linguistic processing of natural language requirements: The contextual exploration approach. In: Proceedings of the 10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality, Riga, Latvia, Essener Informatik Beiträge, 7-8 June
18. Garigliano R (1995) JNLE editorial. Natural Language Engineering, 1:1–7
19. Gervasi V, Nuseibeh B (2002) Lightweight validation of natural language requirements. Software: Practice & Experience 32: 113–133
20. Gervasi V, Zowghi D (2005) Reasoning about inconsistencies in natural language requirements. ACM Transactions on Software Engineering and Methodology (to appear)
21. Goldin L, Berry DM (1997) AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Engineering 4: 375–412
22. Hayes, JF, Dekhtyar A, Sundaram SK, Howard S (2004) Helping analysts trace requirements: an objective look. In: Proceedings of the 12th IEEE International Requirements Engineering Conference, Los Alamitos, pp.249–259
23. Hearst M (1995) TileBars: visualization of term distribution information in full text information access. In: Proceedings of the ACM SIGCHI Conf on Human Factors in Computing Systems, ACM Press, New York, pp.59–66
24. Höst M, Regnell B, Natt och Dag J, Nedstam J, Nyberg C (2001) Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. Systems and Software 59: 323–332
25. Jackson M (1995) Requirements and specifications: a lexicon of software practice, principles and prejudices. Addison-Wesley, New York
26. Jackson P, Moulinier I (2002) Natural language processing for online applications: text retrieval, extraction and categorization. John Benjamins, Amsterdam
27. Kilgariff A (2001) Comparing corpora. Int J Corpus Linguistics 6: 97–133
28. Kristensson P, Magnusson P, Matthing J (2002) Users as a hidden resource for creativity, findings from an experimental study on user involvement. Creativity and Innovation Management 11: 55–61
29. Landauer TK, Foltz PW, Laham D (1998) An introduction to latent semantic analysis. Discourse Processes 25: 259–284
30. Lauesen S (2002) Software requirements: Styles and techniques. Addison-Wesley, UK
31. Macias B, Pulman SG (1993) Natural language processing for requirement specifications. In: Safety Critical Systems, Redmill F, Anderson T (Eds). Chapman and Hall, London, pp.57–89
32. Macias B, Pulman SG (1995) A method for controlling the production of specifications in natural language. The Computer Journal 38: 310–318

33. Manning CD, Schütze H (2001) Foundations of statistical natural language processing. MIT Press, Cambridge
34. Mich L, Franch M, Novi Inverardi P (2004) Market research for requirements analysis using linguistic tools. Requirements Engineering, 9: 40–56
35. Mich L, Garigliano R (2002) NL-OOPS: A requirements analysis tool based on natural language processing. In: Proceedings of the 3rd International Conference on Data Mining. WIT Press, Wessex, pp 321–330
36. Miller GA (1956) The magical number seven, plus or minus two: some limits on our capacity for processing information. The Psychological Review 63: 81–97
37. Minnen G, Carroll J, Pearce D (2001) Applied morphological processing of English. Natural Language Engineering, 7: 207–223
38. Nanduri S, Rugaber S (1996) Requirements validation via automated natural language parsing. Management Information Systems, 12: 9–19
39. Natt och Dag J, Gervasi V, Brinkkemper S, Regnell B (2005) A linguistic engineering approach to large-scale requirements management. IEEE Software 22(1):(to appear)
40. Natt och Dag J, Regnell B, Carlshamre P, Andersson M, Karlsson J (2002) A feasibility study of automated natural language requirements analysis in market-driven development. Requirements Engineering, 7(1): 20-35
41. Osborne MC, MacNish K (1996) Processing natural language software requirements specifications. In: Proceedings of the 2nd International Conference on Requirements Engineering, IEEE CS Press, Los Alamitos, pp.229–236
42. Park S, Kim H, Ko Y, Seo J (2000) Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. Information and Software Technology 42: 429–438
43. Porter MF (1980) An algorithm for suffix stripping. Program, 14: 130–137.
44. Rayson P, Emmet L, Garside R, Sawyer P (2000) The REVERE project: experiments with the application of probabilistic NLP to systems engineering. In: Proceedings of 5th International Conference on Applications of Natural Language to Information Systems, Lecture Notes in Computer Science 1959. Springer-Verlag, Heidelberg, pp.288–300
45. Rolland C, Proix C (1992) A natural language approach for requirements engineering. In: Proceedings of the 4th International Conference on Advanced Information Systems Engineering. Lecture Notes in Computer Science 593, pp.257–277
46. Ryan K (1993) The role of natural language in requirements engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering. Los Alamitos, pp.240–242
47. Sawyer P, Cosh K (2004) Supporting MEASUR-driven analysis using NLP tools. In: Proceedings of the 10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality Riga, Latvia, Essener Informatik Beiträge, 7-8 June
48. Siegel S, Castellan NJ, Jr. (1988) Nonparametric statistics for the behavioral sciences, 2nd edition. McGraw-Hill, Singapore.
49. Sommerville I (2001) Software Engineering, 6th edition, Pearson Education, Harlow
50. Somé S, Dssouli R, Vaucher J (1996) Toward an automation of requirements engineering using scenarios. J Computing and Information, 2: 1110–1132
51. Sutton DC (2000) Linguistic problems with requirements and knowledge elicitation. Requirements Engineering 5: 114–124

52. Wieringa R, Ebert C (2004) RE'03: Practical requirements engineering solutions. IEEE Software 21(2): 16–18

53. Wilson WM, Rosenberg LH, Hyatt LE (1996) Automated quality analysis of natural language requirement specifications. In: Proceedings of the 14th Annual Pacific Northwest Software Quality Conference, Portland, pp.140–151

Author Biography

Johan Natt och Dag is a Licentiate Engineer in Software Engineering at the Department of Communication Systems of Lund University, Sweden. His main interests are in requirements engineering, software product management, software quality, and usability engineering. He received an MSc in Computer Science and Technology from Lund Institute of Technology. He is a member of the ACM and of the Swedish Requirements Engineering Research Network.

Vincenzo Gervasi is a Research Associate at the Dipartimento di Informatica of the University of Pisa, Italy, where he is a member of the Software Engineering group, and Honorary Associate at the Faculty of Information Technology of the University of Technology, Sydney. His main interests are in requirements engineering, natural language processing, specification techniques, and design and evaluation of distributed algorithms. He received his MSc and PhD in Computer Science from the University of Pisa.