Extracting *Visions* from Textual Requirements Documents

Miriam Sayão[1] **and Carlos A. Prolo**[1]

[1]Faculdade de Informática, PUCRS,
Av. Ipiranga, 6681, prédio 32, sala 505
90619-900 Porto Alegre, RS, Brazil

{miriam,prolo}@inf.pucrs.br

Abstract. *Software requirements are frequently written as a document in natural language. For large systems it is difficult to index them so that we can extract information concerning a particular point of view of interest. Indeed they frequently come with no interesting indexing system. In this paper we investigate the use of information retrieval techniques based on natural language processing to produce indexing information for requirements documents written in natural language. We focus on two aspects: one is how to retrieve the collection of requirements related to a particular user-oriented view of interest. The other is how to produce a suitable indexing system for the requirements document, which an expert software engineer would see as a general and natural way of accessing the document for purposes ranging from understanding the system to maintenance, to requirements verification and validation.*

1. Introduction

When building a software system one initial step is to define the *System Requirements Specification* (*SRS*) [Sommerville 2001], a document containing the requirements the software will have to attend. These requirements are usually written in natural language in one of several standards ways, such as *requirement sentences* – a piece of text in natural language containing one or more sentences – or the more structured *use cases* [Quatrani 1998], among many others [Sommerville 2001].

The resulting document generally has some internal structure, that reflects the view of the *software engineer*, which is not necessarily the view of the clients or users. Moreover, the same stakeholder may actually have different structuring views at different moments, depending on their particular purpose at that moment. Later in the development stage and when it is effectively in use one may want to scrutinize over the document either manually or automatically to retrieve information in systematic ways. The reasons range from simply trying to understand what the system is about to verification/validation (V&V) activities. In these cases we want to organize the requirements document according to what is covered in there or by pre-selecting a topic – which may even be found not to be covered in the document for the misfortune of the stakeholder.

A problem then arises when the document structure is not adequate to let the stakeholder to extract the particular information he wants at a particular moment. For instance sometimes the document is only split into functional and non-functional requirements, and one may want to obtain information concerning specifically *safety* of the system, or some particular functional issue, such as *flying tickets*, or *purchases*. When these documents are large, to obtain this information may be very hard. This is the problem we tackle in this paper.

Following [Palmer and Liang 1992], we call *vision* a subset of requirements that share characteristics or properties of the system. The vision may have an associated *term* or *index* that describe the commonality of the requirements in the subset, so that one can understand what the vision is about without having to read the requirements. An indexing system for the requirements document is a set of visions – that is, a set of subsets of requirements – with corresponding indexes.

Throughout this paper we will assume that each requirement is a linear sequence of natural language sentences. [1]

This paper describes ongoing work towards providing the stakeholders the following capabilities:

1. We want to give the stakeholders the ability to retrieve information about certain *views of interest*, each view being given by a natural language term, such as *safety* or *booking tickets*. That means, given a set of indexes, we want to generate the full corresponding indexing system.
2. We want to be able to automatically produce a suitable indexing system for the requirements document, which an expert software engineer, or even a client or user, would see as a general and natural way of accessing that document. There are two possible ways of doing this. One is to first automatically find an appropriate set of indexing terms, and from these, to generate the indexing system, as above. The other is to apply *clustering* techniques from the IR literature [2] to directly define the visions (i.e., the subsets of requirements). And then in a subsequent step we might try to infer the corresponding indexing terms, if we are lucky enough that such terms exist.
   A very relevant issue in the second approach has to do with the restrictions to the kind of clustering required. Each cluster of requirements should cover one relevant theme of the document. Therefore, unfortunately, the set of clusters cannot be a partition, since requirements may carry information regarding more than one theme, so clusters are not to be disjoint.

[1] We assume that whatever standard has originally been used to write the requirement, it can be converted in such a list of sentences, with some possible loss in meaning. We will not discuss this point further in this paper.

[2] In the IR terminology, clustering is the process of grouping documents in related subsets. If we see each of our requirements as a document, we can apply clustering techniques to structure our requirements document into a set of visions.

3. One may claim that the previous capabilities have inherent problems from their conception. There might not be a unique set of clusters, or equivalently a unique set of topics that best characterize the document. Indeed we believe there is not. It is conceivable that for different stakeholders different organizations of the document may be more appropriate. For instance, the granularity problem. We may have sets of topics with a few general terms or larger ones, more granular with more specific terms. Although there are many possible issues one can rise on behalf of this discussion, we focus here on the capability of iteratively (and interactively) generating the organization of the document. We want the stakeholder to be able to start with an automatically generated set of clusters and respective topics – from capability 2 above, – and then to progressively adjust the outcome by including/removing topics iteratively. Notice that the iterative steps are therefore applications of capability 1.

At first sight one can think that the solution is merely to use the standard techniques already described in the Information Retrieval (IR) literature. Or perhaps even using tools already available. Capability 1 is studied in the IR literature under the name of *text categorization* (or *text classification* or *document categorization* [3]. Capability 2 involves *text/document clustering*, if we obtain the sets of clusters directly; *information extraction*, to obtain the terms either from the whole document of requirements or from the clusters; and again, *text/document categorization*, if we find the clusters from the indexing terms. In the rest of this introduction we motivate our research arguing that things are not so simple. In subsequent sections we describe the experiments we have tried so far to achieve our stated goals.

We can not use general classifiers: One could look for already existent general classifiers and apply them to the set of requirements. That does not work for two main reasons. The indexing terms appropriate for each requirements document is certainly different from each other and hence different in general from any pre-existing classifier. Secondly, the meaning of the indexing terms in each system is particular to that system. For instance, in general document classifiers, one would expect the term *safety* to be associated with *jail*, *robbery*, *gates with remote control*. Conversely in some software systems it could be attached to a requirement that the system do not lose integrity in case a transaction is aborted in the middle.

We can not use supervised learning: Most of the work on efficient text categorization is supervised learning [Sebastiani 2002]. A classifier is automatically induced from a large amount of data annotated with the correct answer. Well we do not have that. We are given as input a requirements document of a particular system each time. Of course we

[3]In our context the word *document* in the expressions *document categorization* and *document clustering* refers to a single requirement – not to be confused with the "document" that contains all the requirements.

cannot count on having a pre-classified subset of requirements. And neither can we pre-classify one or a set of requirement documents to use as training data, since the resulting classifier would not be appropriate for any other system as has been just argued. Now, work on unsupervised learning is much harder, the results are not nearly as effective as supervised learning and the literature is scarcer.

We need overlapping clusters: If on the one hand the work with text categorization has been mostly supervised, on the other hand, clustering algorithms, which are typically unsupervised, generally generate a partition as outcome, that is, the clusters are non-overlapping. But as we argued before, requirements can be related to more than one topic of interest. Hence we need non-overlapping text clustering algorithms.

2. Related Work

There are very few works on this subject. [Palmer and Liang 1992] first pointed out the need for grouping requirements for V&V activities. They idealized a clustering algorithm for verbs, called Two Tiered Clustering (TTC), focusing therefore on actions. The algorithm starts with the set of verbs in the document and uses a thesaurus composed by verbs extracted from previous work on an operating systems project. It was as early attempt involving a lot of manual work. No quantitative results were provided.

The works of [Hsia and Gupta 1992] and, more recently, [Chen et al. 2005], were based on looking at shared *resources*. [Hsia and Gupta 1992] manually identifies *Abstract Data Types (ADT)* corresponding to objects which are referred to in the requirements (either as modified or accessed). Requirements are then grouped based on the sharing of these ADT's. They were aiming at incremental software development, and the clusters were intended to point out the increments. The approach requires an intertwining of the specification aspects (requirements definition) with the development (design) phase, where the ADT's belong to. [4]

In [Chen et al. 2005] an undirected graph is built in which the requirements are the nodes. They identify five types of relationships between requirements based on how they manipulate shared resources. Each kind of relationship is assigned a weight. For any pair of requirements in one of these relationships, an edge is added to the graph, with the weight of the relationship. The clusters are then defined based on connectivity among nodes, taking the weights of the edges into account. It is a semi-automatic approach. The authors are aware of the need of human intervention. There is no mention on how the resources are identified. Apparently they are manually defined from obvious functional objects of the system (e.g., *checkout lists, books, users,* ...). Clearly a human has to judge which relationship applies to each pair of nodes.

Besides the particular limitations of each individual approach, they share one

[4] In their case they had actually to bring the ADT's definition forward to the requirements phase.

drawback: they attend only developer needs. They are not available to the other stakeholders, in the sense that they can not impose their point of view.

3. Experiments [5]

The experiments we discuss in this paper have been run over a large requirements document from a well-established commercial software company (Tlantic Sistemas de Informação). The document is for an information system in the area of tourism, separating functional and non-functional requirements. There are 169 requirements, four of these being non-functional, amounting to 29.554 words – average of about 180 words per requirement. The document is written in European Portuguese.

3.1. Experiment 1: A Simple Way to Classify the Documents Given a Set of Indexing Terms of the Requirements Document

A simple way to accomplish with Capability 1 is, given an input term – that is, a word sequence – to retrieve all the requirements that contain that term. In order to prevent undesirable mismatches due to morphological differences between words due to verb inflection, or noun features such as number and gender, we apply stemming to both the indexing term and the requirement words prior to comparing. [6]

The above process is rather naive. However, as we have mention in the previous section, the traditional, significantly more efficient, algorithms for learning classifiers are obtained throw supervised learning, what cannot be done here.

Notice that we binarily classify the requirements for each indexing term, so that a single requirement may appear in more than one class.

3.2. Experiment 2: A Simple Way to Find an Appropriate Set Indexing Terms of the Requirements Document

We ranked the words appearing in the requirements document according to three criteria: *frequency*, *log-likelihood*, and χ^2 [Manning and Schütze 1999] – one ranked list per criterion. [7] Only the top 30 words were kept in each list. The resulting set was manually scrutinized to eliminate words not relevant to particular topics (*stop words* for the domain) and a final set of 5 was chosen. Over the process of testing, additional terms were being added by empirical observation, namely non-functional terms, in the spirit of Capability 3, that is, that the stakeholder interferes with the process iteratively. Table 1 shows the terms used.

[5] In the experiments we used [Orengo and Huyck 2001]'s *stemmer* and PreText [Matsubara et al. 2003] to generate the document representation as *bag of words*.

[6] Stemming also provides a unique representation for words of different syntactic categories – e.g., a noun and a verbal form – which are seen to derive one from the other, such as *sell* and *sale*. Alternatively one could suggest using a lemmatizer instead, which would preserve the distinction between the syntactic categories.

[7] We could have also generated lists for bigrams, or trigrams to obtain multiple-word terms, although we did not do so here.

comunicação	automatically generated
pacote	automatically generated
pagamento	automatically generated
segurança	manually added
sistemas externos	manually added
viagem	automatically generated
parametrização	manually added
reserva	automatically generated

Table 1. Indexing Terms

3.3. Experiment 3: Directly Clustering the Documents

The idea here is to obtain an adequate set of visions for the requirements document without first obtaining the indexing terms as we discussed in Capability 2. The reason for doing this is that it is clearly hard to automatically obtain a good set of indexing terms. And a bad quality of such initial set compromises the visions found in a subsequent step. On the other hand there are plenty of available unsupervised learning algorithms for text clustering that can be used to directly obtain the sets of visions. We applied the *k-means* algorithm [Manning and Schütze 1999] available among the *Weka* tools [Witten and Frank 2000]. Each requirement is first converted into a *bag-of-words* representation: a high-dimensional vector where each word w is a dimension, and the coordinate value at that dimension is the number of occurrences of w in the requirement text. It is important to note that: (1) words are actually *stems*, what greatly reduces dimensionality; (2) all vectors contain exactly the same dimensions, namely all the stems appearing in any of the requirements; (3) however irrelevant words are excluded from the dimensions, using a *stop list*. The *stop list* contains very common words (i.e. with high frequency in the texts), typically from closed categories such as articles and prepositions, whose inclusion serves only to confuse the algorithms, since they are not relevant to take any sensible decision concerning clustering. Following this spirit, additional words may be added in an interactive process, by the stakeholder, which are irrelevant and very frequent in the particular domain.

Once the requirements have been converted into vectors, the *k-means* algorithm works as follows to obtain a set of k clusters. Initially a set of k points (vectors) are randomly chosen as clusters centroids. Then each requirement (the corresponding vector) is assigned to the cluster whose centroid is the closest to it. Then the centroid of each cluster is re-evaluated. The process repeats until the clusters converge to a stable partition.

A drawback of this approach is that we have to provide as input the number k of clusters having no clue at all of what could be the desired granularity. Recall that although a similar such decision has to be made regarding the number of relevant terms that are to be used when the terms are determined first, in that case we can be guided by looking at

the terms and evaluating their relevance.

3.4. Experiment 4: A More Elaborate Way of Classifying the Requirements

We start with an initial set of indexing terms, which will, at the end, determine the generated subsets of requirements. For each term t, we generate a list of word contexts, that tend to appear in the document together with t forming a relevant collocation. We can think of this list, as defining relevant contexts for the occurrence of t in the requirements document. The terms with their associated lists compose a hierarchical, two-level indexing structure. We are actually interested in the first level of the original terms only. The second level is intended to help improve precision of categorization by specifying more specific relevant contexts for the use of the word t.

For each term t we obtain an initial set of collocation candidates as follows. For each occurrence of t in the document, we extract the sequence of words starting five words before and ending five words after t – a *concordance*. Now let $c = l_5\ l_4\ l_3\ l_2\ l_1\ t\ r_1\ r_2\ r_3\ r_4\ r_5$ be one such concordance. A collocation candidate is either a pair $< s_l, t >$, where s_l is a substring of $c = l_5\ l_4\ l_3\ l_2\ l_1$, or a pair $< t, s_r >$, where s_r, is a substring of $r_1\ r_2\ r_3\ r_4\ r_5$. Actually, the word components in these collocation candidates are converted to their stem. Each candidate is then evaluated with respect to its relevance in the requirements document using the *T-score* measure and *mutual information* [Manning and Schütze 1999].

$$T(x,y) = \frac{freq(x,y) - \frac{freq(x)*freq(y)}{N}}{\sqrt{(freq(x,y))}}$$

$$mi(x,y) = log_2(\frac{P(x,y)}{P(x)*P(y)})$$

$Freq(x)$, $freq(y)$ and $freq(x,y)$ are the occurrence frequencies of terms x and y and the of their joint occurrence. $P(x)$, $P(y)$ and $P(x,y)$ are probability estimates obtained from the requirement document.

Finally, using the *stop list*, irrelevant words are removed from the candidates considered relevant. The resulting sequences form the list collocations for the term t, which is the second level of the hierarchical index structure.

Once the two-level structure is generated we are able to classify requirements as follows. We insert the requirement in the category for a term t, if the requirement text matches one of the collocations. The text matches a collocation of the form $< s_l, t >$, if it contains a subsequence of the form $s_l\ ?\ t$, where ? is a sequence of up to four words. Collocations involving s_r are worked out similarly. (These same matching criteria is also used for counting $freq(x,y)$ above.)

Indexing Term	Precision	Recall	$F_{\beta=1}$
pacote	0.42	0.31	0.36
pagamento	0.88	1.00	0.94
viagem	0.98	1.00	0.63
reserva	0.56	0.63	0.59

Table 2. Evaluation of Experiment 1

Indexing Term	Precision	Recall	$F_{\beta=1}$
pacote	0.55	0.26	0.35
pagamento	0.71	0.69	0.69
viagem	0.80	0.66	0.72
reserva	0.62	0.38	0.47

Table 3. Evaluation of Experiment 4

4. Partial Empirical Evaluation

We first generated a set of indexing terms running the Experiment 2 above. We asked a computer scientist familiar with software engineering to generate an annotation table for the document where each line was a requirement and the columns were the indexing terms. Each cell would be marked *yes* in case the requirement was judged to be related to the indexed term, or *no*, otherwise. This became our *gold standard*.

Then we ran the simple algorithm of the Experiment 1, obtaining the accuracy results of Table 2. The figures for precision and recall have been obtained in the standard ways, as below. $F_{\beta=1}$ is their harmonic average. Table 3 presents the results for the more elaborate experiment 4.

$$Precision = \frac{Number\ of\ requirements\ correctly\ retrieved\ for\ the\ term}{Number\ of\ requirements\ retrieved\ for\ the\ term}$$

$$Recall = \frac{Number\ of\ requirements\ correctly\ retrieved\ for\ the\ term}{Number\ of\ requirements\ associated\ to\ the\ term}$$

$$F_{\beta=1} = \frac{2 * Precision * Recall}{Precision + Recall}$$

Unfortunately we did not see improvements in precision as we were expecting in experiment 4.

5. Conclusions and Future Work

We tackle a hard problem of applied information retrieval using natural language processing methods. Traditional supervised algorithms are not applicable. In this initial work we tried a few approaches which provide a baseline for accuracy results. Some experiments have been quantitatively evaluated.

With respect to the clustering algorithm we could not find a sensible way to evaluate it. A rather serious problem is that the usual clustering algorithms generate a partition, i.e., non-overlapping clusters. That is the case of the *k-means* and all other algorithms for clustering provided at *Weka*.

When trying to access the clusters and tell what they were about, we could not make sense of the clusters as referring to reasonable topics of interest, that is, we were not successful in finding reasonable index terms. We believe this may be, at least partly, due to the algorithm being non-overlapping, while the subsets of requirements are largely overlapping w.r.t. to reasonable sets of indexing terms.

A main direction for future work is to look for non-supervised learning and non-standard techniques applicable to our problem.

References

Chen, K., Zhang, W., Zhao, H., and Mei, H. (2005). An approach to constructing feature models based on requirements clustering. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 31–40, Paris, France.

Hsia, P. and Gupta, A. (1992). Incremental delivery using abstract data types and requirements clustering. In *Proceedings of the Second International Conference on Systems Integration*, pages 137–150, Morristown, NJ, USA.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, USA.

Matsubara, E. T., Martins, C. A., and Monard, M. C. (2003). Pretext: Uma ferramenta para pré-processamento de textos utilizando a abordagem bag-of-words. Technical Report 209, ICMC-USP, São Carlos, Brazil.

Orengo, V. M. and Huyck, C. R. (2001). A stemming algorithm for the portuguese language. In *8th International Symposium on String Processing and Information Retrieval (SPIRE'2001)*, pages 186–193, Laguna de San Rafael, Chile.

Palmer, J. D. and Liang, Y. (1992). Indexing and clustering of software requirements specifications. *Information and Decision Technologies*, 18(4):283–299.

Quatrani, T. (1998). *Visual Modeling with Rational Rose and UML*. Addison-Wesley, Reading, MA, USA.

Sebastiani, F. (March, 2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1).

Sommerville, I. (2001). *Software Engineering*. Addison-Wesley, Boston, MA, USA.

Witten, I. H. and Frank, E. (2000). *Data mining: practical machine learning tools and techniques with java implementations*. Morgan Kaufmann, San Francisco, CA, USA.