

# Informal Requirements Analysis Supporting System for Human Engineer

Harksoo Kim, Youngjoong Ko, Sooyong Park, Jungyun Seo  
Department of Computer Science, Sogang University  
Seoul, 121-742, Korea

{ hskim, kyj }@nlpzodiac.sogang.ac.kr, sypark@ccs.sogang.ac.kr, seojy@ccs.sogang.ac.kr

## ABSTRACT

As software becomes more complicated and large-scaled, user's demands become various and the expectation level about a software product becomes higher. Therefore, it is very important that a software engineer analyzes user's requirements precisely and applies them effectively in the development step. This paper proposes a requirement analysis supporting system that effectively reduces and revises errors of requirements analysis. The proposed system measures the similarity between requirements specifications or sentences. It extracts the sentences that contain ambiguous words. An indexing method for the similarity measurement combines a sliding window model and a parser model. This method can complement each model's weak points. Using these methods in information retrieval, the proposed system supports a function to trace dependency between documents, improve completeness in a document, reduce inconsistency between sentences, and improve document quality. This paper verifies the efficiency of the proposed system in similarity measurement techniques through experiments, and presents a process for requirements specifications analysis using the system.

## 1. INTRODUCTION

Requirements analysis plays an increasingly important role in software development because most problems are caused by an incorrect analysis of customer's requirements. Some specific reasons why a correct understanding of customer requirements include

- errors in system level requirements that are not corrected early in the software development life cycle ultimately end up as maintenance problems in the delivered software. Recent studies show that while we expend 5% of the total cost of a major system on design and development, 70% of the ability to influence the quality comes from this meager amount[20].
- the total system failure is often caused by the misunderstanding of requirements between a customer and an analyst.
- the later in the development life cycle that a software error is detected, the more expensive it will be to repair.

In terms of percent of effort expended, requirements analysis consumes merely 5% of the total cost of the effort, while affording 50% of the leverage to influence improved quality. In terms of pre-production costs, system definition absorbs 5% of these costs and provides 50% of the leverage to influence the improvement of quality[20]. Last, many errors remain latent and are not detected until well after the stage at which they are made because a software system has become larger and more complex. If it was possible to better understand how to detect flaws during the systems level requirements phase, there could be extensive savings in development cost and time.

Software requirements often have some additional problems like imprecision, conflict, inconsistency, ambiguity, and incompleteness because they are generally written in a natural language format, and are affected by the characteristic of an analyst. If these problems are handed over to the next phase without being resolved, they may generate other critical problems because the next phase of the software development cycle is based on the current one. In the worst case, we may have to totally redesign and reconstruct the system. So, it is very important that we check dependency and consistency between requirements, and check for conflicts and ambiguity. In this paper, we propose an efficient requirements analysis supporting system in Korean, which is based on similarity measures in information retrieval. The system can alleviate some critical problems such as conflict, inconsistency, and ambiguity.

This paper begins with a survey of related works in section 2. In section 3, we propose a requirements analysis system and explain an indexing method and similarity measuring methods. In section 4, we report some promising experimental results. After showing the implementation of our system in section 5, we draw some conclusion in section 6.

## 2. RELATED WORKS

There has been extensive work in software engineering to build automatic requirements analysis supporting systems[9][12][13][14][16]. These systems can be roughly classified into two groups according to the approach adopted, the free-text indexing approach as defined in information retrieval (IR), and the knowledge-based approach as defined in artificial intelligence (AI)[9]. The AI approach can be useful in some applications. However, the IR approach is generally preferred for the following reasons:

- Cost: the system is built entirely automatically
- Transportability: the system can be rebuilt for a domain, since it requires manually provided domain knowledge.
- Scalability: the repository can be easily updated when new components are inserted.

Traditional requirements analysis supporting systems have used similarity measures and keywords extracted from the specifications. Palmer proposed the TTC(Two-Tired Clustering) algorithm[13] which indexes and clusters requirements specifications. It, first, identifies the verbs and keywords of each requirement, then it clusters these requirements by functionality. Next, each cluster is subdivided using cosine similarity between clusters. To automatically construct software libraries, Maarek proposed the method to index and cluster UNIX manual pages, which uses a sliding window and a level clustering technique[13]. Many researchers in information retrieval have focused on indexing documents and measuring similarity between documents using MRD(Machine Readable Dictionary) like thesaurus or a linguistic feature like reiteration and collocation of words[2][3][5][6][7][8][22]. However, methods using MRD or

linguistic features have some problems. They do not give an analyst efficient information like conflict, inconsistency, and ambiguity between requirements in a specification. Although methods using thesaurus outperform others in precision, building MRD is difficult, domain specific and time-consuming. Although there has been lots of work on indexing and clustering in information retrieval, we can not easily find a system based on them.

### 3. INDEXING AND MEASURING SCHEMES

#### 3.1 An overview of the requirements analysis supporting system

A requirements analysis supporting system consists of three modules: one to measure similarity between documents, one to measure similarity between sentences and one to extract ambiguous sentences. Measuring similarity between documents is useful for tracing dependency between mixed specifications in a distributed developing environment. The module consists of two sub-modules as shown in Figure 1.

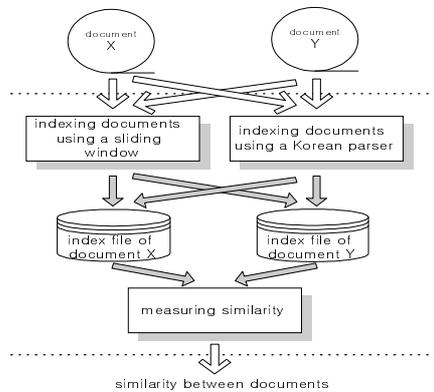


Figure 1: The module to measure similarity between documents

The sub-module to index documents extracts word pairs including co-occurrence information by the use of a sliding window and a parser using the syntactic dependency relations. The system gets simple co-occurrence information between words within a span of five words using the sliding window. The parser generates modifier-modified relations between word-phrases which are called dependency relations. The dependency relation supplements simple co-occurrence information extracted by the sliding window and covers co-occurrence information over a span of five words. The module to measure similarity between sentences is useful for checking consistency between a requirement in a high level specification and a requirement in a low level specification or between requirements in a specification. The module is similar in structure to Figure 1. However, algorithms to measure similarity have to be different because the number of indexes extracted from a sentence is much less than the number of indexes extracted from a document. Therefore, the system uses two different methods.

The module to extract ambiguous sentences finds inadequate sentences in a requirements specification. To support this function, the system makes use of a MRD and a POS(Part-Of-Speech) tagger as shown in Figure 2. The ambiguous words in the MRD are manually extracted from a lot of requirements specifications.

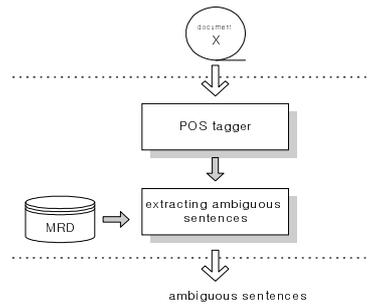


Figure 2: The module to extract ambiguous sentences

#### 3.2 The indexing scheme

There has been much work in IR dealing with natural language text: a large variety of techniques have been devised for indexing, classifying, and retrieving documents[17][18]. One of the main concerns in IR is the automatic indexing of documents, which consists of producing, for each document, a set of indexes that form a *profile* of the document. A *profile* is a short-form description of a document, easier to manipulate than the entire document, which plays the role of a surrogate at the retrieval stage. Another main concern in IR is the methods for generating a *profile* from a given document. The methods are generally divided into two types. One is a single-term index, each of which is a single word without contextual information. The other is a term-phrase index, which uses term phrase as indexing units rather than single terms so as to refine the meaning of constituent words. Although the former method can extract a lot of indexes from a small document, it can not provide a refined analysis including semantics. A possible solution to this problem is to use information such as part-of-speech or dependency relation in order to provide further control over phrase formation. In this paper, we propose an efficient term-phrase index scheme that uses a sliding window and a syntactic parser.

##### 3.2.1 The sliding window technique

A term-phrase index is derived from the notation of LA(Lexical Affinity). In linguistics, a syntagmatic lexical affinity, also termed a lexical relation, between two units of language stands for a correlation of their common appearance in the utterances of language[19]. The observation of LA's in large textual corpora has been shown to convey information on both syntactic and semantic levels and provides us with a powerful way of taking context into account[21]. Ideally, LA's are extracted from a text by paring it, since two words share a lexical affinity if they are involved in a modifier-modified relation. Unfortunately, automatic syntactic parser of a free-style text is still not very efficient, and the number of LA's extracted by parsing is not enough to measure similarity between documents or sentences if we do not have a large corpus. So, the system makes use of simple co-occurrence information extracted by a sliding window and supplement its flaws using a syntactic parser. It has been shown by Martin *et al.* that 98% of lexical relations relate words which are separated by at most five words within a single sentence[10]. Therefore, most of the LA's involving a word  $w$  can be extracted by examining the neighborhood of each occurrence of  $w$  within a span of five words (-5 words and +5 words around  $w$ ). In this paper, the system just extracts LA's within +5 words around  $w$  because modified words in Korean usually locate after modifiers. A sliding window technique[9] consists of sliding a window over the text and storing pairs of words involving the head of the window if it is an open-class

word<sup>1</sup> and any of the other open-class elements of the window. The window is slid word by word from the first word of the sentence to the last, the size of the window decreasing at the end of the sentence so as not to cross sentence boundaries, since LA's can not relate words belonging to different sentences. The window size being smaller than a constant, the number of extracted LA's is linear to the number of words in the document.

### 3.2.2 The parser using syntactic dependency relations

Although a sliding window technique extracts a lot of LA's, it can not extract LA's out of the boundaries of a fixed window. Dependency relations extracted from a text by parsing it can supplement the weak point because the distance between a modifier and a modified word is flexible from 1 to the length of a sentence. The system extracts LA's using a syntactic parser which is based on dependency grammar[4][11] as well as using the sliding window. Although the LA's extracted by the parser play an important role in extracting long distant relations, they have some problems as indexes. First, a dependency relation shows the relation between word-phrases rather than the relation between words. If a word phrase consists of some open-class words, dependency relations within a word-phrase are not extracted. For example, if “각각의 부서코드는 별도로 관리된다. (The codes of each department are separately managed.)” is analyzed using the parser, the noun phrase, *각각의 부서코드* (*The codes of each department*), depends on the verb phrase, *관리된다*. (*are managed*). However, the relation between *각각의 부서* (*each department*) and *코드* (*the codes*) is not extracted. The relation between *코드* (*the code*) and *관리된다*. (*are managed*) is also not extracted. Second, verbs in Korean are often generated using *a noun + do/become* or *a noun + be*. For example, *retrieve* is generated using *retrieval + do*. In those cases, a modifier usually depends on *do/become* or *be*. For example, if “신규코드가 입력되면, 관련 코드를 DB에서 검색한다. (When new codes are input, the system retrieves the DB associated with the codes.)” is analyzed using the parser, *DB* depends on *do* because *retrieve* is expressed as *retrieval + do*. However, the relation is not suitable for index because *do* carries less information than *retrieval*. The relation that *DB* depends on *retrieval* is more suitable for index.

1. The last content word in a modifying word-phrase depends on the last word in a modified word-phrase. For example, in “정보검색은 자연어처리분야이다. (Information retrieval is a part of natural language processing.)”, the last word *검색* (*retrieval*) in *정보검색* (*Information retrieval*) depends on the last word *분야* (*part*) in *자연어처리분야* (*a part of natural language processing*).
2. Each content word in a word-phrase depends on the next word in the phrase. For example, in “정보검색기법 (information retrieval scheme)”, *정보* (*information*) depends on *검색* (*retrieval*), and *검색* (*retrieval*) depends on *기법* (*scheme*).
3. If a modified word consists of a *noun + do/become/be* or a *noun + be*, modifier depends on the noun. For example, in “DB를 검색한다. (The system retrieves DB.)”, *DB* depends on *검색* (*retrieval*).

**Figure 3: Indexing heuristics**

<sup>1</sup> In general, open-class words include nouns, verbs, adjectives and adverbs, while closed-words are pronouns, prepositions, conjunctions, and interjections.

In this paper, we propose some heuristics to overcome these problems as shown in Figure 3.

### 3.2.3 Generating a profile from LA's

When analyzing a document, many potential LA's are identified. Some of these LA's are conceptually important and some are not. Although frequency of appearance is a good indicator of relevance, some noise exists, mainly due to words appearing too often in a given context like *be* and *do/become*. To reduce the influence of such words, it is necessary to select the most representative ones among LA's; i.e., those containing the most information among LA's. The *quantity of information* as shown in equation (1) is defined as a measure evaluating the resolving power of a word within a corpus[1][17].

$$INFO(w) = -\log_2(P\{w\}), \quad (1)$$

where  $P\{w\}$  is the observed probability of occurrence  $w$ .

Equation 1 means that the more frequent a word is in a domain, the less information it carries. If we consider words within the textual universe as independent variables to simplify the computation, the *quantity of information* of an LA can be approximated as shown in equation (2)[9].

$$\begin{aligned} INFO((w1, w2)) &= -\log_2(P\{w1, w2\}) \\ &\approx -\log_2(P\{w1\} \times P\{w2\}) \end{aligned} \quad (2)$$

Then, the resolving power of an LA in a given document is computed using the frequency of the LA in the document and the *quantity of information* of the LA as shown in equation (3). The resolving power is called  $\rho$ -score[9].

$$\rho((w1, w2, f)) = f \times INFO((w1, w2)), \quad (3)$$

where  $f$  is the frequency of the LA's.

To be able to compare the relative performances in terms of resolving power of different documents, the system transforms the  $\rho$ -score into a standardized score as shown in equation (4). In equation 4,  $\bar{\rho}$  is the mean of the  $\rho$ -scores, and  $\sigma$  is the standard derivation of the  $\rho$ -scores. The standardized score is called  $z$ -score[9].

$$\rho_z = \frac{(\rho - \bar{\rho})}{\sigma} \quad (4)$$

In this paper, the system extracts LA's using the sliding window and the parser and generates *profiles* including  $z$ -scores of these LA's. Each *profile* represents a requirements specification.

### 3.3 Measure of similarity between documents

In information retrieval, numerous measures of similarity between documents have been defined[15]. The simplest of all is defined as:

$$|X \cap Y|, \text{ where } X \text{ and } Y \text{ are the index file of two documents.} \quad (5)$$

This measure represents the number of common index units. The other measures like *Dice's coefficient*, *Jaccard's coefficient* and *Salton's cosine coefficient* are almost normalized version of equation 5[15]. These measures perform well in IR, but they are not applicable to the system because LA's have more information

than just the presence or absence of index units. Therefore, the system uses the measure like equation 6 because the equation can reflect the  $z$ -scores of LA's without information loss[9].

$$\delta(X, Y) = \sum_{i \in p(X) \cap p(Y)} (\rho X(i) \times \rho Y(i)) \quad (6)$$

In equation 6,  $p(X)$  means the *profile* of a document  $X$ , and  $p(Y)$  means the *profile* of a document  $Y$ .  $\rho X(i)$  means the  $z$ -score of  $i$ th index of  $p(X)$ , and  $\rho Y(i)$  means the  $z$ -score of  $i$ th index of  $p(Y)$ . In this paper, the system adds 1 point to each  $z$ -score and consider only LA's with the  $z$ -score which is greater than 0 in order to alleviate sparse data problems caused by small corpus.

### 3.4 Measure of similarity between sentences

To measure similarity between sentences, the system uses another measure of similarity, *Salton's cosine coefficient*[17], as shown in equation 7 because equation 6 is not applicable to measure similarity between sentences according to the following reasons: First,  $f$  in equation 6 means the frequency of LA's in a document. If the system counts  $f$  between sentences, the system can not get a meaningful value because the  $f$  is always around 1. Second, the number of LA's to measure similarity between sentences is much less than the number of LA's to measure similarity between documents. So,  $\bar{\rho}$ s, means of  $\rho$ -scores, are similar to each other, and  $z$ -scores have similar values as a result.

$$\cos(x, y) = \frac{|x \cap y|}{|x| \times |y|} \quad (7)$$

In equation (7),  $|x|$  is the number of LA's in sentence  $x$ , and  $|y|$  is the number of LA's in sentence  $y$ .  $|x \cap y|$  is the number of common LA's in sentence  $x$  and sentence  $y$ .

### 3.5 Extraction of ambiguous sentences

To extract ambiguous sentences from a document, the system makes use of a POS tagger and a MRD including ambiguous words which are extracted manually from a lot of requirement specifications. In processing, the system annotates a given document with POS tag. Then, it extracts verbs in the document and compares each verb with the words in the MRD. If a word in the document is same to a word in the MRD, the system displays the sentence including the word because the sentence has a high possibility of being ambiguous.

## 4. EXPERIMENTAL RESULTS

### 4.1 The experimental data

For the experiment, we collected requirements specifications from real fields. The data consisted of 33 documents with 1,090 sentences. We divided them into two test sets: One to evaluate the module to measure similarity between documents, the other to evaluate the module to measure similarity between sentences. The module to measure similarity between documents can be used to check dependency between requirements specifications in a distributed developing environment. So, we assumed such a situation and constructed the experimental data as follows. We divided a document into two sub-documents. We constructed *data-A* and *data-B* including each 33 sub-documents. In the experiment, the module measures similarities between a sub-document in *data-A* and all sub-documents in *data B*. If a sub-document in *data-B* has the highest rank, and the given sub-document in *data-A* and the sub-document in *data-B* were a same document before being separated, we regard it as the correct

answer.

The module to measure similarity between sentences can be used to check inconsistency between a high level sentence and a low level sentence. So, we divided the data into two groups by hand. One consisted of high level documents and the other, low level documents. In the experiment, the module measures similarities between a sentence in a high level document and all sentences in a low level document. If the module finds a correct low level sentence of the given high level sentence, we regard it as the correct answer.

To evaluate the module to extract ambiguous sentences, we used the same experimental data used for evaluating the measure of similarity between sentences. The MRD including ambiguous words was manually built by two graduate students.

### 4.2 The experimental result

To evaluate the module to measure similarity between documents, we compared three methods to extract LA's: a method using only the sliding window, a method using only the syntactic parser and the proposed integrated method. The precision in Table 1 means how many sub-documents in *data-B* are included in the given top rank with the correct answer as shown in equation (8).

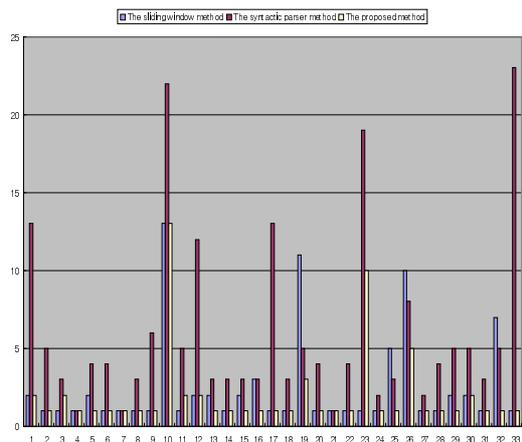
$$precision = \frac{\text{the number of correct answers in the given rank}}{\text{the number of sub - documents in data-B}} \times 100 \quad (8)$$

The table reveals that the proposed method is superior to the other methods.

**Table 1: The precisions of three methods**

Rank	The sliding window method	The syntactic parser method	The proposed method
Top-1	60.6 %	9.1 %	72.7 %
Top-3	81.8 %	42.4 %	90.9 %

The method using only the syntactic parser had much worse precision than the others. We believe that it is caused by the sparse data problem. Figure 4 shows the ranks of correct documents in *data-B* that are compared to each document in *data-A* using the proposed method.



**Figure 4: The ranks of correct documents**

In Figure 4, the proposed method ranked the correct answer of the 16th document first although the other methods ranked it

third. It reveals that the proposed method efficiently puts together co-occurrence information and dependency relation. If an analyst makes use of the proposed module, he/she can find a target document with a precision of 90% after looking up just three documents. It means that he/she can reduce his/her efforts by 9.4%.

To evaluate the module to measure similarity between sentences, we compared two indexing schemes: single-term indexing scheme and term-phrase indexing scheme. The term-phrase indexing scheme is superior to the single-term indexing scheme as shown in Table 2. It reveals that LA's carry more information than single-term indexes and the module can check inconsistency between a high level sentence and a low level sentence with 72.7% precision.

**Table 2: The precisions of two indexing schemes**

Rank	Single-term indexing scheme	Term-phrase indexing scheme
Top-3	68.2 %	72.7 %

The experimental result to extract ambiguous sentences is not shown in terms of precision because the result can be changed according to an analyst's preference. In this paper, we tried to design a convenient system which displays all ambiguous sentences in its window. We expect that the module can be used to improve qualities of requirements specifications.

### 5. IMPLEMENTATION

We implemented the requirements analysis supporting system in a client-server architecture. The server was implemented on Solaris 2.5.1 using C language. The client was implemented on MS-Windows 98 using Visual Basic. Figure 5 shows the initial snapshot of the client.



**Figure 5: The initial snapshot of the client**

The client consists of a preprocessing part and a function part. The system generates *profiles* of requirements specifications in the former and gives an analyst some useful tools in the latter. The system supports a function to trace dependency between documents, improve completeness in a document, reduce inconsistency between sentences, and improve quality of a document.

The system measures similarity between documents as shown in Figure 6. Using this function, an analyst can trace dependency between documents because he can easily find which specifications are associated with a new requirements specification.



**Figure 6: The snapshot to measure similarity between documents**

The system measures similarity between a sentence in a high level document and a sentence in a low level document as shown in Figure 7.



**Figure 7: The snapshot to measure between a high level sentence and a low level sentence**

Using this function, an analyst can efficiently check consistency between documents because the system automatically displays which sentence in a low level document is most similar to the given sentence in a high level document. For example, the first sentence in Figure 7 does not have any candidate sentences. It means that there are no sentences in the low level document associated with the given sentence in the high level document. Looking through those sentences, an analyzer can improve completeness of requirements specifications. Figure 8 shows the function to measure similarity between sentences in a document.



**Figure 8: The snapshot to measure similarity between sentences in a document**

Using this function, an analyst can reduce inconsistency between sentences and can improve completeness in a document. For example, he/she can find the facts that the second sentence in Figure 8, *Sentence Number:3*, is inconsistent with its candidate sentence and that the first sentence, *Sentence Number:2*, is duplicate of the first candidate sentence, *Sentence Number:12*.

The system supports the function to display ambiguous sentences as shown in Figure 9. This function helps an analyst to improve the quality of a document because non-quantifiable words produce an ambiguity problem. To support this function, the system uses MRD including ambiguous words like *많다* (many), *적다*(little), *쉽다*(easy), *크다*(big), *작다*(small), *빠르다* (fast), *느리다*(slow), *편리하다*(convenient), etc.



Figure 9: The snapshot to display ambiguous sentences

## 6. CONCLUSION

In this paper, we proposed the requirement analysis supporting system, which can be used to identify potential errors in requirements. The system supports functions to trace dependency between documents, improve completeness, reduce inconsistency between sentences that improve quality of a requirements document. To provide these functions, it uses an indexing scheme and the measures of similarity that are used in information retrieval. We combined the sliding window method with the syntactic parser method including some heuristics in order to extract indexes containing more information about a document or a sentence. The experimental results reveal that the combined method efficiently complements each method. To measure similarity between documents or sentences, the system uses *z-scores* and *Salton's cosine coefficients*. The system has some advantages such as simple framework, easy implementation because it was done only using a POS tagger, a syntactic parser and some heuristics. It also shows effectiveness in terms of performance. The developed system supports an analyst to identify potential problems of requirements documents.

## 7. ACKNOWLEDGEMENT

This work was supported in part by the ministry of information and telecommunication under grant 96055-IT2-III1.

## 8. REFERENCES

[1] Ash R., *Information Theory*, New York:Wiley-Interscience, (1965).  
 [2] Hajime M., Takeo H. and Manabu O., "Text Segmentation with Multiple Surface Linguistic Cues," *Proceedings of the COLING-ACL'98*, (1998) 881-885.  
 [3] Hearst M., "Multi-Paragraph Segmentation of Expository

Text," *Proceedings of the ACL'94*, (1994).  
 [4] Hellwig P., "Dependency Unification Grammar," *Proceedings of Colling86*, (1986) 195-198.  
 [5] Jobbins A. and Evett L., "Text Segmentation Using Reiteration and Collocation," *Proceedings of the COLING-ACL'98*, (1998) 614-618.  
 [6] Kim M., Klavans J. and McKeown K., "Linear Segmentation and Segment Significance," *Proceedings of the 6th International Workshop of Very Large Corpora(WVLC-6)*, (1998) 197-205.  
 [7] Kozima H., "Text Segmentation Based on Similarity between Words," *Proceedings of ACL'93*, (1993) 286-288.  
 [8] Litman D. and Passonneau R., "Combining Multiple Knowledge Sources for Discourse Segmentation," *Proceedings of the 33rd ACL*, (1995).  
 [9] Maarek Y., Berry D. and Kaiser G, *An Information Retrieval Approach For Automatically Construction Software Libraries*, IEEE Transaction On Software Engineering, Vol. 17, No. 8, (1991) 800-813.  
 [10] Martin W.J.R., Al B. P. F., and van Sterkenburg P. J. G., "On the processing of a text corpus: From textual data to lexicographic information," *Lexicography: Principles and Practice* (Applied Language Studies Series), Hartmann R. R. K., Ed. London: Academic, (1983).  
 [11] Mel'cuk I. A., *Dependency Syntax: Theory and Practice*, State Univ. of New York Press, (1988).  
 [12] Palmer J. D., Liang Y. and Wang W., "Classification as an approach to requirements analysis," *Proceedings of ASIS '90 Workshop on Classification Research*, Toronto, Canada 4, (1990).  
 [13] Palmer J. and Liang Y., *Indexing and clustering of software requirements specifications*, Information and decision Technologies, Vol 18, (1992) 283-299.  
 [14] Park S. and Palmer J. D., "Automated Support to System Modeling from Informal Software Requirements," *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering*, (1994).  
 [15] Rijsbergen C. J., *Information Retrieval*, 2nd ed. Stoneham, MA: Butterworths, (1979).  
 [16] Samson D., "A knowledge-based requirements system," *Productivity: Progress, Prospects, and Payoff*, 27th Annual Technical Symposium, D. C. Chapter of the ACM and NBS, Washington, (1991).  
 [17] Salton G. and McGill M.J., *Introduction to Modern Information Retrieval* (Computer Series), New York:McGraw-Hill, (1983).  
 [18] Salton G., *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*, Reading, MA:Addison-Wesley, (1989).  
 [19] Saussure F. *Cours de Linguistique Generale*, Quatrieme Edition, Paris: Librairie Payot, (1949).  
 [20] Shumskas A. S., Askman V. N. and Cunningham J. F., "Conceptual information retrieval using RUBIC," *Proceedings of the 10th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (1987).  
 [21] Smadja F. A., *Lexical Co-occurrence: The missing link*, Literary and Linguistic Computing, vol. 4, no. 3, (1989).  
 [22] Yaari Y., "Segmentation of Expository Texts by Hierarchical Agglomerative Clustering," *Proceedings of RANLP'97*, (1997) 135-142.