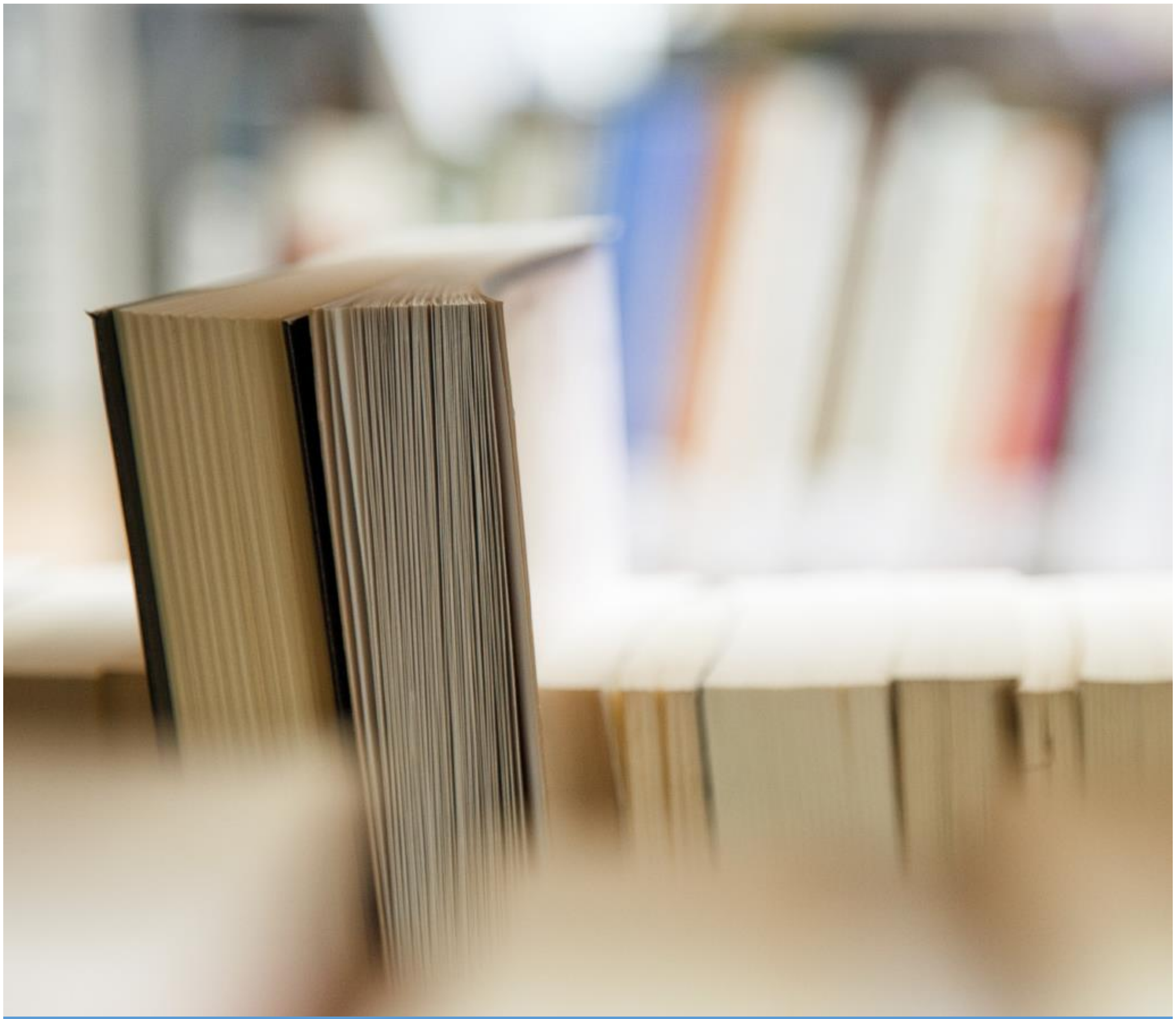

SPL-3 Report

A Research and Development Project…

M. A. Nur Quraishi 10/27/17 SE-801: SPL-3

Project Name:

“Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing”

Written by

M. A. Nur Quraishi
Roll: BSSE-0615
Institute of Information Technology,
University of Dhaka

Supervised by

Rayhanur Rahman
Lecturer,
Institute of Information Technology,
University of Dhaka

Table of Contents

Abstract:

[An information retrieval method named latent semantic indexing is applied to diagnose and analyze the traceability links from system documentation to program source code automatically. It is a very cost effective and highly flexible method to apply with regards to preprocessing and/or parsing of the source code and documentation.]

1 Introduction

Most of the documentations including software requirement specification (SRS), design documents, user manuals and also annotations of individual programmers and teams associated with an application or large software system are free text documents in form of either PDF or DOC interpreted in a natural language. Moreover, these documents often cover the available knowledge of the application domain. Even when (semi-)formal models are applied, free text is largely adopted either to add semantics and context information in the form of comments or ease the understanding of the formal models to nontechnical readers. (Antoniol, Canfora, Casazza, & De Luc, Recovering traceability links between code and documentation, 2002)

The whole software engineering community (both research and industrial) in this era stride to improve the explicit connection of documentation and source code. A number of integrated development environments and computer aided software engineering (CASE) tools are particularly focused on this issue. These tools and techniques have played an important role in documentation to source code traceability for the development of new software systems. Miserably, several of these methods intrinsically incompatible for existing and/or legacy systems.

The need for tools and applications to recover documentation to source code traceability links in legacy systems is particularly important for various software engineering tasks. Such as- general maintenance tasks, impact analysis, program comprehension, and more encompassing tasks such as reverse engineering for redevelopment and systematic reuse. (Marcus & Maletic, 2003)

The preeminent obstacle in link recovery process is that the links are rarely explicit and based on the semantic meaning of the prose in the documentation. Contrasting some sort of natural language analysis of the documentation with that of the source code is an obviously difficult problem. Here, we are offering a cost effective and more pragmatic solution to this problem by utilizing an advanced information retrieval technique (latent semantic analysis) to extract the meaning (semantics) of the documentation and source code. Then use this information to identify traceability links based on similarity measures.

The mechanism uses all the comments (internal documents) and identifier names within the source code to produce semantic meaning with respect to the entire input document space. This experimental study is well-supported by the work of Anquetil (Anquetil & Lethbridge, 1998) and others in determining the importance of this information in existing software. This entails the postulates that the comments and identifiers are reasonably named however, the alternative bares little hope of deriving a meaning automatically (or even manually).

The major advantage of using this method is that it does not depend on predefined vocabulary or grammar (including sentence structure) for the documentation and source code. As a result, this procedure does not require large amounts of preprocessing or manipulation of the input, which drastically reduces the costs of link recovery.

2 Background Study

There is a wide range of information retrieval (IR) mechanisms. Conventional approaches (Faloutsos & Oard, 1995) such as signature files, inversion, classifiers, and clustering. Other methods applying parsing, syntactic information, natural language processing techniques, neural networks, and advanced statistical methods that attempt to capture more information about the documents in order to achieve better performance. Much of this work deals with natural language text and a large number of techniques exist for indexing, classifying, summarizing, and retrieving text documents. These methods produce a profile for each document where the profile is an abbreviated description of the original document that is easier to manipulate. This profile is typically represented as vector, often real valued. The method, we are going to implement also has an underlying vector space model. We are now going to present a discussion of these type of representations. (Marcus & Maletic, 2003)

2.1 Vector Space Model

The vector space model (VSM) (Salton & McGill, 1983) is a widely used classic method for constructing vector representations for documents. It encodes a document collection by a term-by-document matrix whose [i, j]th element indicates the association between the ith term and jth document. In typical applications of VSM, a term is a word, and a document is an article. However, it is possible to use different types of text units. For instance, phrases or word/character n-grams can be used as terms, and documents can be paragraphs, sequences of n consecutive characters, or sentences.

The importance of VSM is that it represents one type of text unit (documents) by its association with the other type of text unit (terms) where the association is calibrated by explicit evidence based on term occurrences in the documents. A geometric view of a term-by-document matrix is as a set of document vectors occupying a vector space spanned by terms. We call this vector space VSM space.

The similarity between documents is typically measured by the cosine or inner product between the corresponding vectors, which increases as more terms are shared. In general, two documents are considered to be similar if their corresponding vectors in the VSM space point in the same (general) direction. (Marcus & Maletic, 2003)

2.2 Latent Semantic Indexing

Latent Semantic Indexing (LSI) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990) is based on VSM for inducing and representing aspects of the meanings of words and passages reflective in their usage.

Experiment applying LSI to natural language text by (Landauer & Dumais, 1997) has shown that LSI not only covers significant portions of the meaning of individual words but also of whole passages such as sentences, paragraphs, and short essays. The basic idea of LSI is that the information about word contexts in which a particular word appears, or does not appear, provides a set of mutual constraints that determines the similarity of meaning of sets of words to each other. (Marcus & Maletic, 2003)

2.3 Singular Value Decomposition and LSI

For text analysis, LSI uses a user-constructed corpus to create a term-by-document matrix. Then it applies Singular Value Decomposition (SVD) (Salton & McGill, 1983) to the term-by-document matrix to construct a subspace, called an LSI subspace. New document vectors (and query vectors) are obtained by orthogonally projecting the corresponding vectors in a VSM space (spanned by terms) onto the LSI subspace. Following the mathematical formulation of LSI, the term combinations which are less frequently occurring in the given document collection like to be excluded from the LSI subspace. According to our examples above, one could argue that LSI performs “noise reduction” if it was true that less frequently co-occurring terms are less mutually-related, and therefore less sensible.

The factor behind using SVD is rather complex and lengthy to elaborate here. The interested reciter is referred to (Salton & McGill, 1983) for details. Apparently, in SVD (Marcus & Maletic, 2003) a rectangular matrix X is decomposed into the product of three other matrices. One component matrix (U) describes the original row entities as vectors of derived orthogonal factor values. Another component (V) describes the original column entities in the same way, and the third is a diagonal matrix (Σ) containing scaling values such that when the three components are matrix-multiplied, the original matrix, X is reconstructed (i.e., $X = U\Sigma V^T$). The columns of U and V are the left and right singular vectors, respectively, corresponding to the monotonously decreasing (in value) diagonal elements of Σ which are called the singular values of the matrix X. When fewer than the necessary number of factors are used, the reconstructed matrix is a least-squares best fit. One can reduce the dimensionality of the solution simply by deleting coefficients in the diagonal matrix, ordinarily starting with the smallest. The first k columns of the U and V matrices and the first (largest) k singular values of X are used to construct a rank-k approximation to X through $X_k = U_k\Sigma_k V_k^T$. The columns of U and V are orthogonal, such that $U^TU = V^TV = I_r$, where r is the rank of the matrix X. X_k constructed from the k-largest singular triplets of X (a singular value and its corresponding left and right singular vectors are referred to as a singular triplet), is the closest rank-k approximation (in the least squares sense) to X.

With regard to LSI, X_k is the closest k-dimensional approximation to the original term-document space represented by the incidence matrix X. As stated previously, by reducing the dimensionality of X, much of the “noise” that causes poor retrieval performance is thought to be eliminated. Thus, although a high-dimensional representation appears to be required for good retrieval performance, care must be taken to not reconstruct X. If X is nearly reconstructed, the noise caused by variability of word choice and terms that span or nearly span the document collection won't be eliminated, resulting in poor retrieval performance. (Marcus & Maletic, 2003)

The implementation of LSI has been empirically studied. The experiment in (Dumais, 1991) explores the effects of several term weighting schemes to instantiate the input term-by-document matrix. Evaluation was based on the precision/recall curves on the retrieval tasks with the dimensionality of the LSI subspace being fixed. Several term-weighting schemes, which combine global weights (i.e., statistics in the collection) and local weights (i.e., statistics within each document), were investigated. Log Entropy, which is a combination of a local log weight and a global entropy weight, showed superiority over the combinations of the local term frequency and global weighting schemes or no global weighting. Two well-known global weightings (i.e., Gfldf and Normal) produced performance worse than no global weighting.

Particularly, various experimental studies have shown that the performance of LSI significantly varies over the dimensionalities of the LSI subspace (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990). In practice, the dimensionality is chosen experimentally, or blindly picked by following the existing work.

After the documents are imitated in the LSI subspace (Marcus & Maletic, 2003), the user can compute similarities measures between documents by the cosine between their corresponding vectors or by their length. These measures can be used for clustering similar documents together, to identify concepts and topics in the corpus. This method is typically used for text analysis tasks. The LSI representation can also be applied to map new documents (or queries) into the LSI subspace and find which of the existing documents are similar (relevant) to the query. This usage is typical for information retrieval tasks.

3 Literature Review

A major drawback of VSM is that it does not consider relations between terms. For instance, having "automobile" in one document and "car" in another document does not contribute to the similarity measure between these two documents.

The fact that VSM produces zero similarity between text units that share no terms is an issue, especially in the information retrieval task of measuring the relevance between documents and a query submitted by a user. Usually, a user query is tiny and does not consider all the vocabulary for the target concept. In VSM, “car” in a query and “automobile” in a document do not contribute to retrieving this document (synonym problem). LSI attempts to overcome this problem by selecting linear combinations of terms as dimensions of the representation space. The examples in (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990 & Landauer, Foltz, & Laham, Discourse Processes, 1998) show that LSI may solve this synonym problem by generating positive similarity between related documents sharing no terms.

As the LSI subspace grabs the most important factors (i.e., those contains the largest singular values) of a term-by-document matrix, it is anticipated to capture the relations of the most repeatedly simultaneously occurring terms. This fact is understood when we realize that the SVD factors a term-by-document matrix into the largest one-dimensional projections of the document vectors, and that each of the document vectors can be regarded as a linear combination of terms. From this perspective, LSI can be viewed as a corpus-based statistical method. However, the relations among terms are not modeled explicitly in the computation of LSI subspace, which makes it hard to understand LSI in general. Despite an LSI subspace provides the best low rank and cost effective approximation of the term-by-document matrix, it does not denote that the LSI subspace approximates the “true” semantics of documents.

Another criticism of this method is that when it applied to natural language texts, it does not capitalize on word order, syntactic relations, or morphology. Nevertheless, very good representations and results are acquired without this information. This feature is very much suitable to the domain of source code and internal documentation (Programmer annotations or comments). Because much of the informal abstraction of the problem concept may be embodied in names of key operators and operands of the implementation. Here, the word ordering is meaningless. Source code is hardly English prose but with the convention of selective naming, much of the high level meaning of the problem-at-hand is transmit to the reader (the programmer). Internal source code documentation (comment) is also commonly written in a subset of English that also utilized in IR methods. This makes automation exceptionally easier and directly supports programmer defined variable names that have implied meanings (e.g., avg) which are not in the

English language vocabulary. The meanings are determined from their usage rather than a predefined dictionary. This is a certain advantage over using a traditional natural language processing approach.

Like other IR methods, LSI does not use a grammar or a predefined vocabulary. However, it uses a list of "stop words" that can be extended by the user. These words are excluded from the analysis. Regardless of the IR method used in text analysis, in order to identify two documents as similar they must have in common concepts represented by the association of terms and their context of usage. In other words, two documents written in different languages will not appear similar. In the case of source code, our main assumption is that developers use the same natural language (e.g., English, French, German etc.) in writing internal documentation and external documentation. In addition, the developer should have some sense and consistency in defining and using identifiers. (Marcus & Maletic, 2003)

4 Related Work

The research topic on which we are working in this paper specifies two distinct problems: applying IR methods to support software engineering tasks and recovering source code to documentation links. The previous works (Fischer, 1998 & Maarek, Berry, & Kaiser, 1991) related to this research uses a technique named indexing reusable components as information retrieval method in order to identify the traceability link from documentation to source code. Remarkably, the work of Maarek (Maarek, Berry, & Kaiser, 1991) on the use of information retrieval method for automatically constructing software libraries contributes most among them. The success of this work as well as the incompetence and less cost effectiveness of constructing the knowledge base associated with natural language parsing methods to this problem (Etzkorn & Davies, 1997) are the main motivations behind our work. In brief, it is very costly (and often unrealistic) to build the knowledge base(s) necessary for parsing methods to derive more significant semantic information from source code and corresponding documentation. Employing IR methods (based on statistical and heuristic methods) may not generate a result close to the actual one, but they are pretty cost effective to implement. From these stuidies, we can conclude that it may generate close and low cost results.

More recently, (Marcus & Maletic, 2001) used LSI to extract similarity measures among source code elements. Later, these measures were used to cluster the source code to support the identification of abstract data types in procedural code. Moreover, these measures were used to define a cohesion metric for components. The work on which we are working extends these results in a new direction.

Concurrently, Antoniol and others inspected the use of IR methods to support the traceability link recovery process. In particular, they used both a probabilistic method (Antoniol, Canfora, Casazza, De Lucia, & Merlo, Tracing Object-Oriented Code into Functional Requirements, 2000) and a vector space model (Antoniol, Canfora, Casazza, & De Luc, Recovering traceability links between code and documentation, 2002) to identify links between source code and documentation and between source code and requirements. Their results were decent in each case. Moreover, this work also supports the selection of vector space models over probabilistic IR. Applying LSI makes the work presented here quite different in many aspects and yet provides complementary results.

5 References

❖ Anquetil, N., & Lethbridge, T. (1998). Assessing the Relevance of Identifier Names in a Legacy Software System. (pp. 213-222). CASCON. Retrieved October 6, 2017

❖ Antonio, G., Canfora, G., Casazza, G., & De Lucia, A. (2000). Information Retrieval Models for Recovering Traceability Links between Code and Documentation. *IEEE International Conference on Software Maintenance (ICSM'00)* (pp. 40-51). ICSM. Retrieved October 2, 2017

❖ Antoniol, G., Canfora, G., Casazza, G., & De Luc, A. (2002, October). Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering, 28*(10), 970 - 983. Retrieved September 12, 2017

❖ Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., & Merlo, E. (2000). Tracing Object-Oriented Code into Functional Requirements. *8th International Workshop on Program Comprehension (IWPC'00)* (pp. 79-87). Limerick, Ireland: IWPC. Retrieved October 1, 2017

❖ Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). *Indexing by Latent Semantic Analysis".* American Society for Information Science. Retrieved October 12, 2017

❖ Dumais, S. (1991). Improving the retrieval of information from external sources. *23*(2), 229-236. Retrieved October 3, 2017

❖ Etzkorn, L., & Davies, C. (1997, October). Automatically Identifying Reusable OO Legacy Code. *IEEE Computer, 30*(10), 66-72. Retrieved September 30, 2017

❖ Faloutsos, C., & Oard, D. (1995). *A Survey of Information Retrieval and Filtering Methods.* Technical Report CS-TR-3514, University of Maryland. Retrieved September 14, 2017

❖ Fischer, B. (1998). Specification-Based Browsing of Software Component Libraries. *ASE*, (pp. 74-83). Retrieved October 2, 2017

❖ Landauer, T., & Dumais, S. (1997). *A Solution to Plato's Problem: The Latent Semantic Analysis Theory of the Acquisition, Induction, and Representation of Knowledge.* Psychological Review. Retrieved October 15, 2017

❖ Landauer, T., Foltz, P., & Laham, D. (1998). An Introduction to Latent Semantic Analysis. *25*(2&3), 259-284. Retrieved October 13, 2017

❖ Maarek, Y., Berry, D., & Kaiser, G. (1991). An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions on Software Engineering, 17*(8), 800-813. Retrieved October 3, 2017

❖ Marcus, A., & Maletic, J. (2001). Supporting Program Comprehension Using Semantic and Structural Information. *23rd International Conference on Software Engineering (ICSE 2001)* (pp. 103-112). Toronto, Ontario, Canada: ICSE. Retrieved September 27, 2017

❖ Marcus, A., & Maletic, J. (2003). Recovering Documentation-to-Source-Code Traceability Links using. *ICSE '03 Proceedings of the 25th International Conference on Software Engineering* (pp. 125-135). Portland, Oregon: IEEE Computer Society Washington, DC, USA. Retrieved August 4, 2017

❖ Salton, G., & McGill, M. (1983). *Introduction to Modern Information Retrival.* McGraw-Hill. Retrieved October 16, 2017