Discrete Characterization of Domain Using Semantic Clustering

Sanjay Madan
Comviva Technologies Ltd.,
MBS-PACS, Gurgaon, India.
Email: san.madan@gmail.com

Shalini Batra
Computer Science and Engineering Department,
Thapar University, Patiala, Punjab, India
Email: sbatra@thapar.edu.

***Abstract*—Lots of approaches have been developed to understand the software source code and majority of them are focused on program structural information which results in the loss of domain semantic crucial information contained in the text or symbols of source code. To understand software as a whole, we need to enrich these approaches with conceptual insights gained from the domain semantics. This paper proposes the mapping of domain to the code using the information retrieval techniques to use linguistic information, such as identifier names and comments in source code. Concept of Semantic Clustering has been introduced in this paper and an algorithm has been provided to group source artifacts based on how the synonymy and polysemy is related. Based on semantic similarity automatic labeling of the program code is done after detecting the clusters, and is visually explore in 3-Dimension for discrete characterization. This approach works at the source code textual level which makes it language independent. The approach correlates the semantics with structural information applies at different levels of abstraction (e.g. packages, classes, methods).**

***Index Terms*— Information retrieval, Semantic clustering, Software reverse engineering.**

I. Introduction

To get knowledge about a software system is one of the main activities in software reengineering. It has been estimated that up to 60 percent of software maintenance is spent on comprehension [1]. This is because a lot of knowledge about the software system and its associated business domain is not captured in an explicit form. Most approaches that have been developed focus on program structure [2] or on external documentation [3, 4]. However, the identifier names and the source code comments are the main fundamental source of information.

The source code comprises of two types of communication: human-machine communication through program instructions and human to human communications through names of identifiers and comments [5]. The executables are for machine where as code is written for humans not for machines. Let us consider a small code example, which tell whether a time value is in the morning:

```
/** Return true if the given 24-hour time is in the morning and false otherwise. */
public boolean isMorning(int hours,int minutes,int seconds) {
  if (!isDate(hours, minutes, seconds)) throw
        Exception("Invalid input: not a time value.")
  return hours < 10 && minutes < 60 && seconds < 60; }
```

When we strip away all identifiers and comments, from the machine point of view the functionality remains the same, but for a human reader the meaning is obfuscated and almost impossible to figure out. In our example, retaining formal information yields:

```
public type1 method1(type2 a, type2 b, type2 c) {
    if (!method 2(a, b ,c)) throw Exception(literal 1).
            return (a < A) && (b < B) && (c < C);
}
```

In this informal information, the vocabulary is presented in random order and the domain of the code is still recognizable. In this example, retaining only the naming yields:

```
is int hours minutes int < minutes input hours is
seconds && boolean morning false 24 time minutes not
60 invalid && value seconds time < seconds hour
given hours 60 12 < morning date int is otherwise [5].
```

It is a well known fact that information retrieval provides means to analyze, classify and characterize text documents based on their content and the given representation of documents as bag-of-terms is a well-established technique in information retrieval (IR) used to model documents in a text corpus. Apart from external documentation, the location and use of source-code identifiers is the most frequently consulted source of information in software maintenance. In the software analysis different approaches that apply IR on external documentation [6, 7], but only few work has been focused on treating the source code itself as data source. In this case we are using information retrieval to derive topics from the vocabulary usage at the source code level. First three steps of the domain extraction from source code include: pre-processing, applying LSI, and clustering. Furthermore we retrieve the most relevant terms for each cluster, thus in short the approach is:

(1) **Pre-processing the software system.** Break the system into documents and build a term-document-matrix that contains the vocabulary usage of the system.
(2) **Applying Latent Semantic Indexing.** Use LSI to compute the similarities between source code documents and illustrate the result in a correlation matrix [10].
(3) **Identifying topics.** Then cluster the documents based on their similarity, we rearrange the correlation matrix and each cluster is a linguistic topic.
(4) **Describing the topics with labels**. Use LSI again to retrieve for each cluster the top-n most relevant terms.

II. Latent Semantic Indexing

Latent Semantic Indexing (LSI) is a technique common in information retrieval to index, analyzes and classifies text documents. It analyzes how terms are spread over the documents of a text corpus and creates a search space with document vectors: similar documents are located near each other in this space and unrelated documents far apart of each other. Since LSI can be used to locate linguistic topics in a set of documents [8, 9], it is applied to compute the linguistic similarity between source artifacts (e.g. packages, classes or methods) and cluster them according to their similarity. This clustering partitions the system into linguistic topics that represent groups of documents using similar vocabulary. It is used to analyze the linguistic information of a software system as the source code is basically composed of text documents.

To illustrate it further, like other IR techniques, Latent Semantic Indexing is based on the vector space model (VSM) approach. This approach models documents as bag-of-words and arranges them in a Term-Document Matrix A, such that $a_{i,j}$ equals the number of times term t_i occurs in document d_j.

LSI has been developed to overcome problems with synonymy and polysemy that occurred in prior vectorial approaches, and thus improves the basic vector space model by replacing the original term-document matrix with an approximation. This is done using singular value decomposition (SVD), a principal components analysis (PCA) technique originally used in signal processing to reduce noise while preserving the original signal. Assuming that the original term-document matrix is noisy (the synonymy and polysemy), the approximation is interpreted as a noise reduced – and thus better – model of the text corpus.

For example, a typical search engine covers a text corpus with millions of web pages, containing some ten thousands of terms, which is reduced to a vector space with 200-500 dimensions only. In Software Analysis, the number of documents is much smaller and we reduce the text corpus to 20-50 dimensions.

There is a wide range of applications of LSI, such as automatic assignment of reviewers to submitted conference papers [10], cross-language search engines, spell checkers and many more. In the field of software engineering LSI has been successfully applied to categorized source files [11] and open-source projects [12], detect high-level conceptual clones [13], recover links between external documentation and source code [14,15]. Furthermore LSI has proved useful in psychology to simulate language understanding of the human brain, including processes such as the language acquisition of children.

Figure 1 schematically represents the LSI process. The document collection is modeled as a vector space. Each document is represented by the vector of its term occurrences, where terms are words appearing in the document. The term-document-matrix A is a sparse matrix and represents the document vectors on the rows. This matrix is of size n × m, where m is the number of documents and n the total number of terms over all documents. Each entry ai,j is the frequency of term ti in document dj . A geometric interpretation of the term-document-matrix is a set of document vectors occupying a vector space spanned by the terms. The similarity between documents is typically defined as the cosine or inner product between the corresponding vectors. Two documents are considered similar if their corresponding vectors point in the same direction.

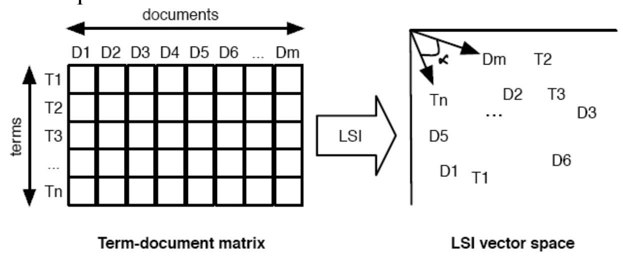

Figure 1, LSI takes as input a set of documents and the terms occurrences, and returns as output a vector space containing all the terms and all the documents. The similarity between two items (terms or documents) is given by the angle between their corresponding vectors [5].

LSI starts with an input as term-document-matrix, weighted by a weighting function to balance out very rare and very common terms. SVD is used to break down the vector space model into less dimensions. This algorithm preserves as much information as possible about the relative distances between the document vectors, while collapsing them into a much smaller set of dimensions.

SVD decomposes matrix A into its singular values and its singular vectors, and yields – when truncated at the k largest singular values – an approximation A` of A with rank k. Furthermore, not only the low-rank term-document matrix A` can be computed but also a term-term matrix and a document-document matrix. Thus, LSI allows us to compute term-document, term-term and document-document similarities.

As the rank is the number of linear-independent rows and columns of a matrix, the vector space spanned by A\` is of dimension k only and much less complex than the initial space. When used for information retrieval, k is typically about 200-500, while n and m may go into millions. When used to analyze software on the other hand, k is typically about 20−50 with vocabulary and documents in the range of thousands only, and since A\` is the best approximation of A under the least-square-error criterion, the similarity between documents is preserved, while in the same time mapping semantically related terms on one axis of the reduced vector space and thus taking into account synonymy and polysemy. In other words, the initial term-document-matrix A is a table with term occurrences and by breaking it down to much less dimension the latent meaning must appear in A\` since there is now much less space to encode the same information. Meaningless occurrence data is transformed into meaningful concept information.

III. Term And Document Similarity

To show the SVD factors geometrically, the rows of the matrices are taken as coordinates of points representing the documents and terms vector dimensional space. The nearer one points to the other, if they are more similar documents or terms (see Figure 2). Similarity is typically defined as the cosine between the corresponding vectors:

$\mathrm{sim}(d_i, d_j) = \cos(v_i, v_j)$

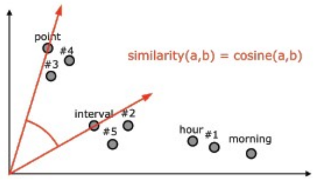

Figure 2: On the left: An LSI-Space with terms and documents, similar elements are placed near each other [5].

Computing the similarity between document d_i and d_j is done taking the cosine between the i-th and j-th row of the matrix. The resulting cosine value, similarity values range from 1 to 0: 1 for similar vectors with the same direction and to 0 for dissimilar, orthogonal vectors. Theoretically cosine values can go all the way to −1, but because there are no negative term occurrences, similarity values never goes below to zero.

IV. Semantic Clustering

Semantic clustering is a non-interactive and unsupervised technique to analyze the semantics of a software system. Semantic clustering offers a high level view on the domain concepts of a system, abstracting concepts from software artifacts. Firstly, Latent Semantic Indexing (LSI) is used to extract linguistic information from the source code and then clustering is applied to group the related software artifacts into clusters and groups of artifacts having the same vocabulary are identified and these are called clusters. Thus each cluster reveals a different concept of the system. Most of these are domain concepts, some are implementation concepts. The actual ratio depends on the naming convention of the system.

At the end, the inherently unnamed concepts are labelled with terms taken from the vocabulary of the source code. An automatic algorithm labels each cluster with most similar terms, and in this way provide a human readable description of the main concepts in a software system. Additionally, the clustering is visualized as a shaded Correlation Matrix that illustrates:

- the semantic similarity between elements of the system, the darker a dot the more similar its artifacts,
- a partition of the system into clusters with high semantic cohesion, which reveals groups of software artifacts that implement the same domain concept,
- semantic links between these clusters, which emphasize single software artifacts that interconnect the above domain concepts.

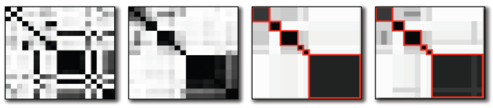

Figure 3: From left to right: unordered correlation matrix, then sorted by similarity, then grouped by clusters, and finally including semantic links [5].

V. Building The Text Corpus

Text corpus is a large and structured set of texts. To build a semantic model, Latent Semantic Indexing (LSI) is used to analyze the distribution of terms over a text corpus. When applying LSI on a software system we break its source code into documents and use the vocabulary found therein as terms. The system can be split into documents at any level of granularity, such as modules, classes or methods, it is even possible to use entire projects as documents [16].

The vocabulary of source can be extracted both from the content of comments and from the identifier names. Comments are parsed as natural language text and compound identifier names split into their parts. As most modern naming conventions are used camel case, it is straight forward to split identifiers: for example, FooBar becomes foo and bar. In case of legacy code that uses other naming conventions, more advanced algorithms and heuristics are required [17]-[18].

Common stop words are excluded from the vocabulary, as they do not help to discriminate documents, and stemmer algorithm is used to reduce all words to their morphological root. Finally the term-document matrix is weighted with tf-idf (Term frequency, inverted document frequency), to balance out the influence of very rare and very common terms.

VI. Semantic Similarity And Correlation Matrix

Semantic similarity is the likeness of meaning/semantic content within a set of documents or terms. Latent Semantic Indexing (LSI) can be used to extract linguistic information from the source code. The result of this process will be an LSI index L with similarities between software artifacts as well as terms. Based on the index we can determine the similarity between these elements. Software artifacts are more similar if they cover the same concept, terms are more similar if they denote related concepts. Since similarity is defined as cosine between element vectors, its values range between 0 and 1. The similarities between elements are arranged in a square matrix A called the Correlation Matrix.

To visualize the similarity values we map them to gray values: the darker, the more similar. In that way the matrix becomes a raster-graphic with gray dots: each dot $a_{i,j}$ shows the similarity between element d_i and element d_j. The elements are arranged on the diagonal and the dots in the off-diagonal show the relationship between them.

Without proper ordering, the correlation matrix looks like a television tuned to a dead channel. An unordered matrix does not reveal any patterns: arbitrary ordering, such as the names of the elements, is generally as useful as random ordering [19]—therefore, matrix will be clustered such that similar elements are put near each other and dissimilar elements far apart of each other. After applying the clustering algorithm, the similar elements are grouped together and aggregated into concepts. Hence, a concept is characterized as a set of elements that uses the same vocabulary. Documents that are not related to any concept usually end up in singleton clusters in the middle or in the bottom right of the correlation matrix. The correlation matrices are ordered using average linkage clustering algorithm.

The matrix will be reordered first, and then dots will be grouped by clusters and colour them with their average cluster similarity. As with the element similarities in the previous section, the similarities between clusters are arranged in a square matrix A. When visualized, this matrix becomes a raster-graphic with gray rectangles: each rectangle $r_{i,j}$ shows the similarity between cluster R_i and cluster R_j , and has the size ($|R_i|$, $|R_j|$). The clusters are arranged on the diagonal and the rectangles in the off-diagonal show the relationship between them—see the third matrix on Figure 3.

A correlation matrix is gray-scale raster-graphic: each dot $a_{i,j}$ shows the similarity between element d_i and element d_j—the darker, the more similar. The elements are arranged on the diagonal while the dots in the off-diagonal show the relationship between them. An unordered matrix does not reveal any patterns; therefore we cluster the elements and sort the matrix: all dots in a cluster are grouped together and are colour with their average similarity; this is semantic cohesion [20]. This offers a high-level view on that system, abstracting from elements to concepts.

VII. Discrete Characterization of Clusters

Visualization of the cluster in 3-Dimesion extended the domain detection concept much simpler in terms of distributed application. Just visualizing clusters is not enough; labelling is required to describe the cluster. Often just enumerating the names of the software artifacts in a cluster gives a sufficient interpretation. If the names are badly chosen or unnamed software artifacts are analyzed, we need an automatic way to identify labels. Figure 4 shows the labels in the concept of LAN example.

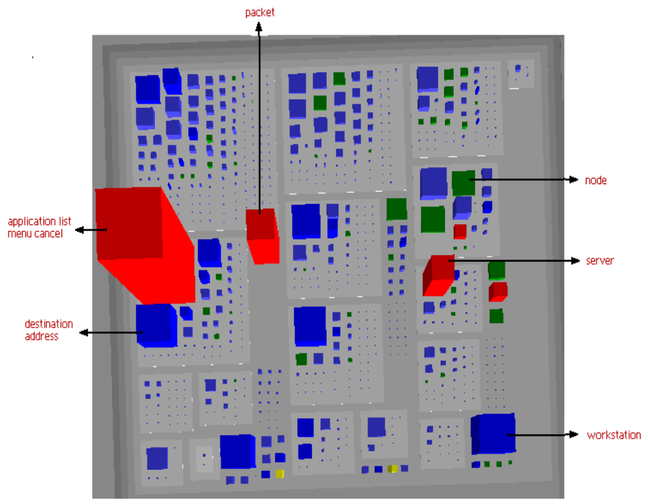

Figure 4: Automatically retrieved labels describe the concepts. The labels were retrieved using the documents in a concept cluster as query to search the LSI space for related terms.

To obtain the most relevant labels comparison will be performed between the similar terms of the current cluster and similar terms of all other clusters.

All the steps of the domain extraction from source code include: pre-processing, applying LSI, clustering and retrieve the most relevant terms for each cluster and the similarity measurement to identify topics in the source code will follow the flow as depicted in the figure:

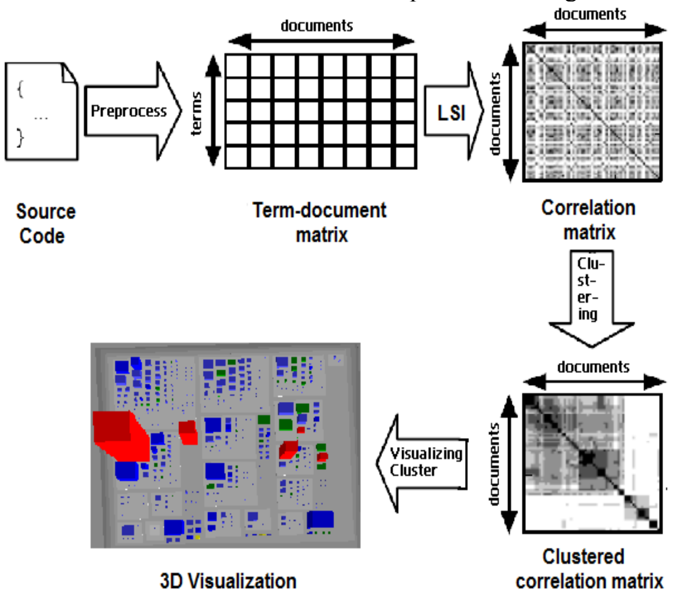

Figure 5: Modified Semantic clustering of software source code [5].

VIII. Conclusion

When understanding a software system, analyzing its structure reveals only half of the story. The other half resides in the domain semantics of the implementation. Developers put their domain knowledge into identifiers name or comments. This work presented the use of Semantic Clustering to analyze the textual content of source code to recover domain concepts from the code itself [22]. To identify the different concepts in the code, we applied Latent Semantic Indexing (LSI) and cluster the source artifacts according to the vocabulary of identifiers and comments. Each cluster represents a distinct domain concept. To define the concept and to retrieve the most relevant labels for clusters, LSI technique has been used. For each cluster, the labels are obtained by ranking and filtering the most similar terms [16]. The result of applying LSI is a vector space, based on which we can compute the similarity between either documents or terms.

References

[1] A. Abran, P. Bourque, R. Dupuis, L. Tripp, "Guide to the software engineering body of knowledge (*ironman version*)," Tech. rep., IEEE Computer Society (2004).

[2] S. Ducasse, M. Lanza, "The class blueprint: Visually supporting the understanding of classes," *IEEE Transactions on Software Engineering* 31 (1) (2005) 75–90.

[3] Y. S. Maarek, D. M. Berry, G. E. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Transactions on Software Engineering* 17 (8) (1991) 800–813.

[4] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering* 28 (10) (2002) 970–983.

[5] Adrian Kuhn, Stephane Ducasse, Tudor Girba, "Semantic Clustering: Identifying Topics in Source Code," *Language and Software Evolution Group*, LISTIC, Universite de Savoie, France, 2006

[6] Yo¨elle S. Maarek, Daniel M. Berry, and Gail E. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Transactions on Software Engineering*, 17(8):800–813, August 1991.

[7] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[8] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society of Information Science* 41 (6) (1990) 391–407.

[9] A. Marcus, A. Sergeyev, V. Rajlich, J. Maletic, "An information retrieval approach to concept location in source code", in: *Proceedings of the 11thWorking Conference on Reverse Engineering* (WCRE 2004), 2004, pp. 214–223.

[10] S. T. Dumais, J. Nielsen, "Automating the assignment of submitted manuscripts to reviewers," In *Research and Development in Information Retrieval*, 1992, pp. 233–244.

[11] J. I. Maletic, A. Marcus, "Using latent semantic analysis to identify similarities in source code to support program understanding," In: *Proceedings of the 12th International Conference on Tools with Artificial Intelligences* (ICTAI 2000), 2000, pp. 46–53.

[12] S. Kawaguchi, P. K. Garg, M. Matsushita, K. Inoue, "Mudablue: An automatic categorization system for open source repositories," in: *Proceedings of the 11th Asia-Pacific Software Engineering Conference* (APSEC 2004), 2004, pp. 184–193.

[13] A. Marcus, J. I. Maletic, "Identification of high-level concept clones in source code," in: *Proceedings of the 16th International Conference on Automated Software Engineering* (ASE 2001), 2001, pp. 107–114.

[14] A. De Lucia, F. Fasano, R. Oliveto, G. Tortora, "Enhancing an artefact management system with traceability recovery features," in: *Proceedings of 20th IEEE International Conference on Software Maintainance* (ICSM 2004), 2004, pp. 306–315.

[15] A. Marcus, D. Poshyvanyk, "The conceptual cohesion of classes," in: *Proceedings Internationl Conference on Software Maintenance (ICSM 2005), IEEE Computer Society Press, Los Alamitos CA*, 2005, pp. 133–142.

[16] Adrian Kuhn, Stephane Ducasse, and Tudor Girba, "Semantic clustering: Exploiting source code linguistic information," *Information and Software Technology*, submitted, 2006.

[17] Bruno Caprile and Paolo Tonella. Nomen est omen, "Analyzing the language of function identifiers," In *Proceedings of 6th Working Conference on Reverse Engineering* (WCRE 1999), pages 112–122. IEEE Computer Society Press, 1999.

[18] Nicolas Anquetil and Timothy Lethbridg, "Extracting concepts from file names; a new file clustering criterion," *In International Conference on Software Engineering* (ICSE'98), pages 84–93, 1998.

[19] Jaques Bertin, "Graphics and Graphic Information Processing," Walter de Gruyter, 1981.

[20] Andrian Marcus and Denys Poshyvanyk, "The conceptual cohesion of classes," In *Proceedings Internationl Conference on Software Maintenance* (ICSM 2005), pages 133–142, Los Alamitos CA, 2005. IEEE Computer Society Press.

[21] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Review*, 37(4):573–597, 1995

[22] Adrian Kuhn, St´ephane Ducasse, and Tudor Gˆırba, "Enriching reverse engineering with semantic clustering," In *Proceedings of Working Conference on Reverse Engineering (WCRE 2005), pages 113–122, Los Alamitos CA*, November 2005. IEEE Computer Society Press.

Sanjay Madan, Author

Sanjay Madan is working as Software Engineer in Comviva Technologies Ltd, Gurgaon since 2009. He has done Post graduation from Thapar University, Patiala. He had worked on more than six professional/research projects. He is the author/co-author of four publication in international conferences and journals. His research area of interest include Web semantics and machine learning particularly semantic clustering and classification. He had taken courses in their teaching career as of Data Structure, Web Technologies and Computer Graphics.

Shalini Batra, Author

Shalini Batra is working as Assistant Professor in Computer Science and Engineering Department, Thapar University, Patiala since 2002. She has done her Post graduation from BITS, Pilani and is pursuing Ph.D. from Thapar University in the area of Semantic and Machine Learning. She has guided fifteen ME theses and presently guiding four. She is author/co-author of more than twenty-five publications in national and international conferences and journals. Her areas of interest include Web semantics and machine learning particularly semantic clustering and classification. She is taking courses of Compiler construction, Theory of Computations and Parallel and Distributed Computing.