

Study of Information Retrieval Systems and Software Reuse Libraries

Usa Rungratchakanon and Hisham Haddad
CSIS Department
Kennesaw State University
Kennesaw, GA 30144

Abstract: *Classification of reusable software components is essential to successful software reuse initiatives and a critical feature of library development. This paper provides a survey of storage and retrieval methods and highlights the main characteristics of each class of methods. The work focuses on information retrieval methods with emphasis on Component Rank and Latent Semantic Analysis models that are applied to component classification. These models have shown to provide efficient storage and retrieval algorithms as they narrow the results of a user query and provide accurate component selection. Thus, leading to efficient reusable repository.*

Keyword: *Reuse Libraries, Asset Classification, Asset Retrieval, and Retrieval Methods.*

1. Introduction

The increasing complexity and variety of software assets are adding to the difficulty of classifying such assets within reuse libraries. During the last 30 years, research work on structuring software repositories has been conducted and many repositories have been developed. However, few software repositories remain usable and the efficiency of their classification and retrieval methods is unknown. Therefore, classification of reusable components is a challenge for both developers and reuse librarians because the successful deployment of software reuse is dependent upon the ability to accurately classify and retrieve reusable asset. Therefore, from user's viewpoint, the most important feature of a software library is the effectiveness of the library's classification and retrieval algorithm.

The objective of classification and retrieval methods is not only to promote, but also to facilitate reuse components. This work starts with an investigation of classification and retrieval methods developed over the past three decades. Mili and Mili [1] have investigated and classified storage and retrieval methods used in the organization of software assets into six classes: *information retrieval (IR) methods, descriptive methods, operational semantics methods, denotational semantics methods, topological methods, and structural methods*. However, the most common approach is IR methods, but these methods are simple as they return broad results for a user query. Thus, it is essential to apply

these methods with other concepts or models to narrow the search results. In this paper, we present two IR models that are widely applied to narrow retrieved software components from reusable libraries. The first model, called *Component Rank*, ranks each software component based on its relationships with other components. The other model, called *Latent Semantic Analysis (LSA)*, is a method for extracting and representing characterization of word meaning. Both models are described in section 3.

2. Survey of Storage and Retrieval Methods

Promoting software reuse requires support for easy identification and retrieval of reusable assets. Such support requires effective classification methods. This section provides analysis and summary of six classes of storage and retrieval methods used with software repositories. Tables 1 and 2 provide summary of attributes characterizing each of these classes of methods.

2.1 Information Retrieval Methods

This class of methods depends on textual analysis of software asset descriptions using full-text search and keyword matching along with natural language queries to locate assets. However, the question is how accurate a method in retrieving assets that meets the user needs by getting relevant data without being distracted by the irrelevant data. An important aspect of an information retrieval system is to keep precision and recall as high as possible in order to achieve the reasonable cost and well-formulated queries.

2.2 Descriptive Methods

Descriptive methods depend on characterization of a software asset by keywords or metadata. This class of methods proceeds by matching a keyword query against assets that are represented by lists of descriptive keywords. These methods are appropriate for non-textual assets such as audio, video, or images, as the nature of these assets can be described with text that can be searched in much the same manner as the above information retrieval methods. Descriptive methods are universally applicable, irrespective of the nature of the asset to be described. However, this class of methods has some problems that even a complete match between descriptors cannot guarantee a match between the items thus described, or it can infer that the failure to find a matching descriptor occur because there is no matching

component. Therefore, the performance of descriptive methods depends on precise interpretation of terminology, and the terminology must be shared between users and administrators. In addition, the scope of the library is limited.

2.3 Operational Semantics Methods

Operational semantics methods are dependent upon the operational semantic of software assets. They exploit the executable nature of some assets by comparing input specified by a search query to output produced by stored assets. Software component can be uniquely identified within a large software library on the basis of its behavior on a few randomly selected sample inputs. Due to the nature of these operational methods, the scope the library is limited to those that can be executed, such as compiled source code or executable UML, and those assets are represented by means of executable source code segment. The storage structure is a flat list of components that have an *equivalence relation on signatures*. All components that have the same signature are stored together. An *ordering relation on signatures* is that a component is greater than another if it has a superset of its parameters. An *ordering relation on behavior* is that a component is greater than another if its behavior subsumes the other.

2.4 Denotational Semantics Methods

Denotational methods are based on a semantic representation of software assets; retrieval takes place by matching the semantic representation against a similarly denoted query. Denotational methods can be applied to non-executable assets such as specification, and they proceed by checking a semantic relation between the user query and a surrogate of the candidate asset. There are two types of software assets: executable software components and requirements specifications. This method is commonly used with executable code and requirements specifications. Most of the work on signature matching and functional matching deal with assets written in functional programming languages that exhibit the signature of a function in an explicit manner. These methods represent queries in a monolithic manner that means when the library assets are requirements frameworks, the query is represented by the definition of a missing requirement; and when the asset is a proof, the query is represented by a concrete theorem.

2.5 Topological Methods

Given a query that describes some required features of an asset, topological methods identify assets that most closely match a description of the features provided in a formal specification language. These methods are divided into two categories according to the precise definition on their retrieval goal: *Exclusive approximate retrieval* methods make a distinction between exact retrieval and approximate retrieval. *Inclusive approximate retrieval* methods make no distinction between exact retrieval and approximate retrieval. They focus on identifying assets

that minimize some measure of distance to the query and expect that the outcome to be either an exact match or one or more approximate matches.

The measure of distance can be divided into two classes: *Measure of functional (semantic) distance* (assets act a like) reflects the extent of similarity between the functional properties of the query and those of candidate components. *Measure of structural (syntactic) distance* (assets look a like) reflects the extent of similarity between the structure of solutions to the query and the structure of candidate components. Topological methods are typically used for executable code. Thus, any asset that can be represented by text in the library's specification notation qualifies as an asset. The scope of the library depends on measure (structural or functional) of distance. Moreover, These methods are not uniform of any representation, and are not dependent on storage structure.

2.6 Structural Methods

Structural methods seek to retrieve assets based on their similarity to a specified structure. Most components follow the two basic matching measures: functional or structural, so that the user can decide whether to select an asset by matching its functional properties against desired functional features. It is suitable to choose the functional criterion (it can imply to black box reuse) if the intent is to retrieve the asset without modification. However, using the structural criterion is best for seeking assets that was modified. Source code assets are commonly used with these methods. but all assets must be represented according to a system of syntactic patterns dependent on the library implementation. Such systems are relatively rare and exist only in prototype systems, perhaps because of the difficulty in overcoming the representation language.

3. Information Retrieval Methods

The traditional IR approach depends on textual analysis of software asset such as full-text searching and keyword matching. As IR is a discipline of the traditional retrieval in the library, it uses test-based search to query reusable assets. The focus of stored information is on textual information, and thus natural language queries are used to locate assets. IR relies on the construction of the library, which is done in two steps. First, attributes are automatically extracted from natural language documents using a new free index scheme, then a hierarchy for browsing is automatically generated using a cluster technique that draw only on the information provided by the attributes. However, using only traditional information retrieval would return a board results.

3.1 Component Rank Model

The Component Rank Model is based on a graph representation scheme of the components, and is derived from the concepts of component graph and reuse frequency. This model is a collection of components represented as weighted directed graph. One node on the graph represents one software component, and a directed edge linking the

nodes represents a use relationship between the components. On the other hand, an incoming edge means that a component is used or referenced by other components, while an outgoing edge means that a component uses other components. The total weight of the system is equal to 1. Each node has its own weight that can be calculated from all incoming edges, and each edge can be computed by the weight of node divided by distribution of outgoing edge. However, component can be developed at the different location and different time, or can be copied or modified from previous components. Therefore, the measure of similarity between components is essential. Similar components are clustered into one node so that duplicate components are removed. After that, nodes are ranked by their weights; moreover, each node is recomputed for reuse frequency. A node has a higher reuse frequency if many nodes use or depend on it, or if some nodes depend on it have a high reuse frequency. The resulting rank is used to prioritize the query result so that the user quickly sees highly ranked components.

Following these concepts, classification and retrieval using Component Rank is divided into two parts: “Constructing Database” part and “Searching Components” part. The “Constructing Database” part relates to how to classify each component in the system; each component is analyzed and stored in the archive with information. The component is extracted as tokens and then is indexed to form a dictionary for retrieval. If the component refers to or is related to other components, a use-relationship is created, and the similarity of two components is computed. Therefore, results from the use-relationship and similarity are analyzed to create clustered component graph. After that, all component graphs are calculated and ranked by the component rank and the result is stored in the ranked component. The “Searching Components” part relates to how to retrieve components from the library. The first step is entering the query keyword, and then components containing the same token as the query keywords are searched. All tokens are explored and the results are listed by the order of component rank.

3.2 Latent Semantic Analysis Model

The Latent Semantic Model is a method for the characterization of word meaning. Some librarians propose that Latent Semantic Indexing (LSI) can be the solution because it has the feature of synonymy and polysemy. Synonymy refers to the existence of similar terms to express the same idea, and polysemy refers to the fact that some words have multiple meanings. With synonymy and polysemy, the LSI space contains the index of components and the relevant components to the query. However, this method has a stated advantage over traditional IR methods that LSA does not make use of word order, syntactic relations, or morphology.

The classification and retrieval with LSI is divided into two parts: classification part and retrieval part. In the classification part, each software component document is

extracted to create the identifiers. After that the system removes the meaningless identifiers (the identifier that appears only in one software component or appears in more than half of all software components is a meaningless identifier). Next step, the system computes cosine of each identifier and performs system analysis. After that the system is converted into a corpus. The Single Value Decomposition (SVD) is applied to create the similarity of the identifier because similar meanings do not always share the same words. Therefore, the synonymy and polysemy are performed at this stage. In the final step, the system collects all identifiers and synonym to the LSI. The steps of the retrieval part start with typing the query keywords, and then the document that represents the query is created and mapped onto the LSI. Then the LSAI creates a corresponding vector v and returns the set of all components in the system, ranked by the similarity measure to the user query.

4. Discussion and Conclusion

The investigation of software repositories showed that most libraries rely on IR methods that depend on textual analysis of software component. However, using only information retrieval does not create efficient storage and retrieval system because it returns board results. Therefore, to improve the efficiency of an IR system, it should be applied with additional method to narrow the results of a user query. We also realized that the following issues should be considered before constructing the repository: relevance relation, ranking order, and similarity. Component rank and Latent Semantic Analysis are in conformance with those issues. Both models utilize relevance of each component; the similarity between components, and the order of the result. However, they are different in specific detail. While the similarity of Latent Semantic Analysis is based on the meaning and returns the result of a query ranked by the similarity measure of word meaning, the similarity of component rank is based on the structure and its results are not only ranked by the similarity, but also rank by reuse frequency. It should be noted that the similarity is divided to syntactic and semantic similarity and both models just use only one measure of similarity. Therefore, it is possible for the future work to have both type of similarity utilized to improve the IR system.

In summary, various classification and retrieval methods were analyzed, and related work is also analyzed in order to improve the effectiveness of information retrieval in software repositories. This work serves as a guide to those interested in exploring the issues of component classification and storage and retrieval in reuse libraries.

5. References

- [1] Mili, A., Mili, R., & Mittermeir, R. T. (1998). “A survey of software reuse” libraries. *Annals of Software Engineering*, 5, 349-414.
- [2] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, Shinji Kusumoto. “Ranking

- Significance of Software Components Based on Use Relations.” *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp 213-225, 2005.
- [3] Reishi Yokomori, Takashi Ishio, Tetsuo Yamamoto, Makato Matsushita, Shinji Kusumoto, Katsuro Inoue. “Component Rank: Relative Significance Rank for Software Component Search.” *Proceedings of the 25th International Conference on Software Engineering (ICSE2003)*, pp14-24, Portland, Oregon, U.S.A., May 6-8, 2003.
- [4] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, Jonathan I. Maletic. “An Information Retrieval Approach to Concept Location in Source Code,” *wcre*, pp. 214-223, *11th Working Conference on Reverse Engineering (WCRE'04)*, 2004
- [5] T. G. Bay and K. Pauls “Reuse Frequency as Metric for Component Assessment” *Technical Report 464, ETH Zürich, Chair of Software Engineering*, 2004.

Attributes	Information Retrieval Methods	Descriptive Methods	Operational Semantics Methods
Nature of library	No restriction on the nature of the asset.	Any type of reusable software asset.	Executable components.
Scope of library	Limit on the scope or project.	Limited by scope of common terminology of reference.	Typically, within product line (Application domain)
Query representation	Natural language queries.	One or more keywords per facet.	Oracle, and sample input data
Asset representation	Natural language, characteristic indices.	One or more keywords per facet.	Executable code
Storage Structure	Hierarchy using semantic links.	Any structure: Indexed representation or facet-based classification.	Typically flat
Navigation scheme	Follows the semantic link.	For each facet: linear.	Typically exhaustive search
Retrieval goal	Informal inclusive approximate retrieval.	Retrieving all assets that pertain to keywords.	Retrieving all correct components
Relevance criterion	Total or partial match between the query and the candidate component.	Keyword matches. (Any assets that deal with the topic of the query.)	Correct behavior on sample data
Matching criterion	Total or partial match between the query and the candidate component.	Keyword matches. (All keywords of the query match the corresponding entries of the asset.)	Correct behavior on sample data

Table-1: Characterization Attributes.

Attributes	Denotational Semantics Methods	Topological Methods	Structural Methods
Nature of library	Executable components, specification framework, proofs.	Unrestricted	Source code.
Scope of library	Typically, within product line (Application domain)	Typically, within project	Typically, unlimited
Query representation	Signature specification, functional specification.	Wide variance, from logic to AI	Name of a syntactic pattern
Asset representation	Signature specification, functional specification.	Wide variance, from logic to AI	Source code, or roles and constraints
Storage Structure	Typically flat. Some are hierarchical, some are partitioned.	Typically flat.	Typically flat.
Navigation scheme	Typically exhaustive search, some are selective.	Linear scan	Typically exhaustive search.
Retrieval goal	Retrieving all correct components	Inclusive or exclusive approximate retrieval	Retrieving all components of a given structure
Relevance criterion	Functional matching such as functional equivalence or refinement. Signature matching such as data equivalence or refinement.	Minimal distance	Structure match
Matching criterion	Weak (efficient) version of the relevance criterion.	Minimal distance	Name identity or successful parse.

Table-2: Characterization Attributes