EasyChair Preprint

№ 6055

Conceptual Model for Component Selection: A Resaerch Review on Existing Techniques

Nazia Bibi, Tauseef Rana, Qurat-Ul Ain and Ayesha Naseer

July 27, 2021

Conceptual Model for Component Selection: A Research Review on Existing Techniques

Nazia Bibi
Department of Computer Software Engineering
National University of Sciences and Technology
Islamabad, Pakistan
nazia.phdcse@students.mcs.edu.pk

Tauseef Rana
Department of Computer Software Engineering
National University of Sciences and Technology
Islamabad, Pakistan
tauseefrana@mcs.edu.pk

Qurat-ul-Ain
Department of Computer
Air University
Islamabad, Pakistan
Quratulain.raja15@gamail.com

Ayesha Naseer
Department of Computer Software Engineering
National University of Sciences and Technology
Islamabad, Pakistan
ayeshanaseer@mcs.edu.pk

***Abstract*—Nowadays, software developers pay more attention to component-related innovations with the implementation of software reuse; most of which have been used in the development of large-scale complex applications to improve software development efficiency and speed up time to market. The development of component-based software is a well-known technique that establishes the reusability of software and cost-effectively reduces development. The major issue, however, is determining how to select a component. The principle rationale of this paper is to give a reference highlight future exploration by ordering and arranging distinctive categories of component selection methods and accentuating their individual qualities and shortcomings. To achieve this, the rationale for employing these techniques, as well as the use of a hybrid mechanism in the component selection method, is demonstrated. Finally, a conceptual approach for component selection is proposed which utilizes existing methodologies. Hopefully, it can help researches to locate the current status of this issue and fill in as a reason for future exercises**

Keywords— software component; component reterieval; component selection

I. INTRODUCTION

The greatest challenge that organizations are facing in this age of data digitization is to have a quality software product in shorter period. One possibility would be that companies employ more software developers in less time to produce better products but that will increase budget amount. This is not a viable solution because organizations require less budget and time to deliver a high-quality product. Component Based Software Development (CBSD) provides a solution to this problem [1][2].

The trend for CBSD is growing [3][4] due to the availability of component market and the enormous increase in software repository components [5]. For the adoption of CBSD, various problems must be addressed. The first problem is component availability and maintenance; second, there is no perfect component search and retrieval technique that meets the requirements of users and the third problem is users need to learn software and languages [6] to enter the query in a specified format. Elementary approaches for component search and retrieval have been developed [7] but these approaches have some limitations.

CBSD is a systemic process of software development which is derived from the reusable components from different sources; for instance, local component repositories and commercial component vendors. The way applications are developed and passed on to the end-user is being transformed by CBSD. It shifts the paradigms of software development, primarily with the implementation of different standards of component design. CBSD is not very common among today's developers because it takes developers a lot of time and effort to search for the best eligible component [1][8].

In solving the problems of reuse, software component reuse has opened doorways for researchers. For code recommendation and efficient retrieval, researchers have invented different methods. The key problem, faced during the research, is how the component can be selected from the available list of components that meet the requirements [3]. In this paper, the component selection techniques are explained that help to select the components that can fulfil the requirements. This paper aims to examine the applicability of software component selection techniques from online repositories and to study the techniques. A conceptual model for component selection is described in a simple and efficient way after examining the strengths and limitations of the techniques. This is the first research, provides an extensive analysis of the component selection techniques. To conclude, in the context of a conceptual model, differences in existing research and possible future opportunities have been addressed and presented.

The rest of the paper is organized in three sections. Related techniques for component selection and retrieval along with the

existing problems are presented in Section II. Research methodology is presented in Section III. Section IV presents the proposed hybrid conceptual model for component selection and retrieval. Discussion and analysis are explained in Section V. Conclusion and future work is discussed in the final Section VI.

II. TECHNIQUES FOR SPFTAWRE REUSE AND RETRIEVAL

A. Keyword Based

A basic keyword-based technique is applied by text description analysis. Domain specific dictionaries of keywords minimize the dimensions of the function space. The observations of the classifier are contrasted with the classifiers of the decision tree and SVM. Figure 1 provides the architecture of the keyword-based technique.

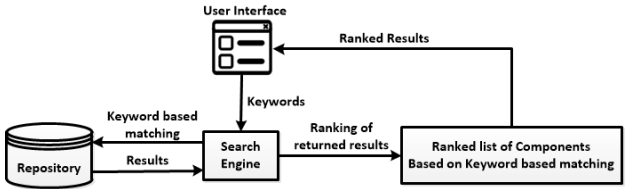


Fig. 1. keyword based Technique

The software component can be retrieved using an approach based on keywords. As the user is unaware of the corpus/schema and query processing language, this strategy masks the user's difficulty. An abstract interface is provided for the user to enter keywords, which are then submitted to the search engine. A graded list of components is returned by the search engine. Based on the keyword match and the occurrence of the keyword in the corpus, ranked components are given. Based on the frequency of keyword match occurrence, software components are ranked in descending order and the component with maximum or higher occurrence will get higher priority [9].

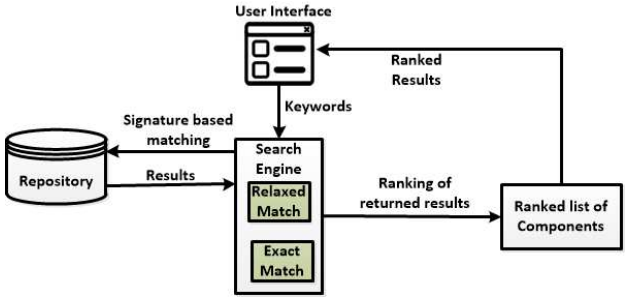
Example: For keyword search mechanisms, it is important that the person using it uses the type of terms that are applicable to what is being searched for. For example, if the individual using it was looking for how the stack is implemented, searching should consider the word stack.

Limitation*:* The fact that the number and selection of words is necessary for it to function is a problem that is often faced during keyword use. The use of one keyword can lead to a high recall, but lack of accuracy and the use of several keywords, on the other hand, will have the opposite effect. The quest becomes a boring process that results in trial and error before the user finds the correct one with a multitude of searches. To get the results, an experienced user is typically needed. Thanks to its user-friendly concept, the advantages of a keyword approach can be extended easily. To search a keyword, certain numbers of indices are required. For instance, the keywords that are linked with objects are represented as OBJECT1-KW1, KW2, KW3, and OBJECT2-KW3, KW4, KW5. If one searches for KW3 then the objects that are related to this keyword will be returned as a result. It will also retrieve a few unrelated objects. The cost is high in manual indexing, and for this, professional individuals are needed. The ambiguous nature of keywords, which can cause a wide discrepancy between keyword choices, is another shortcoming of the keyword process.

B. Signature Based

Signature matching is a way for software libraries to organize, navigate through, and perform retrieval process. Method signature is considered as a part of method declaration. If the extract name is used by various methods, only one method is selected during the method call. The signature of a method is specified by its name and its list of parameters. For a class, the signature of the system should be unique. The parameter name and return function form are not a part of the method signature [10].

Fig. 2. Signature based Technique



For component retrieval, the signature matching technique as provided in Figure 2 is also used as it decides which component is matched with a query signature. Other methods retrieve components based on exact query match, but there are components that are not exactly identical to the query, these components are similar in certain ways, so if the query or component is slightly changed, it will match a query [11]. For component retrieval, all cases (exact and relaxed) of query matching are therefore considered. With relaxed matches, the assumption is that the results are returned close enough to be useful for the developers. For example, relaxation matching will allow the input parameters of the library function to be reordered. Matching the signature in the most generalized form which is written as follows:

$$QuerySignature, MatchPredicate, ComponentLibrary \rightarrow Seto fComponents\ SignatureMatch(q, M, C) = \{\, c \in C : M(c, q)\}$$

In other words, when giving a query q, a match predicate M and a library component C, Signature Matching returns the set of components that met the matching predicate.

Example: For a queue of digits and stack of integers, take the signatures given in Table 1, these signatures are called isomorphs and therefore to show what we can list as the vocabulary problem. Users that reuse the program connect discrete semantics with unique names, such as pop and enqueue. Hence, the component developer can mislead the users towards the semantics of components by choosing names, or have no means of distinguishing between components.

TABLE I. SIGNATURE COMPARSION

Signature of a Stack	Signature of a Queue
Create:→Stack	Create:→Queue
Push: Stack x Integer →Stack	Enqueue: Queue x Integer: →Queue
Pop: Stack→Stack	Dequeue: Queue→Queue
Top: Stack→Integer	Front: Queue→Integer
Empty: Stack→Boolean	Empty: Queue→Boolean

Take another instance where we want to look for a signature-named component (int, int), i.e., a function that takes two arguments of the integer form and returns an integer value. It can contain a list of the following functions:

$$int\ sum\ (int\ , int)$$
$$int\ add\ (\ int\ , int)$$
$$int\ sub\ (int, int)$$
$$int\ mul\ (int, int)$$
$$int\ div\ (int\ , int)$$
$$int\ avg\ (int, int)$$

Depending on the activities, the given list has objectives with distinct actions. Matching of actions is performed to find the addition of two numbers. To test the already identified responses, the feature responds, and the contradictory behaviors have been deleted. The resulting list will appear as int sum (int,int) int add (int,int) int avg int avg (int,int). In all the functions given above, the probable behavior is shown. Here, the sum(), add() is set to "EXACT MATCH" and avg() is set to "RELAXED MATCH".

Limitation: With structural classification, there are many limits. First, it is only targeted at the reuse of the white box, so it only looks for approximate part retrieval. Second, these approaches are predominantly for laboratory use to date. Third, the proof theorem that must be matched in the library is very complex and last, when the application engineer knows a partial or a full signature description of the necessary variable, the signature matching technique is useful.

C. *Faceted classification*

A faceted classification is a method of classification used to arrange information into a hierarchical order. To construct the complete classification entry, a faceted classification uses semantic categories, which combines either general or subject-specific categories. The method of classification and recovery of facets is the most detailed [2]. A word is placed in the specified sense of the language and is defined by a particular angle of view (called facet) representing the critical feature of a software component in the facet classification [3][4], a facet being a fundamental feature identified in a domain. A software component is defined from different profiles by each facet, a component can be represented by several facets and many words in a facet, a component can be described by different facets from different angles. In a facet, organized term space is generated by common and special relationships; there are a number of terms [12].

Example: Experts extract keywords from program specifications and documentation for faceted index approaches and organize the keywords by facets into a classification scheme, which is used as a standard software component descriptor. For each facet, a thesaurus is extracted to solve ambiguities, to ensure that the matched keyword may only be within the sense of the facet.

Limitations: Faceted classification and retrieval has proven to be very effective in retrieving reuse component from repositories, but the approach is labor intensive.

D. *Behavioral Matching*

The behavioral retrieval technique to retrieve the repository components was described by Qualid Khayati [13]. This focuses primarily on the component's behavior. By leveraging the executability of software elements, behavioral retrieval works. Programs are executed using components and the responses of component are recorded. Retrieval is done by selecting those components whose responses are nearest to a pre-determined set of desired responses (with respect to the program). Originally, this concept was called "behavioral matching". To retrieve the components that display the predicted actions, behavioral matching usually executes each component with input data. To retrieve the components whose behavior is predicted, the process of behavioral matching usually executes each component with input data [12].

Example: For example, both integer addition and subtraction have the same signature, yet completely opposite behavior; strcpy and strcat have the same signature for the C library routines, but if one was replaced by the other, users will be unhappy. The component's behavior should conform to the specified behavior (specification) for its effective execution.

Limitations: First, it uses randomly selected "samples" to perform all software library operations with a signature matching the appropriate one. Second, the developer has to measure the predicted outcome by hand, the technique obviously has some resemblance to what we call today's black-box testing.

E. *Semantic Matching*

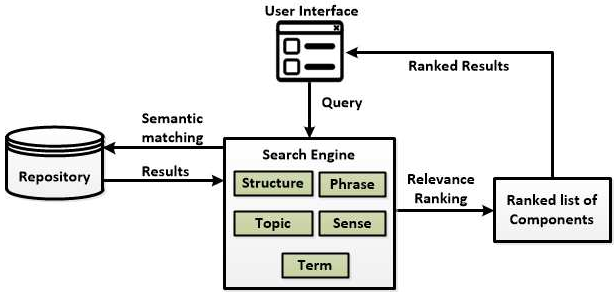
The technique of semantic matching relies on semantic knowledge. Lightweight ontologies are used to encode the semantic information that is to define and linked nodes semantically. Given two graphs (like structure and classification), an ontology matching operator works to classify or align the nodes of two graphs that are semantically related to each other. The component is obtained based on relevance, which is if the query and the component's recorded aspects are same, then the component is referred to as the candidate component. In practice, the more aspects that are similar between query and the component, the greater the correlation exist between them. So by using semantic matching, the problem of mismatch can also be resolved [14].

Example: For example, when applied to a file system, the "source code" folder may be found which is semantically similar to the "code" folder since it has the same meaning. You may obtain this data from a linguistic resource, such as WordNet. As researchers do not make a clear distinction between them, the terms semantic matching and semantic

search are distinct. The database is searched by query as far as semantic matching is concerned, where both query and document are called unstructured data. Semantic search often searches the knowledge base and database by query, where it is not inherently important to format the document and query, but structured data is the knowledge base [6]

The main explanation for the mismatch (between the query and the stored component) is that there was no proper language analysis [15]. Language cannot be understood by the machine, but it is not impossible. In contrast with other approaches, the semantic matching approach is more practical (e.g., Keyword and signature). Semantic matching involves phrase analysis, word normalisation, topic analysis, structure analysis, word sense analysis, and matching between the question and part is carried out on word sense aspect, form aspect, phrase aspect, structure aspect and topic aspect as shown in Figure 3.

Fig. 3. Semantic based Technique



Limitations: This technique is in contrast with a drawback for words with different meanings, but on the other hand, is with the benefit of multiple senses. While this drawback is not important, it is clear that there is no processing advantage for terms with multiple definitions, contrary to the agreed view in the literature. In addition, when the stimuli were chosen to mitigate the influence of word senses, Rodd et al. (1999) found a major disadvantage in visual lexical decision for terms with more than one context, compared with unambiguous words. Previous claims of an ambiguity benefit may therefore be the product rather than their numerous meanings of multiple senses which have high-ambiguity stimuli.

III. Research Methodology

This analysis was carried out according to the system used by Kitchenham and Charters [16], which, in turn, is divided into three phases.

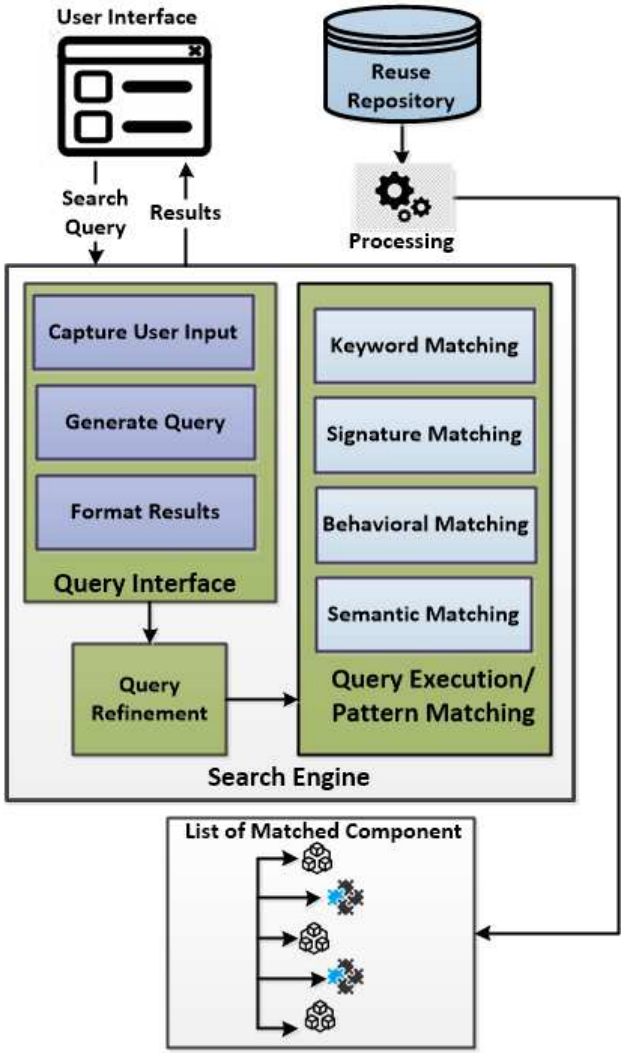
- The first stage includes the planning of the analysis and the creation of the study protocol to be followed.
- The second stage involves carrying out the analysis, then conducting the search, selecting the documents, and extracting and synthesising information
- In the third level, the analysis is reported and the distribution mechanisms and formats of the main report are specified. Such a step contributed to the elaboration of this paper in this context.

IV. Proposing a Conceptual Model

In this section, we propose a hybrid conceptual model for the selection of components that fills the gap between the prospective perspective of the present and the future. We infer from the previous section that several methods have been used for the collection and retrieval of software components. But, as we discussed in Section II, each solution has its own strengths and weaknesses. We referred to the suggested conceptual model as “hybrid” because it combines the features of all the approaches explained in the previous section.

The user defines the criteria that he/she is interested in using natural language for the components (nominal or imperative sentences). Initially a query is generated and augmented and takes into account along with domain-specific knowledge derived from specific domain models, within the context where the user performs the search.

Fig. 4. Conceptual Model



Keywords from the user’s query are mapped to the domain ontology and appropriate terms derived which are used in the search query. If required, ontology is also used to extend the

query with synonyms to expand the search. The proposed hybrid model thus offers a natural and versatile way for the user to determine component search specifications. At the same time, since it uses specific domain knowledge, efficiency is greatly enhanced. In retrieving these, the hybrid approach also considers relationships between components, resulting in a consistent collection of components as it combines the set of features of different approaches as shown in Figure 4.

A. *Working of Proposed Conceptual Model*

The proposed component retrieval approach consists of the 3 steps mentioned below: a) initial generation of queries, b) refinement of queries, c) retrieval and feedback of components. All of these are set out below.

- **Step 1:** Initial Query Generation

  The user can check for the component by entering the particular requirement of the component by entering insignificant sentences or imperative. Examples of the procedure used include: 1) a set of sentences or terms specifying the commencement of a query; 2) a set of pronouns that can be removed by parsing; 3) articles that can be removed by parsing; and 4) heuristics to create SQL statements.

- **Step 2:** Query Refinement

  Depending on the requirements entered in the form of a search query, the search engine will select one of the methods. If the keyword is matched with the stored component description in the repository, the user is presented with the output data, which is the list of components. It is also possible to search for the user requirement against the method signature defined in the repository. The search engine can conduct a behavioral and semantic search if the match is still not found. To make sure that correct phrases are used to query, the concepts and keywords defined in the former step are mapped against ontology. To expand the query, similar words which are based on the context of the retrieval are also defined. By the domain model, the context of the retrieval is created. For example, if the consumer is interested in implementing a specific auction site function, the corresponding objective is used as the anchor point within the auction domain model to decide the required processes and activities that must be part of the system and the appropriate components to support them [17].

- **Step 3:** Component Retrieval

  The component recall process starts when the user enters the query. Since the search engine incorporates the impact of several approaches as defined in Section II, null is not returned to the user query. The user-specified functional requirement is broken down into particular processes and activities using the domain model. They are then compared to the definition stored and the percentage of behavior endorsed by a specific component shows how important it is to the requirement of users. Objects are retrieved automatically that match a certain threshold value (e.g., a percentage). The user will assess the utility of the retrieved component and provide input on the basis of which the threshold value can be modified [18].

V. DISCUSSION

Numerous software component selection techniques have been proposed to enhance the proficiency and usefulness of informal procedures. Current processes of selections are not enough to discourse the detail and assessment of both the functional and non-functional requirements. Designing and developing of information system within very little time endures being a stressful process. This hassled situation has revived the research on the reusability of software especially in the development of component-based software. Several techniques for software component retrieval for instance signature matching, behavioral matching, faceted classification, semantic matching, and keyword matching, etc. have been discussed in the literature. The keyword approach can be a cause of inappropriate information and inconsistency in choosing keywords due to the uncertain nature of keywords. The approach of semantic matching becomes useless if the terms contain numerous definitions or variant contexts. If we take behavioral matching into account, the developers are required to predict the outcome on a manual basis. Furthermore, faceted retrieval and classification can be recognized as the most effective technique to retrieve the reusable components from repositories, however, a lot of labor is required for successful results.

The proposed hybrid approach for searching and retrieving software components can be much advantageous to the organizations practicing “reusability”, the commercial reusable components providers and distinct software developers as well. Developers are facilitated to discover repositories, execute all the plain text searches required to components and consequently, collect a group of all the possible components. Through this entire collection, the developers are enabled to execute a more precise search by utilizing the domain knowledge that is contained in certain domain models and ontologies. The main resolution of this research is to implement the common queries that can be used by a developer to specify the parameters of the search for retrieval of components i.e., size, run-time platform, or further technical contemplations. It can be useful for the developer if he/she tends some general queries like, “Give me all the components implemented in JavaBeans.” The designer can be concerned with this query to understand that what can he/she access from the repository that is relatable to the certain application which he/she is designing using a specific architecture.

VI. CONCLUSION

To enhance the proficiency and usefulness of informal procedures, the methods of component selection have been presented. The goal of the research is to discover such methodologies that can be used for the development of some tools and techniques to implement inter-enterprise information systems through reusable components and the development of reusable software also. This research paper is focused on experimentation related to the testing of various methods of component retrieval and indexing. Decisions about component selection are usually made on an ad-hoc basis. Inconvenient

search and retrieval of those reusable components which fulfil all the requirements can be considered as one of the chief issues that are related to the development based on components. However, the deficiency of sophisticated query procedures and techniques is the reason behind this issue. A hybrid approach proposed in this research allows the users to implement queries in an intelligent manner using techniques of NLP (natural language processing) and domain knowledge. The rationality of utilizing has been shown using hybrid mechanism applied in the method of component selection, which finally presented the theoretical approach for selecting components utilizing current practices. By entering the nominal or imperative sentences and exact requirements of the components, the users are allowed to search the components. The user can retrieve the components right after the query is entered. The query entered by the user cannot be nulled because the search engine syndicates the effect of many approaches like names. By using the domain model, the functional requirement of the user is divided into precise methods and actions. Those objects that qualify a specific threshold value can be retrieved accordingly. This specific threshold value can be adjusted according to the feedback of users after determining the utility of the component that is retrieved.

REFERENCES

[1] Y. Wahab, M. I. Babar, and S. Ahmed, "Single Repository for Software Component Selection (SRSCS): A Reusable Software Component Selection Technique.," J. Theor. Appl. Inf. Technol., vol. 26, no. 1, 2011.

[2] S. Younoussi and O. Roudies, "All about Software Reusability: A Systematic Literature Review.," J. Theor. Appl. Inf. Technol., vol. 76, no. 1, 2015.

[3] A. W. Brown, "An overview of components and component-based development," in Advances in Computers, vol. 54, Elsevier, 2002, pp. 1–34.

[4] H. Jain, "Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise." Taylor & Francis, 2001.

[5] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: An internet-scale software repository," in Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation, 2009, pp. 1–4.

[6] L. Kaur and A. Mishra, "Software component and the semantic Web: An in-depth content analysis and integration history," J. Syst. Softw., vol. 125, pp. 152–169, 2017.

[7] H. Mili, P. Valtchev, A. M. Di Sciullo, and P. Gabrini, "Automating the Indexing and Retrieval of Reusable Software Components.," in NLDB, 2001, pp. 75–86.

[8] L. F. Capretz, M. A. M. Capretz, and D. Li, "Component-based software development," in Industrial Electronics Society, 2001. IECON'01. The 27th Annual Conference of the IEEE, 2001, vol. 3, pp. 1834–1837.

[9] H. Mili, E. Ah-Ki, R. Godin, and H. Mcheick, "An experiment in software component retrieval," Inf. Softw. Technol., vol. 45, no. 10, pp. 633–649, 2003.

[10] O. Khayati and J.-P. Giraudin, "Components retrieval systems," 2002.

[11] S. Singh, "An experiment in software component retrieval based on metadata and ontology repository," Int. J. Comput. Appl., vol. 61, no. 14, 2013.

[12] J. Kaur and P. Tomar, "Clustering based architecture for software component selection," Int. J. Mod. Educ. Comput. Sci., vol. 11, no. 8, p. 33, 2018.

[13] H. Ben Khalifa, O. Khayati, and H. H. Ben Ghezala, "A behavioral and structural components retrieval technique for software reuse," in 2008 Advanced Software Engineering and Its Applications, 2008, pp. 134–137.

[14] V. Sugumaran and V. C. Storey, "A semantic-based approach to component retrieval," ACM SIGMIS Database DATABASE Adv. Inf. Syst., vol. 34, no. 3, pp. 8–24, 2003.

[15] Y. Shao, M. Zhang, and S. Xu, "Research on decision tree in component retrieval," in 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery, 2010, vol. 5, pp. 2290–2293.

[16] B. A. Kitchenham, "Systematic review in software engineering: where we are and where we should be going," in Proceedings of the 2nd international workshop on Evidential assessment of software technologies, 2012, pp. 1–2.

[17] N. Omer, S. K. Jha, and S. K. Khatri, "Maintaining Reusable Software Components," in 2019 International Conference on Intelligent Computing and Control Systems (ICCS), 2019, pp. 1350–1352.

[18] K. R. Sekar, J. Sethuraman, and R. Manikandan, "A Novel Software Component Selection Through Statistical Models," in 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), 2018, pp. 1391–1396.