Software Component Identification and Composition

State of the Art Report

Submitted in Partial Fulfillment of Requirements for the Degree of

Masters of Technology

By
Anshuman Sekhar Dash
M.Tech. 3rd Sem
Information Security
Reg No: 2020IS04

Under the Supervision of
Prof. D. K. Yadav

Computer Science & Engineering Department
Motilal Nehru National Institute of Technology Allahbad
Prayagraj, India

DECLARATION

I, hereby declare that the work presented in this **State of the Art** report dissertation entitled "Software Component Identification and Composition" has been done by me under the supervision of Prof. D. K. Yadav, and this dissertation embodies my own work.

Name : Anshuman Sekhar Dash

Registration No. : 2020IS04

Date : December 13, 2021

CERTIFICATE BY EXAMINERS

Computer Science & Engineering Department
Motilal Nehru National Institute of Technology Allahbad
Prayagraj, India

This is to certify that the **State of the Art** report entitled **Software Component Identification and Composition** is submitted by ANSHUMAN SEKHAR DASH (Registration No. 2020IS04) has been examined by the undersigned as a part of the examination and fulfills the requirement of Master of Technology in Software Engineering, MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY ALLAHABAD.

Signature	Signature	Signature
Name	Name	Name

Abstract

Reusing of software is found to have escalate the software engineers work output and also the craftsmanship of the final software product. Component based software engineering (CBSE) is a unique area in software engineering wherein software components i.e a chunk of software that contains a set of related functions and data are searched for and pieced together to form a software system.This method promotes re usability and efficiency in development of software systems. CBSE's purpose is to compose software applications using plug and play software components on the framework.There are several strategies for software component identification in a system among which most widely used is object oriented based clustering.we will look into various approaches used in identifying a software component.We propose a approach which is based on clustering and formal methods to solve the problem of software component identification from a software system.The approach is based on describing a software component using various attributes like Domain, Type, Visibility and Language then using clustering algorithms that will groups the components in an hierarchy.when components are arranged in an hierarchical manners it provides a means to store ,browse and retrieve reusable software components.We will then identify and select software components by taking in the requirements from user, extracting the keywords.The keywords will be searched using the hierarchy and then string based searching will be used to compare functionality of the subset of components and requirements keywords.The comparison will generate a similarity score which can be used to select software component.we will use the software components identified to compose a different system for comparison and demonstration of correctness of our approach used.

Contents

List of Figures

1 Motivation

Throughout last decade years wide spread use of software systems in many aspects of life and economy has placed new demands on the software industry for faster development of and software and also to revamp the quality of software and optimizing time take in developing of software.One method of achieving this is to promote software re-usability software re-usability means reusing of software previously developed which fulfills or nearly fulfills the software requirement As is the case with many software system , no two software systems are developed the same way.Every software system differs in one or more requirements and hence software re-usability will be limited unless a software system can be divided into software components each of which can be independently used in software development This is the reason for looking more into component based software engineering, various techniques used to identify software components in a system and then composition of software systems using components.

2 Introduction

2.1 Software Reuse

The main task of reuse system is to describe the reusable components ,insert them into a software component repository and get them whenever need arises.This involves indexing of software components at various level of abstraction.Proper reuse of software increase the cost effectiveness of developing software , decrease the time taken in development and maintenance of software system.
Component Based Software Engineering is a method developed for reusing of software. The whole process of reusing of software using Component based software engineering comprises of :-

1. Identification of components according to domain
2. Representation of software component in a form that can be easily done and understood
3. Storage of components in appropriate repositories after classifying
4. Selection of components according to the requirement specified

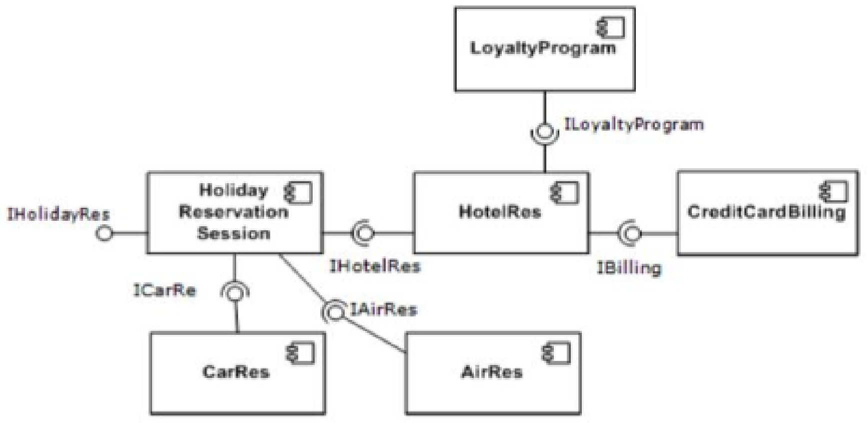


Figure 1: Software Components Example
[17]

2.2 What is Component Based Software Engineering

From the last decade years use of computer software has entered into every part of economy which has in turn generated new demand from the software industry to develop reliable and cost effective software quickly and efficiently. Modern software systems are becoming more and more large scale which further increases the development cost, decreases the productivity and increases the cost of maintenance. In the 90's a understanding was developed among software engineers that a software system need not be developed all the way from scratch and instead the software system can be developed by piecing together small pieces of software from previously developed software system and assembling them to make a final product.This idea was further developed into a field in software engineering named component based software engineering(CBSE).

In the context of CBSE a component will be defined as A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

In the work of [3] a component is defined as abstract, self-contained packages of functionality performing a specific business function within a technology framework.Figure 1 shows an example of software components.

This definition ascertains the fact that a software component is a well defined and independent software that gives it services using well defined interfaces.

Component-based software development is very well known and widely used as a methodology, which helps greatly in the re usability of software and reduces the cost of development significantly.

The primary role of CBSE is to direct making of software system as stitching of parts or components , development of the individual components as reusable entities and maintenance of the system developed with timely replacement of components.

2.3 My Approach

The idea for my approach to solve is taken from [17].Instead of using formal methods to describe components, Software components are stored in a software repository in a proper format by storing code and it's related attributes like Name,Language,Type,Visibility,Domain and Functionality.These attributes will then be used for unsupervised learning namely hierarchical clustering.Hierarchical clustering generates a dendogram or inverted tree using which searching for a component with specific set of requirements is easier.

In the work done this semester i studied various problems found in component based software engineering.The main problems studied were component identification and component selection.Going further deep into component selection problem i found that searching for a particular component from vast repository of software components is challenging.Reducing the search space according to some criteria will help in greatly reducing the search time and accurate software component selection.

I proposed a clustering based unsupervised learning approach that groups the software components by calculating similarity using various attributes of software component such as language, domain, visibility, type.Hierarchical clustering was used because searching using top down approach greatly reduces the search space.

3 Problem Statement

Component based software engineering uses software components that are already developed and stored in a repository for the development of a software system.With the growth of software industry the amount of software components available has grown vastly and the selecting the accurate component from a pool of component based on specific requirement has become a tedious task.

The problem faced in Component based software engineering is identifying and selecting a software component based on a certain requirement.The sub problems for this problem include

1. Developing a classification scheme that is used to store,maintain and group software components in a repository
2. Identifying and recording various attributes needed for describing a software component
3. Developing a technique to decrease the time taken to search for a component
4. Developing a method to calculate the accuracy of the retrieved component.

4 Literature Survey

The structure of component selection problem is that of optimization problem.Hence many optimization methods have been used in component selection.The survey paper [4] mentions various techniques used for software reuse and retrieval.

Keyword based technique uses domain based dictionaries that helps in minimizing the dimensions of search space.The keywords are extracted from a query processing language and based on the frequency of matching components are arranged in descending order[22].

Signature Based technique uses component signature for finding a particular component.Query matching is done by doing matching with query signature and component signature.Both exact matching and relaxed matching cases are considered and the results are shown by ordering from exact to relaxed matched[24].

A faceted classification is a method of classification used to arrange information into a hierarchical order.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[18]The software component is defined using different facet and the organized term space is generated by common and special relationships.

Behavioral matching uses component's behaviour.It uses the executability of software components.The responses of the software component while in execution is recorded.Retrieval is done by selecting those components whose responses are nearest to a predetermined set of desired responses.This mostly uses input data for matching[19].

Semantic matching uses semantic information that are usually enclosed in lightweight ontology.The ontology matching operator works to classify nodes of two graphs that are similar 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[25].

5 Agglomerative Hierarchical Clustering

Cluster analysis or clustering is the project of grouping a set of items in the sort of manner that objects in the same group (called a cluster) are more comparable (in some experience) to every aside from to the ones in different group (clusters)[15]. It is a main task of exploratory information analysis, and a common technique for statistical data analysis, used in many fields, along with sample recognition, photo analysis, statistics retrieval, bioinformatics, information compression, computer photographs and machine learning[21].
Cluster analysis on its own isn't always one unique algorithm, but the preferred task to be solved. It may be completed by way of diverse algorithms that range extensively in their know-how of what constitutes a cluster and how to efficaciously find them. Popular notions of clusters consist of businesses with small distances between cluster individuals, dense areas of the facts space, periods or precise statistical distributions. Clustering can therefore be defined as a optimisation problem with multiple objectives. The appropriate clustering set of rules and parameter settings (such as parameters which includes the distance characteristic to apply, a density threshold or the variety of anticipated clusters) depend upon the person information set and supposed use of the results. Cluster analysis as such isn't always an automatic mission, but an iterative system of information discovery or interactive multi-objective optimisation that entails trial and failure. It is frequently essential to alter information preprocessing and model parameters till the end result achieves the preferred houses. There isn't any objectively accurate clustering set of rules, however as it became referred to, clustering choice and parameters lies with that of the programmer. The maximum suitable clustering algorithm for a

particular problem often desires to be chosen experimentally, except there's a mathematical purpose to prefer one cluster version over another. An algorithm that is designed for one form of version will generally fail on a information set that carries a greatly exclusive type of model. For example, k-method can't discover non-convex clusters.
Clustering algorithms can be separated into 2 categories.One is divisive algorithms and the other is agglomerative algorithms. For divisive algorithms the set X is further divided into a group of clusters using measure of dissimilarity which creates a finer partition. While in agglomerative algorithms a group of finer partitions are merged together using similarity measures to form a courser partition.Each intermediate steps helps in generating a hierarchy of clusters.It gives an understanding of the architecture of the system.The similarity measure is an important factor in the agglomerative hierarchical clustering algorithms. The agglomerative hierarchical clustering algorithms (AHCA) begin from a set of individual entities that are first allocated into small clusters which are then in turn allocated into larger clusters until reaching a final all inclusive clustering. We will be using Agglomerative Clustering since we are identifying components from a software system that is bottom up approach.

6 Proposed Work

From the discussions above it is found that there are two approaches for faster and accurate software component retrieval from a software component library.First approach is indexing while creating and adding software components to library.This approach has the disadvantages that it will be cumbersome for migrating all the previously created software components to a new format of storing software components for indexing.It is also difficult to make changes upon.Second approach is reducing the search space for component search using the requirements given.

I will be following the approach of grouping together various components available into various clusters and generate a hierarchy of software component clusters.

Using the cluster headings of the hierarchy and requirements query text a subset of components for further searching is produced.This subset of components then needs to be compared one by and one and added to the output if it matches the component requirement.

For fine grained searching simple keyword and string matching is done

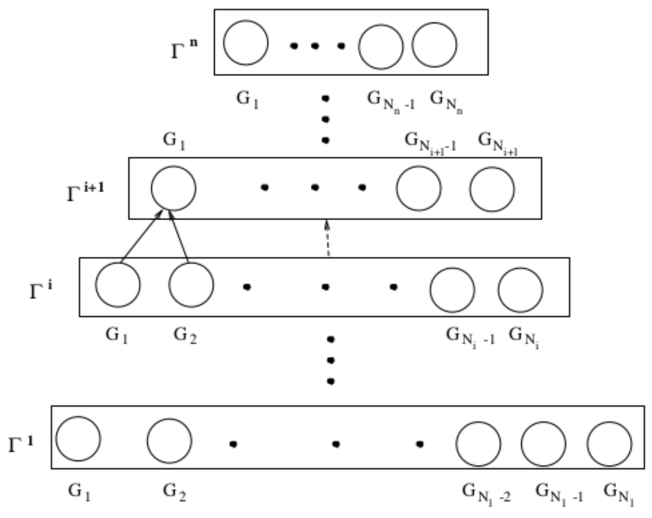


Figure 2: Agglomerative Hierarchial Clustering
[17]

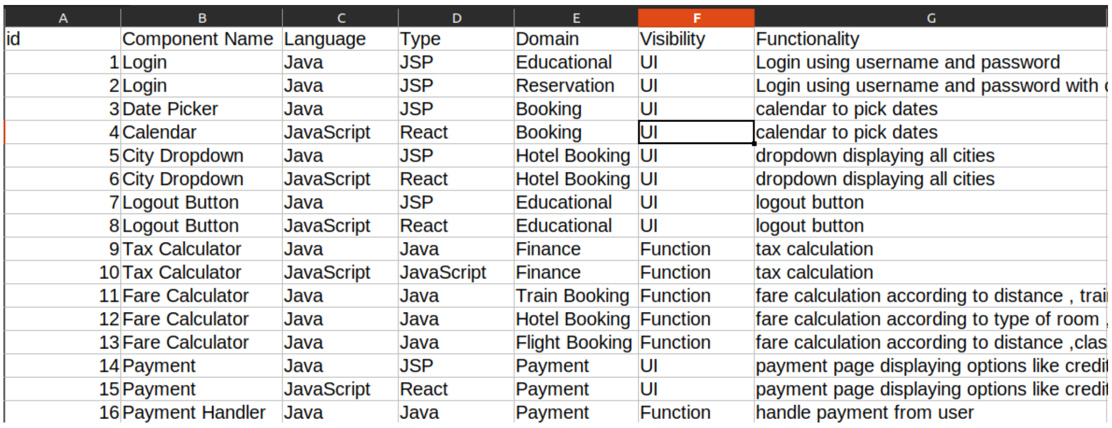
on the functionality attribute of the component.each comparison will generate a similarity score and in the end the components identified will be displayed in decreasing order of the similarity score.

6.1 Assumptions

1. The components available don't have external dependencies except language and library based dependency
2. The components are properly maintained with all the attributes
3. The requirements is very descriptive and giving language and type is a must

6.2 Working

The overall objective is to create a hierarchy of cluster of software components based on similarity of attributes.The hierarchy created will helps in reducing the search space for searching of components.A hierarchical organization of reusable components that will provide a fast means for browsing, retrieving and searching of software components[17].



A	B	C	D	E	F	G
id	Component Name	Language	Type	Domain	Visibility	Functionality
1	Login	Java	JSP	Educational	UI	Login using username and password
2	Login	Java	JSP	Reservation	UI	Login using username and password with
3	Date Picker	Java	JSP	Booking	UI	calendar to pick dates
4	Calendar	JavaScript	React	Booking	UI	calendar to pick dates
5	City Dropdown	Java	JSP	Hotel Booking	UI	dropdown displaying all cities
6	City Dropdown	JavaScript	React	Hotel Booking	UI	dropdown displaying all cities
7	Logout Button	Java	JSP	Educational	UI	logout button
8	Logout Button	JavaScript	React	Educational	UI	logout button
9	Tax Calculator	Java	Java	Finance	Function	tax calculation
10	Tax Calculator	JavaScript	JavaScript	Finance	Function	tax calculation
11	Fare Calculator	Java	Java	Train Booking	Function	fare calculation according to distance , trai
12	Fare Calculator	Java	Java	Hotel Booking	Function	fare calculation according to type of room ,
13	Fare Calculator	Java	Java	Flight Booking	Function	fare calculation according to distance ,clas
14	Payment	Java	JSP	Payment	UI	payment page displaying options like credi
15	Payment	JavaScript	React	Payment	UI	payment page displaying options like credi
16	Payment Handler	Java	Java	Payment	Function	handle payment from user

Figure 3: Software Components Library

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\big(\operatorname{tail}(a), b\big) \\ \operatorname{lev}\big(a, \operatorname{tail}(b)\big) \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) \end{cases} & \text{otherwise,} \end{cases}$$

Figure 4: Levenshtein distance between two strings a, b

- **Step 1: Data Preparation** The components available in the library are added by specifying attributes such as name, language, type, domain, visibility and functionality
- **Step 2: Similarity Calculation** For each component similarity is calculated with every other component based on the attributes language,type,domain and visibility.Functionality attribute will be used in later stage.The similarity is calculated by calculating adding the Levenshtein distance Fig:4 between the attribute values.Since less the distance more similar are the 2 values the distance is multiplied by -1 for proper ordering.
- **Step 3: Clustering** Agglomerative hierarchical clustering is a unsupervised learning algorithm.AHC algorithm is used on the software components using the similarity values.In each step of the algorithm two most similar components are grouped together and the similarity values are updated using the attribute values of this new merged component cluster.
- **Step 4: Keywords Generation** Once the hierarchy of clusters is ready

the software component requirement is taken from user and the query is broken down into individual keywords using natural language processing by removing stop words ,duplicates and punctuation.

- **Step 5: Search Component Subset** The cluster headings and keywords generated are compared and the hierarchy is followed .The components for fine grained searching are added.

- **Step 6: Fine grained Searching** Functionality of all the search component subset is compared individually with the keywords.A similarity score is calculated based on the formula $number\ of\ keywords\ matched/ number\ of\ words\ present\ in\ functionality$.All the components in the search component subset is displayed in decreasing order of the similarity score.

7 Algorithms Used

Algorithm 1 Component Clustering

```
Result: one or more clusters
Require: Input: The set X = { $x_1, x_2, .., x_n$ } $and\ the\ similarities\ s(x_i, x_j), 1 \leq i, j \leq n$
$Limit = 1, N = n$
for $1 \leq i, j \leq N$ do
    $sim(G_i, G_j) = s(x_i, x_j)$
end

while $N \neq Limit$ do
    $N$=$N$-1
    Find pair of clusters $G_p$ and $G_q$ such that $sim(G_p, G_q) = max\{x \epsilon G_i, y \epsilon G_j\ s(x, y)\}$
    update sim($G_i$, $G_j$)
    end
```

The above is the hierarchical clustering algorithm that is used.The input is all the set of components in the dataset.The output is a hierarchy of clusters formed using the components. First clusters are created using 1 component each.The similarity of each cluster is calculated using Levenshtein distance of the attribute strings formed for each cluster.The 2 most similar clusters are merged together i.e whose distance is least and new distance matrix is calculated using the unique attribute strings.These are performed until the number of clusters does not reduce to 1.

Algorithm 2 Fine grained Searching

```
Result: Similarity Score for every component in x with requirement
Require: Input: The set X = { $x_1, x_2, .., x_n$ }
$i = 1$
while $i \neq n$ do
    i=i+1
    matched=0
    for $keyword \leftarrow keywords$ do
        if $keyword\ in\ x_i$ then
            matched$\leftarrow matched + 1$
        end
    end
    similarityScore$\leftarrow matched \div len(x_i)$
    end
```

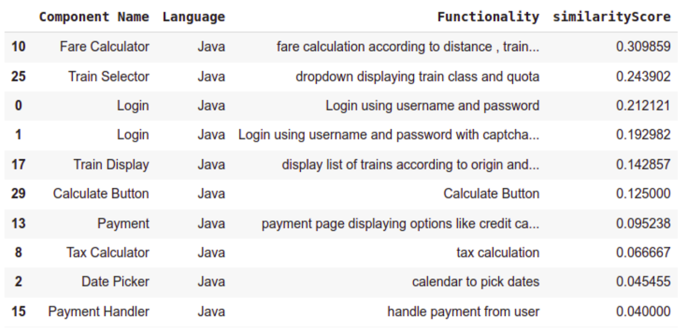
The above algorithm is Fine grained searching algorithm.This algorithm takes a subset of components as input and returns the similarity score of each component in the subset with the keywords string that is formed using the requirements.While looping through each component a keyword from keywords string is taken and if a word from functionality attribute of the component matches with the current keyword then count of matched is increased.after looping through all the keywords in keywords string the similarity score is defined as the ratio of number of components matched to that of number of words present in functionality attribute of the component.

At the end the components are shown with the similarity scored arranged in descending order of similarity score.

8 Results and Discussion

8.1 Results

The query text given is "A java component for railway reservation containing login using username and password with captcha functionality, date picker, dropdown for cities, train class and quota, calculation of fare with distance, train class and quota, dedicated payment page with options like debit card, credit card, upi, page to display list of trains"



	Component Name	Language	Functionality	similarityScore
10	Fare Calculator	Java	fare calculation according to distance , train...	0.309859
25	Train Selector	Java	dropdown displaying train class and quota	0.243902
0	Login	Java	Login using username and password	0.212121
1	Login	Java	Login using username and password with captcha...	0.192982
17	Train Display	Java	display list of trains according to origin and...	0.142857
29	Calculate Button	Java	Calculate Button	0.125000
13	Payment	Java	payment page displaying options like credit ca...	0.095238
8	Tax Calculator	Java	tax calculation	0.066667
2	Date Picker	Java	calendar to pick dates	0.045455
15	Payment Handler	Java	handle payment from user	0.040000

Figure 5: Components identified with similarity score

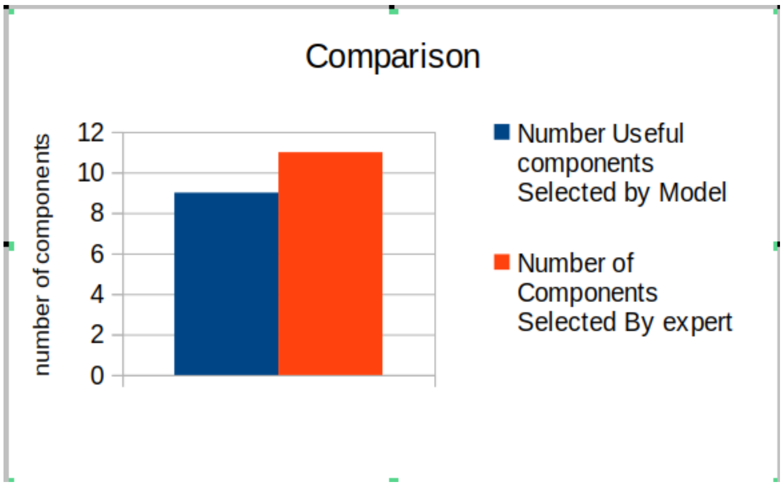


Figure 6: Comparison with an expert

All the above 6 steps were performed on the requirements given and components library.

The end result if displayed in Figure 5.This end result shows the components given by the model along with similarity score. This requirement was simultaneously given to an expert who used the software components library used to get the number of components that will be have to be used for this software requirement.The comparison between model generated component list and expert generated component list is displayed in Figure 6

8.2 Discussion

- The login component with captcha is found to have less similarity then simple username and password component on contrary it should be more indicating that our similarity calculation formula is not useful

- City selector component was not picked up in the subset of components for fine grained search indicating that clustering using attributes values which are string generates improper labels and misses out on some requirements.

Figure 6 shows that our model correctly predicted 9 components that may be required whereas that expert's output contained 12 components.

8.3 Technology and Framework Used

I have used Python to implement by hierarchical clustering.The dataset which contains components with attributes was prepared using spreadsheet program and stores as .csv file.Numpy array and pandas library was used using matrix.nltk library was used for natural language processing.

8.4 Work Progress

As of now clustering of components according to attribute values which are string values is performed which generates cluster headings based on the non matching attribute values.The searching of component is done directly by feeding the component requirement into the code.Fine grained searching used string comparison for generating similarity score.The overall results generated is not up to standard since the use is very limited and the generation of proper result is subject to proper query formation.using of string values to describe components inhibits in generating proper similarity scores and results.

8.5 Future Work

1. Using formal method approach to describe software component
2. developing the web application to enter proper requirement details
3. using weights to some key requirements like language and domain
4. develop a better fine grained search technique
5. using a feedback rating option so as to prepare dataset for performing supervised learning

9 Conclusion

For searching a component the requirements of the software component were taken in plain English language and keywords were extracted using natural language processing available in python by removing stop words and punctuation.The keywords were used to move through the hierarchy of clusters.A subset of probable components were formed using this.

The probable components were taken one by one and similarity score was calculated for each component.This similarity score was based on the ratio of keywords matching to that of number of words present in functionality of component.

The end result was very much dependent on proper English queries.The similarity calculation was based on string comparison which is flawed and does not generate proper result with clustering.

References

[1] Antonia Albani, Sven Overhage, and Dominik Birkmeier. Towards a systematic method for identifying business components. In *Component-Based Software Engineering*, pages 262–277, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[2] Gabriela Arévalo, Nicolas Desnos, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Formal concept analysis-based service classification to dynamically build efficient software component directories. *International Journal of General Systems - INT J GEN SYSTEM*, 38:427–453, 05 2009.

[3] Greg Baster, Prabhudev Konana, and Judy E. Scott. Business components: A case study of bankers trust australia limited. *Commun. ACM*, 44(5):92–98, May 2001.

[4] Nazia Bibi, Tauseef Rana, Qurat ul Ain, and Ayesha Naseer. Conceptual model for component selection: A research review on existing techniques. In *2021 International Conference on Communication Technologies (ComTech)*, pages 22–27, 2021.

[5] Dominik Birkmeier and Sven Overhage. On component identification approaches - classification, state of the art, and comparison. In *Component-Based Software Engineering, 12th International Symposium, CBSE*

2009, East Stroudsburg, PA, USA, June 24-26, 2009, Proceedings, volume 5582 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[6] Zhengong Cai, Xiaohu Yang, Xinyu Wang, and Aleksander Kavs. Afuzzy formal concept analysis based approach for business component identification. *Journal of Zhejiang University - Science C*, 12:707–720, 09 2011.

[7] B.H.C. Cheng and G.C. Gannod. Abstraction of formal specifications from program code. In *[Proceedings] Third International Conference on Tools for Artificial Intelligence - TAI 91*, pages 125–128, 1991.

[8] Jian Cui and Heung Chae. Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. *Information & Software Technology*, 53:601–614, 06 2011.

[9] Iria Estevez-Ayres, Luis Almeida, Marisol Garcia-Valls, and Pablo Basanta-Val. An architecture to support dynamic service composition in distributed real-time systems. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 249–256, 2007.

[10] Shabnam Gholamshahi and Seyed Mohammad Hossein Hasheminejad. Software component identification and selection: A research review. *Software: Practice and Experience*, 49, 10 2018.

[11] Maen Hammad and Rua'a Hasan Banat. Automatic class decomposition using clustering. *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 78–81, 2021.

[12] Mehdi Hariati. Formal verification issues for component-based development. *Informatica*, 44, 12 2020.

[13] Seyed Mohammad Hossein Hasheminejad and Shabnam Gholamshahi. Pci-pso: Preference-based component identification using particle swarm optimization. *Journal of Intelligent Systems*, 28(5):733–748, 2019.

[14] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili. Sci-ga: Software component identification using genetic algorithm. *The Journal of Object Technology*, 12:3:1, 01 2013.

[15] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili. Ccic: Clustering analysis classes to identify software components. *Information and Software Technology*, 57, 06 2014.

[16] Seyed Mohammad Hossein Hasheminejad and Saeed Jalili. An evolutionary approach to identify logical components. *Journal of Systems and Software*, 96, 10 2014.

[17] Jun Jang Jeng and Betty H. C. Cheng. Using formal methods to construct a software component library. In Ian Sommerville and Manfred Paul, editors, *Software Engineering — ESEC '93*, pages 397–417, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[18] Jagdeep Kaur and Pradeep Tomar. Clustering based architecture for software component selection. *International Journal of Modern Education and Computer Science*, 2018.

[19] Hanen Ben Khalifa, Oualid Khayati, and Henda Hajjami Ben Ghezala. A behavioral and structural components retrieval technique for software reuse. In *2008 Advanced Software Engineering and Its Applications*, pages 134–137, 2008.

[20] Soo Dong Kim and Soo Ho Chang. A systematic method to identify software components. In *11th Asia-Pacific Software Engineering Conference*, pages 538–545, 2004.

[21] Sunil Kumar, Rajesh Bhatia, and Rajesh Kumar. K-means clustering of use-cases using mdl. *Communications in Computer and Information Science*, 270:57–67, 01 2012.

[22] Hafedh Mili, Estelle Ah-Ki, Robert Godin, and Hamid Mcheick. An experiment in software component retrieval. *Inf. Softw. Technol.*, 45:633–649, 2003.

[23] Fatihah Mohd, Suryani Ismail, Masita Masila Abdul Jalil, Fatin Izatul Nadia Mohd Zulkarnain, and Norshuhani Zamin. A guidelines for controlled experimental design to evaluate the metrics of software component reusability. In *2021 Fifth International Conference on Information Retrieval and Knowledge Management (CAMP)*, pages 85–90, 2021.

[24] Shekhar Singh. Article: An experiment in software component retrieval based on metadata and ontology repository. *International Journal of Computer Applications*, 61(14):33–40, January 2013. Full text available.

[25] Vijayan Sugumaran and Veda Storey. A semantic-based approach to component retrieval. *ACM SIGMIS Database*, 34:8–24, 08 2003.

[26] Jiafu Tang, Li feng Mu, C.K. Kwong, and X.G. Luo. An optimization model for software component selection under multiple applications development. *European Journal of Operational Research*, 212:301–311, 07 2011.