International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

Graph-Based Keyphrase Extraction for Software Traceability in Source Code and Documentation Mapping

Nakul Sharma[1], Mandar Diwakar[2], Swapnil Shinde[3], Vishnu Suryawanshi[4], Varsha Jadhav[5]
[1]Dept of Artificial Intelligence and Data Science
Vishwakarma Institute of Information Technology
Pune, India
nakul777@gmail.com
[2]Dept of Artificial Intelligence and Data Science
Vishwakarma Institute of Information Technology
Pune, India
mpdiwakar30@gmail.com
[3]Dept of Artificial Intelligence and Data Science
Vishwakarma Institute of Information Technology
Pune, India
swapnil.shinde@viit.ac.in
[4]Department of Electronics and Telecommunication,
Trinity Academy of Engineering,
Pune, India
vishnusam2007@gmail.com
[5]Dept of Artificial Intelligence and Data Science
Vishwakarma Institute of Information Technology
Pune, India
varsha.jadhav@viit.ac.in

Abstract— Natural Language Processing (NLP) forms the basis of several computational tasks. However, when applied to the software system's, NLP provides several irrelevant features and the noise gets mixed up while extracting features. As the scale of software system's increases, different metrics are needed to assess these systems. Diagrammatic and visual representation of the SE projects code forms an essential component of Source Code Analysis (SCA). These SE projects cannot be analyzed by traditional source code analysis methods nor can they be analyzed by traditional diagrammatic representation. Hence, there is a need to modify the traditional approaches in lieu of changing environments to reduce learning gap for the developers and traceability engineers. The traditional approaches fall short in addressing specific metrics in terms of document similarity and graph dependency approaches. In terms of source code analysis, the graph dependency graph can be used for finding the relevant key-terms and keyphrases as they occur not just intra-document but also inter-document. In this work, a similarity measure based on context is proposed which can be employed to find a traceability link between the source code metrics and API documents present in a package. Probabilistic graph-based keyphrase extraction approach is used for searching across the different project files.

Keywords- Keyphrase Extraction;Software Traceability;Source Code Analysis;Text Mining;Program Comprehension.

I. INTRODUCTION

Keyphrase Extraction (KE) implies extracting relevant key-terms or phrases from the source. It is beginning phase employed in various computing tasks with the intent to extract the most relevant features. This includes extracting essential and relevant text from a given document [1]. It forms the part of text and data mining in which processing is done on text or data in general. KE finds close application with the various satellite fields of NLP and text mining such as Information Retrieval, Information storage, sentiment mining, opinion mining and other such related fields.

Source Code Analysis or code analysis implies analysing the source code written in standard programming languages. The source code must first be parsed before it can be understood hence, parsing of the source code entails scanning each keyword, literal, identifiers for enhancing better understanding of the code. There are different parsers which parse the source code through various means and extract relevant information. Several models, graphs, meta-data, developer's profile, developer's expertise can be extracted from such analysis of the code. Keyphrase extraction for source code involves parsing the code to extraction of useful key terms or phrases [2].

Traditional keyphrase extraction approaches can be applied on large source code projects owing to very high run time to extract relevant keywords. The relevant features can be extracted across the documents or within a document by studying similarity across these documents. The similarity can be in terms of the context or in terms of word to word similarity. In the literature of text mining, various similarity measures are proposed. The similarity hence generated can aid in creating a traceability link between different software artifacts across the SDLC.

Software's traceability mechanism needs a rigorous investigation and which can simulated by making use of Keyphrase extraction. The relationships existing between the different source code artifacts which Keyphrase extraction can also be done manually. However, this results in lots of time getting consumed especially in case of large textual documents or projects with large source code documents. Our current work focuses on a methodology of extracting keyphrases from different API documents within a package.

A. Importance of Keyphrase Extraction in Software Traceability

Software traceability aims to capture relevance of context and similarity as it occurs across different software documents. In order to accomplish successful software traceability, keyphrases occurring in different documents must be extracted. There are various text mining approaches applied in achieving keyphrase extraction [15]. Software Traceability needs these keyphrase as input in order to trace linkages across different documents.

II. LITERATURE REVIEW

Graph based extraction approaches have been enhanced by making using knowledge graphs. These knowledge graphs are more knowledgeable as the latent relationships existing between two or more keyphrases are introduced within the graph. There are various graph based approaches which are used on source code as well [3].

The keyphrase extraction can also be done in the process of conducting tasks such as multi-model retrieval. In this research authors use Fisher-LDA technique which is a extension of LDA technique to assign weights for each modality. The author's work included query reformulation but had textual keyphrase based component as well [4].

Graph based keyphrase extraction approaches are used in N-gram filtration technique. The authors here use statistical features present in the document as well the co-location strength. These approaches are hybrid as both statistical and numerical approaches are used in addition to lookup with Dbpedia text in achieving automated keyphrase extraction [5].

Graph based, language independent keyphrase extractor is developed by Litvak et. al. The proposed methodology is compared with standard metrics of precision, recall, f-score.

The tool gave better improvement in the precision and recall when compared with Text rank methodology. However, the proposed tool only gave a fare improvement for recall and f-score metrics [6].

The feature extraction is made better by first conducting feature selection. The feature selection strategy can be improved using optimization techniques for getting smaller number of useful features. This technique is proposed by Huang [7].

Automatic keyphrase extraction is proposed by yang et.al. The author considers relationship between words and sentences while extracting keyphrases. The graph based approach constructs graph between sentences, words separately and also between both sentences and words. K-means clustering is used for clustering related terms together. The evaluation metrics used are precision, recall and f-measure [8].

Research for automated keyphrase extraction and ranking has been done for short documents by Marina et. al. The proposed a new framework for generating topic based keyphrases and ranking them according to a well-defined ranking function. Ranking function had characteristics of coverage, purity, phraseness and completeness [9].

Top K-keywords and top K-documents extraction is done by ciprian et. al. The authors utilized weighed vocabulary for creating top K-keyword. The extracted top K-keyword evolves further to provide a document level top K documents. The benchmark used for evaluation were Okapi BM-25 weighting schemes, relational databases, document oriented database implementation [10].

Overlapping phrases are used for extracting accurate keyphrases by Mounia et. al. The authors make use of DPM-index for overlapping keyphrase in text document. The author's use supervised learning system. Semeval-2010 corpus is used in evaluation of the proposed methodology. The precision and F1-score are used for comparison [11].

There have been considerable attempt to map source code and its associated documentation by several authors [12][13].

Authors propose a modified version of Abstract Syntax Tree (AST). The tools uses IntelliJ IDE for extracting relevant code syntax and developing insights for code analysis [12].

Amir and Amir develop a bag of words and conduct similarity of source code based documents using such information. It is noted by authors that several variations in similarity calculation can be done [13].

Authors conduct unsupervised keyphrase extraction using position-rank algorithm. The approach proposes a graph from unique words and generates a position-biased ranking system using these graphs. The candidate phrases are then scored by calculating sum of individual words [14].

***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

Table-1 gives the summary of different research paper's methodology and the technologies which are used in comparing the proposed methodology.

TABLE I. RESEARCH METHODLOGIES USED AND COMPARATIVE ANALYSIS DONE IN LITERATURE

Ref.	***Title***	***Type of Keyphrase Extraction***	***Methodology Used***	***Comparison Done With***
3	Keyphrase extraction using knowledge graphs	Automated	Distance computation, clustering	Single Rank, Expand Rank, Sccocurance, SCWiki
4	Multimodal retrieval using mutual information based textual query reformulation	Automated	Fisher-LDA,KEA	Wiki parallel dataset
5	A graph-based unsupervised N-gram filtration technique for automatic keyphrase extraction	Automatic	Statistical feature, Co-location strength, N-gram graph	Sem Eval 2010, Duc 2001 dataset
6	Degext: a language-independent keyphrase extractor	Automatic	Graph based keyphrase extraction	Text Rank
7	An efficient automatic multiple objectives optimization feature selection strategy for internet text classification	Automatic	Multiple objective optimization	Reuters-21578,Newsgroup-20, KI-04, 7-Web
8	A graph-based approach of automatic keyphrase extraction	Automated	keyphrase extraction in relation to topic modelling	Hulth2003, DUC2001
9	Automatic construction and ranking of topical keyphrases on collections of short documents	Automated	Ranking By Topic	bLDA, Newton-Raphson iteration method, kpRelInt, kpRel
10	Benchmarking top-k keyword and top-k document processing with T2K2 and T2K2D2	Automatic	Creating benchmark using the most essential K keywords and documents.	Okapi BM25, TF-IDF

III. PROPOSED METHODOLOGY

This section describes the proposed work. The first section of proposed work is keyphrase extraction and construction of source code dependency graph. The second section is calculating the graph based similarity related to the source code from the constructed graph. The proposed methodology is shown in figure-1.

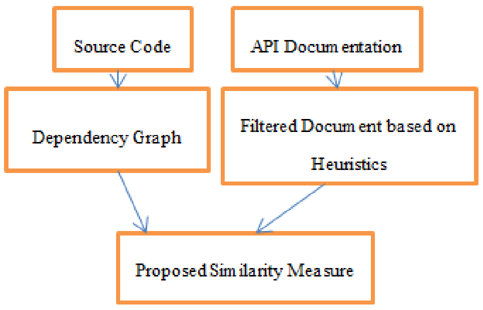

Source Code
API Documentation
Dependency Graph
Filtered Document based on Heuristics
Proposed Similarity Measure

Figure 1. Proposed Methodology

A. *Keyphrase Extraction and Source Code Dependency Graph*

This section involves parsing each source code file and creating the keyphrase dependency graph. The algorithm is titled as Source Code to Keyphrase Extraction (SC2KE). The algorithm for extracting the keyphrase from the source code is presented in Figure-2.

```
INPUT: Source code files
Parameters: path to source code directory
START
Step:1 Read the input source code file
Step: 2 Parse the source code
Step : 2.1Apply document pre-processing using NLP parser
To do tokenization, stemming of each word.
Step : 2.2 Extract Methods , Variables, Class name from Each File
Step: 2.3 construct a tabular representation for each file as follows
For Every Document D in Package, apply Annexure-I to segregate the textual documents with non-textual documents.
 For Every filtered Document D in package,
Do,
1. Perform Basic NLP tasks of Tokenization, stemming, stop word removal using Stanford NLP Parser.
 2. Extract programming constructs, keywords, literals to create bag of words.
```

File Name	Method Name	Variable Name	Class Name
FN_1	MN_1	VN_1	CN_1
FN_2	MN_2	VN_2	CN_2
FN_3	MN_3	VN_3	CN_3
..	…	…	…
FN_N	MN_N	VN_N	CN_N

Figure 2. Keyphrase Based Source Code Dependency Graph-SC2KE

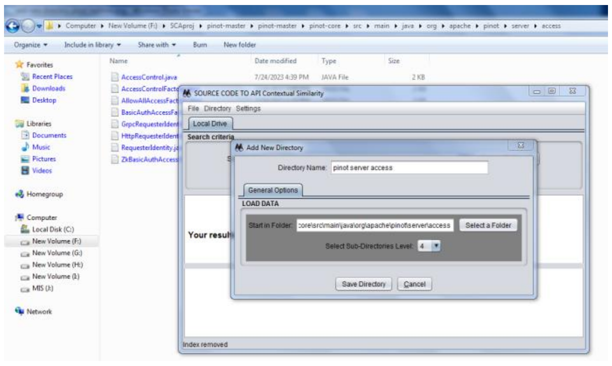
3.2. Source code and documentation similarity measure

```
INPUT: source code metrics collected from 3.1 section and document metrics
Parameters: path to source code directory
Output: contextual similarity measure
START
Step:1 For every file scan the input file
Step:2 Parse the source code to construct a dependency graph
Step : 2.1 Apply document pre-processing using NLP parser.
Step : 2.2 Construct the dependency graph consisting of
the vertices and the edges. Vertices represent the source code methods and fields while edge represents strength of connection between the connection between vertices.
Step :2.3 Edges weight between the two vertex $V_i$ & $V_j$ is calculated using equation-1
Edge weight : $w(i,j)=\frac{TF(Vi)+TF(Vj)}{TF(Vi,Vj)-TF(Vi)-TF(Vj)}$……………eq-1

Where, $TF(V_i)$ is the Term Frequency of Vertex $V_i$,
       $TF(V_j)$ is the Term Frequency of Vertex $V_j$
Edge weights are sorted with positive edge weights sorted.
Step 2.4: The graph based similarity measure is calculated using vertex nodes as follows:
Let $FV(M_i) = (m_1,m_2,….m_n)$ represent at vertex i, all the methods.
$FV(F_i) = (f_1,f_2,….f_n)$ represent at vertex i, all the variables.
$F(M_k)= (dm_1, dm_2,….dm_k)$ represent at vertex k, the API document methods vector.
$F(V_i) = (df_1, df_2,….df_k)$ denotes the API document Variable vector at vertex k.
$|FV(M_i)|=\sqrt{} FV(M_1)^2 + FV(M_2)^2 + ...+FV(M_i)^2$
$|FV(F_i)|=\sqrt{} FV(F_1)^2 + FV(F_2)^2 + ...+FV(F_l)^2$
$|(FM_k)|=\sqrt{} (FM_1)^2 + (FM_2)^2 + ...+(FM_n)^2$
$|(FV_k)|=\sqrt{} (FV_1)^2 + (FV_2)^2 + ...+(FV_z)^2$
$|FV(M_i).FM_j|=FV(M_1).FM_1+FV(M_2).FM_2+…+ FV(M_i).FM_j$
The contextual similarity based on the graph constructed is given as follows:-
$CSE=\frac{\sqrt{|FV(Mi).FMj|*|FV(Mi)|*|FMj|}}{csc\ (FV(Mi,FMj)*|FV(Mi)|*|FMj|}$
STOP
```

IV. RESULTS AND DISCUSSION

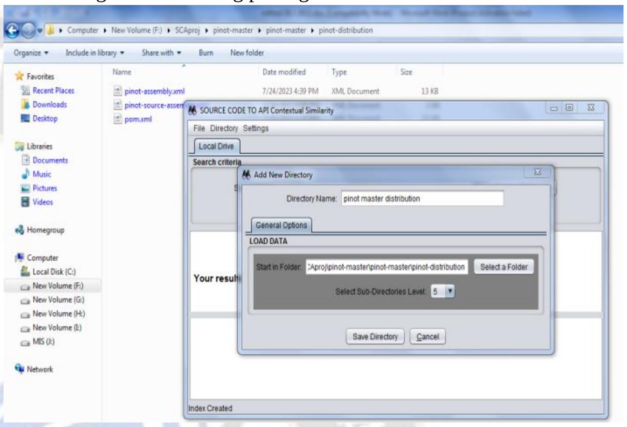
The proposed methodology was evaluated on several open source projects. Figure 4,5,6,7,8 shows different stages of inputs and output got from the proposed system. As an example, the open source pinot project was used for getting the results. Figure-4 shows how system accepts the path for the package. Figure-5 shows providing the keyword "derivtivesIndirection" for searching. Figure-7 and Figure 8 provides the result of running the algorithm SC2KE.


Computer ▸ New Volume (F:) ▸ SCAproj ▸ pinot-master ▸ pinot-master ▸ pinot-core ▸ src ▸ main ▸ java ▸ org ▸ apache ▸ pinot ▸ server ▸ access

Name	Date modified	Type	Size
AccessControl.java	7/24/2023 4:39 PM	JAVA File	2 KB
AccessControlFact			
AllowAllAccessFact			
BasicAuthAccessFa			
GrpcRequesterIdent			
HttpRequesterIdent			
RequesterIdentity.ja			
ZkBasicAuthAccess			

SOURCE CODE TO API Contextual Similarity
Add New Directory
Directory Name: pinot server access
LOAD DATA
Start in Folder: :ore\src\main\java\org\apache\pinot\server\access
Select Sub-Directories Level: 4
Index removed

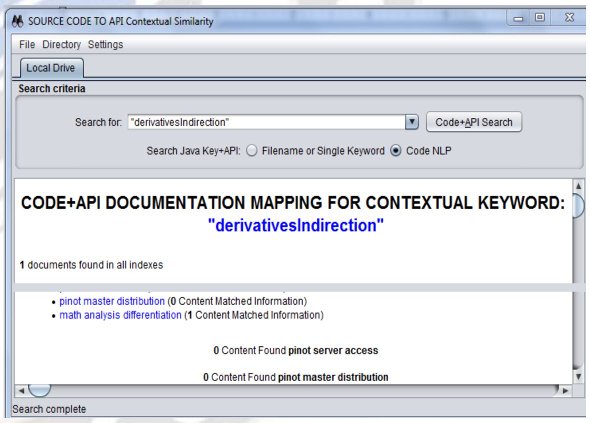
Figure 3. Adding package of source code files


Computer ▸ New Volume (F:) ▸ SCAproj ▸ pinot-master ▸ pinot-master ▸ pinot-distribution

Name	Date modified	Type	Size
pinot-assembly.xml	7/24/2023 4:39 PM	XML Document	13 KB
pinot-source-assem			
pom.xml			

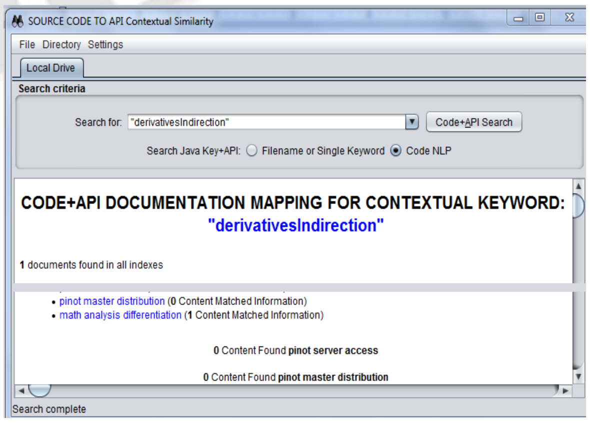
SOURCE CODE TO API Contextual Similarity
Add New Directory
Directory Name: pinot master distribution
LOAD DATA
Start in Folder: :Aproj\pinot-master\pinot-master\pinot-distribution
Select Sub-Directories Level: 5
Index Created

Figure 4. Adding package API documentation to project


SOURCE CODE TO API Contextual Similarity
Search criteria
Search for: "derivativesIndirection"
Search Java Key+API: Filename or Single Keyword [✓ Code NLP]
CODE+API DOCUMENTATION MAPPING FOR CONTEXTUAL KEYWORD: "derivativesIndirection"
1 documents found in all indexes
- pinot master distribution (0 Content Matched Information)
- math analysis differentiation (1 Content Matched Information)

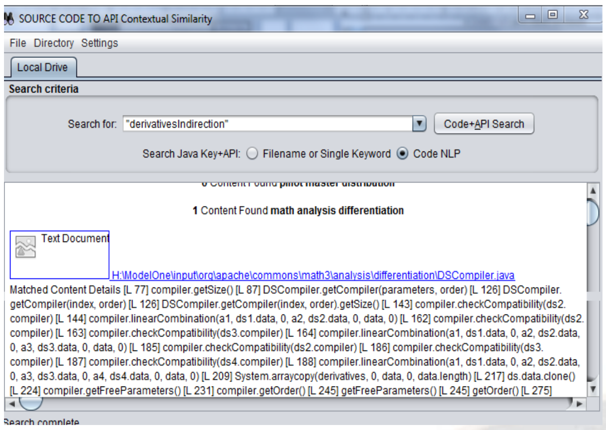
0 Content Found pinot server access
0 Content Found pinot master distribution
Search complete

Figure 5. Parsing the package for the keyword "derivativesIndirection"


SOURCE CODE TO API Contextual Similarity
Search criteria
Search for: "derivativesIndirection"
Search Java Key+API: Filename or Single Keyword [✓ Code NLP]
CODE+API DOCUMENTATION MAPPING FOR CONTEXTUAL KEYWORD: "derivativesIndirection"
1 documents found in all indexes
- pinot master distribution (0 Content Matched Information)
- math analysis differentiation (1 Content Matched Information)

0 Content Found pinot server access
0 Content Found pinot master distribution
Search complete

Figure 6. Output for searching derivatesIndirection keyword



Search for: "derivativesIndirection"

Search Java Key+API: ( ) Filename or Single Keyword [✓ Code NLP]

0 Content Found **pilot master distribution**

1 Content Found **math analysis differentiation**

H:\ModelOne\input\org\apache\commons\math3\analysis\differentiation\DSCompiler.java

```
Matched Content Details [L 77] compiler.getSize() [L 87] DSCompiler.getCompiler(parameters, order) [L 126] DSCompiler.
getCompiler(index, order) [L 126] DSCompiler.getCompiler(index, order).getSize() [L 143] compiler.checkCompatibility(ds2.
compiler) [L 144] compiler.linearCombination(a1, ds1.data, 0, a2, ds2.data, 0, data, 0) [L 162] compiler.checkCompatibility(ds2.
compiler) [L 163] compiler.checkCompatibility(ds3.compiler) [L 164] compiler.linearCombination(a1, ds1.data, 0, a2, ds2.data,
0, a3, ds3.data, 0, data, 0) [L 185] compiler.checkCompatibility(ds2.compiler) [L 186] compiler.checkCompatibility(ds3.
compiler) [L 187] compiler.checkCompatibility(ds4.compiler) [L 188] compiler.linearCombination(a1, ds1.data, 0, a2, ds2.data,
0, a3, ds3.data, 0, a4, ds4.data, 0, data, 0) [L 209] System.arraycopy(derivatives, 0, data, 0, data.length) [L 217] ds.data.clone()
[L 224] compiler.getFreeParameters() [L 231] compiler.getOrder() [L 245] getFreeParameters() [L 245] getOrder() [L 275]
```

Search complete

Figure 7. derivatesIndirection keyword found in DSComplier.java

V. CONCLUSION

This work proposes contextual similarity approach in order to find traceability links between the source code and the API documentation. The authors proposed a contextual similarity based keyphrase extraction mechanism for searching across different project source code. The proposed system is evaluated using Apache Pinot database. The similarity measure can be used to create and fire query for query-based software traceability linkages. The linkage hence created can aid in various program comprehension and SCA tasks.

REFERENCES

[1] G. Martinčić-Ipšić, Sanda, and Ana Meštrović. "Language Networks."
[2] Kirkov, Radoslav, and Gennady Agre. "Source code analysis–an overview." Cybernetics and Information Technologies 10.2 (2010): 60-77.
[3] Shi, Wei, Weiguo Zheng, Jeffrey Xu Yu, Hong Cheng, and Lei Zou. "Keyphrase extraction using knowledge graphs." Data Science and Engineering 2, no. 4 (2017): 275-288.
[4] Datta Deepanwita, Shubham Varma, and Sanjay K. Singh. "Multimodal retrieval using mutual information based textual query reformulation." Expert Systems with Applications 68 (2017): 81-92.
[5] Kumar, Niraj, Kannan Srinathan, and Vasudeva Varma. "A graph-based unsupervised N-gram filtration technique for automatic keyphrase extraction." International Journal of Data Mining, Modelling and Management 8, no. 2 (2016): 124-143.
[6] Litvak, Marina, Mark Last, and Abraham Kandel. "Degext: a language-independent keyphrase extractor." Journal of Ambient Intelligence and Humanized Computing 4 (2013): 377-387.
[7] Huang, Changqin, Jia Zhu, Yuzhi Liang, Min Yang, Gabriel Pui Cheong Fung, and Junyu Luo. "An efficient automatic multiple objectives optimization feature selection strategy for internet text classification." International Journal of Machine Learning and Cybernetics 10 (2019): 1151-1163.
[8] Ying, Yan, Tan Qingping, Xie Qinzheng, Zeng Ping, and Li Panpan. "A graph-based approach of automatic keyphrase extraction." Procedia Computer Science 107 (2017): 248-255.
[9] Danilevsky, Marina, Chi Wang, Nihit Desai, Xiang Ren, Jingyi Guo, and Jiawei Han. "Automatic construction and ranking of topical keyphrases on collections of short documents." In Proceedings of the 2014 SIAM International Conference on Data Mining, pp. 398-406. Society for Industrial and Applied Mathematics, 2014.
[10] Truică, Ciprian-Octavian, Jérôme Darmont, Alexandru Boicea, and Florin Rădulescu. "Benchmarking top-k keyword and top-k document processing with T2K2 and T2K2D2." Future Generation Computer Systems 85 (2018): 60-75.
[11] Haddoud, Mounia, and Said Abdeddaim. "Accurate keyphrase extraction by discriminating overlapping phrases." Journal of Information Science 40, no. 4 (2014): 488-500.
[12] Spirin, Egor, Egor Bogomolov, Vladimir Kovalenko, and Timofey Bryksin. "PSIMiner: A tool for mining rich abstract syntax trees from code." In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp. 13-17. IEEE, 2021.
[13] Amir Hossein Rasekh, Amir Hossein Arshia, Seyed Mostafa Fakhrahmad, Mohammad Hadi Sadreddini; Mining and discovery of hidden relationships between software source codes and related textual documents, Digital Scholarship in the Humanities, Volume 33, Issue 3, 1 September 2018, Pages 651–669, https://doi.org/10.1093/llc/fqx052.
[14] Florescu, Corina, and Cornelia Caragea. "A position-biased pagerank algorithm for keyphrase extraction." Proceedings of the AAAI conference on artificial intelligence. Vol. 31. No. 1. 2017.

Annexure-I

Algorithm for determining candidate documents for similarity calculation

```
START
Step 1: Check the extension of each document in the package.
Step 2: If the file extension is .dll,.exe, then ignore  the file
Step 3: if the file extension is .chm, .txt,.doc,.xml then include it for calculating the similarity.
STOP.
```