An Empirical Study on Source Code Feature Extraction in Preprocessing of IR-Based Requirements Traceability

Bangchao Wang[1,2], Yang Deng[1], Ruiqi Luo[1,2,*], and Huan Jin[1]
[1]School of Computer Science and Artificial Intelligence, Wuhan Textile University, Wuhan, China
[2]Engineering Research Center of Hubei Province for Clothing Information, Wuhan Textile University, Wuhan, China
wangbc@whu.edu.cn, dd0028y@163.com, rqluo@wtu.edu.cn, jh_0230@163.com
*corresponding author

***Abstract*—In information retrieval-based (IR-based) requirements traceability research, a great deal of researches have focused on establishing trace links between requirements and source code. However, as the description styles of source code and requirements are very different, how to better preprocess the code is crucial for the quality of trace link generation. This paper aims to draw empirical conclusions about code feature extraction, annotation importance assessment, and annotation redundancy removal through comprehensive experiments, which impact the quality of trace links generated by IR-based methods between requirements and source code. The results show that when the average annotaion density is higher than 0.2, feature extraction is recommended. Removing redundancy from code with high annotation redundancy can enhance the quality of trace links. The above experiences can help developers to improve the quality of trace link generation and provide them with advice on writing code.**

Keywords- requirements traceability; software reliability; software engineering；code feature extraction; empirical study

I. INTRODUCTION

Requirements Traceability (RT) is defined as '*the ability to describe and follow the life of a requirement in both a forward and backward direction* (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and periods of ongoing refinement and iteration in any of these phases)'[1]. A large number of RT techniques are used in ensuring system quality and responding to changing requirements, effectively helping developers to discover inter-product dependencies, assuring requirement coverage, and calculating the impact of requirements changes[2].

With the increased size and complexity of software systems, manually recovering and maintaining trace links is time-consuming and costly[3,4]. Therefore, the advantage of using information retrieval (IR) is that it can automatically generate trace links through text similarity. Due to the specificity of code artifacts, problems such as vocabulary mismatch and data redundancy occur when using IR techniques to establish trace between requirements and source codes, and the result of trace link generation is often unsatisfactory. In recent years, researchers have proposed different improvement strategies in three stages: preprocessing stage, links generation stage, and links refinement stage, aiming to increase the connection between code and artifacts. In the preprocessing stage, researchers have weighted the code documents by analyzing a certain identifier of the code (e.g., class, annotation, etc.). The current strategies are creating Syntax Tree[5,6], extracting Code Annotations[7], Term Classification[8]. Strategies such as Class Clustering[8], Configuration Management Log[9] and Code Ownership[10] have been proposed in the link generation stage. In the link refinement stage, Analyzing Closeness Relations[11] and Commonality and Variability Analysis[9] are mainly proposed to refine the generated trace links by the dependencies between code classes.

However, during these mentioned studies, there is little attention paid to how some important features of the code artifacts should be extracted and what impact the combination of different features has on the establishment of trace links between source code and requirements. In order to investigate the above issues, the following empirical studies have been conducted on five open-source datasets.

1) Different feature entities such as class names (CN), variables names (VN), method names (MN) and annotations (CMT) are extracted and these code features are combined with different combinations. The annotations that affect the trace link generation the most are analyzed by calculating average density annotations and ablation experiments. It is demonstrated that code feature extraction is more effective when the average annotation density is higher than 0.2. The feature combination "CN_MN_CMT" is more effective in trace links generation for the Vector Space Model (VSM), and "CN_CMT" is more effective for the Latent Semantic Indexing (LSI).

2) Annotation redundancy is analyzed. A method called CAJP (described in Section II-D) is proposed that further divides annotations. The results of the study show that the quality of IR-based trace link generation does not decrease after removing annotation redundancy. For datasets with more annotation redundancy, removing annotation redundancy improves the quality of trace link generation.

3) In response to the above findings, research suggestions are made for trace link generation between requirements and source code. These suggestions are concluded from three perspectives, which are code feature extraction, annotation importance assessment, and annotation redundancy removal, to improve the quality of trace link generation.

The rest of this paper is organized as follows: a detailed description of the research problem, dataset, experimental environment, quality assessment and the experimental process is introduced in Section II. The results of the experiments are presented and analyzed in Section III. Section IV provides

2693-9177/22/$31.00 ©2022 IEEE
DOI 10.1109/QRS57517.2022.00110

suggestions to researchers on code writing specifications, code feature extraction and annotation redundancy, and analyzes the validity threats and related work. Section V introduces the conclusion and future work.

II. Empirical Study

A. Research Questions

To investigate how to better establish trace links between requirements and source code, three research questions (RQs) are determined, as shown in Table 1.

Table 1. Research questions of this work

Research Question	Motivation
RQ1: What is the impact of code feature extraction on the trace link generation between requirements and source code?	In order to investigate the impact of extracting code features on improving the quality of IR-based trace link generation and provide experience for researchers on the feature extraction combination.
RQ2: What is the impact of annotations in the source code on the trace link generation between requirements and source code?	To further investigate the causes of the experimental results of RQ1 and help researchers to understand how annotations help establish trace links.
RQ3: What is the impact of annotation redundancy on the trace link generation between requirements and source code?	To explore whether removing annotation redundancy can improve the quality of trace link generation.

B. Datasets

In this study, five open-source datasets are chosen: iTrust, eTOUR, Albergate, EasyClinic, and SMOS. As shown in Table 2, the data are selected based on the following principles: 1) These data are obtained from the free open-source community CoEST[1], which helps other researchers to facilitate replication. 2) To enhance the experimental reliability, datasets of different sizes are selected. According to[12], when "Space" is larger than 3000, it is "large", otherwise, it is "small". Experiments of RQ1 and RQ2 use all datasets in the table, and experiments of RQ3 use iTrust and SMOS, because only the iTrust and SMOS conform to the java annotation specification, while the other datasets do not.

Table 2. Experimental datasets

Name	Source Artifacts (Count)	Target Artifacts (Count)	Space	Trace Links	Scale
ITrust	Use Cases (131)	Java Code (226)	29606	418	Large
ETOUR	Requirements (58)	Code (116)	6728	308	Large
SMOS	Use Cases (67)	Java Code (98)	6566	1027	Large
EasyClinic	Use Cases (30)	Class Description (47)	1410	93	Small
Albergate	Requirements (17)	Code (55)	935	53	Small

C. The Process of IR-based Trace Links Generation

This paper only considers fully automated requirements trace link generation without considering the refinement phase. According to the objectives and research questions, the process of trace link generation contains four steps, which are code feature extraction, requirements and source code pre-processing, model selection, and threshold selection. Figure 1 illustrates the process of IR-based trace link generation between requirements and source code. The yellow boxes indicate the method of extracting code features, a method called CAJP (described in the next section) is proposed to divide the annotations more delicately and help to remove annotation redundancy. The preprocessing and removing redundancy are the generic steps. The red boxes indicate the method of trace link generation, and the blue boxes indicate the extracted code features and corpus, which are the generic steps.

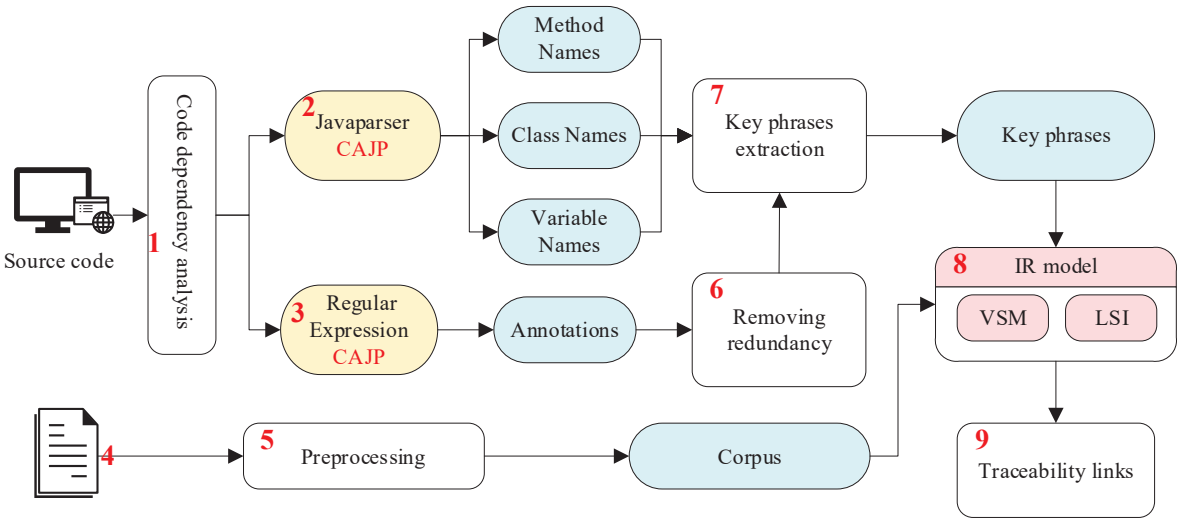

Figure 1. Process for IR-based trace links generation for requirement and source codes

[1] http://www.coest.org/

D. The Detail of Preprocessing

1) Code Feature Extraction

To preprocess the source code, the textual information of CN, MN, VN, and CMT of each source code are extracted into four files as shown in Table 4. First, apply a code dependency analysis to the code **(1)**. Then, according to the code specification, JavaParser[12], CAJP, and regular representation are used to extract the code features **(2)**. The extraction results are shown in the figure of RQ3. JavaParser is used to parse each Java file into an abstract syntax tree using the ASTParser of Eclipse JDT[12]. Code features can be extracted by querying the abstract syntax tree.

In contribution, a new method called CAJP is proposed, which can extract code features for CN, MN, VN, and CMT. The CAJP enhances block annotation extraction quality compared to JavaParser and has the ability to analyze the annotation content to remove annotation redundancy. As shown in Figure 3 of RQ3, CAJP first extracts each line (tag) in the block annotation before dividing the Tag into Context and DocletTag of value and DocletTag of name. Then, removing the redundancy of the block annotation by subsequently extracting the Context and Doclet of value, the Doclet of name is considered as the annotation redundancy **(6)**. Table 3 presents the pseudo-code for the CAJP algorithm.

Table 3. Annotation extraction algorithm

Algorithm: Annotation content extraction

Input：Tags list (T)

Output：AnnotationDecomposer Object

```
1. Function TagsDecomposer(T)
2.    C: Save a text list of Tags text content
3.    Contexts: Annotation Context storage list
4.    Doclets: Document Part Storage List
5.    While T≠Ø:
6.       Get the Tag in the current tag list
7.       If Tag_size>2:
8.          Tag_TaxtContext = ExtractContext (Tag);
            // Extracting tag context
9.          C.add (Tag_TaxtContext);
10.      End if
11.   End while
12.   While C≠Ø:
13.      Get the current Tag text content
14.      If Tag_TaxtContext.isDocletTag():
15.         // The current Tag text content is DocletTag
16.         docletTag = docletTagDecomposer (Tag_TaxtContext)
            // Creating DocletTag objects
17.         DocletTags.add(docletTag)
18.      End if
19.      Else:
20.         Contexts.add (Tag_TaxtContext)
21.      End else
22.   End while
23.Return InitAnnotationDecomposer ()
```

Lines 2-4 initialize the relevant variables, lines 5-11 divide the block annotation into line annotation (tags), lines 12-22 divide the tags into Context, DocletTag of value, and DocletTag of name, and line 23 returns the data.

If the JavaParser extraction method is used, there is no need to remove the annotation redundancy and they do not go through **(6)**. For the extraction of annotations that are not standardized and line annotations, the regular expressions is used to extract annotations **(3)**.

Table 4. Source code sections used in experimentation

Acronym	Identifier Type
CN	Class Name
MN	All Public and Private Method Names of a Class
VN	Class and Method Variable Names of a Class
CMT	All Block and Single line Annotations of a Class

2) Requirements and Source Code Pre-processing

In the pre-processing of the requirement artifacts, all sub-files and sub-files names in the artifacts are separately merged into one document **(4)**. As an illustration, each line in document A which contains the contents of the subfiles, represents a subfile and each line in document B, which contains the names of the subfiles, corresponds to the line in document A **(4)**. Then, non-alphabetic characters are removed, and NLTK[13] is used to split words and remove punctuation and stop words, label the lexicon and retain verbs and nouns[14]. To get Corpus, the stems are extracted and written to a file using the Stemming algorithm **(5)**.

The code feature files are combined in a total of fifteen combinations. After using NLTK to split the text containing annotations and remove punctuation and stop words, The text in each of the fifteen files is pre-processed. Using the Stemming algorithm, we extract the word stems from each text file to obtain the key phrase **(7)**.

E. Model Selection

According to[12], VSM and LSI are the two most used IR model, and standard VSM has the best performance in trace link generation, while LSI is an improvement to VSM that mainly addresses the problem of synonyms and polysemy. Therefore, this experiment uses the standard IR model, VSM, and LSI models respectively to generate trace links **(8)**.

VSM[13] is to represent each text as a term set $T = \{t_1, t_2 \dots, t_n\}$, and then calculate the weight w_i for each term t_i in T by the TF-IDF algorithm, so that a dataset T is viewed as an n-dimensional space vector. And the values belonging to the weights $W = \{w_1, w_2 \dots, w_n\}$ are the corresponding values for each dimension. Where w_i is calculated as follows:

$$w_i = tf_d(t_i) \times idf_i \tag{1}$$

$tf_d(t_i)$ denotes the frequency with which t_i in a document occurs in document d, and idf_i denotes the frequency with which the term t_i occurs in the whole corpus. Where $tf_d(t_i)$ is calculated as follows.

$$idf_i = \log\left(\frac{D}{df_i}\right) \tag{2}$$

where D denotes the overview of documents in the corpus and df_i denotes the number of documents containing the term t_i.

The above formula allows the textual relationship between the artifacts to be directly translated into a vector operation. Finally, the similarity between two documents is calculated by cosine similarity. The equation is as follows:

$$sim(a,b) = \frac{\sum a_i \times b_i}{\sqrt{\sum a_i^2 \times \sum b_i^2}} \quad (3)$$

LSI[15,16] is an improvement compared with VSM. Firstly, the weights in the words are calculated and the document matrix is generated by the TF-IDF. For high-dimensional matrices, dimensionality reduction is usually performed using the Singular Value Decomposition (SVD) algorithm. The similarity of the text is calculated using cosine similarity, and LSI uses corpus to analyze the semantic relevance of the documents. However, the effect of LSI is dependent on the information given in the context of the document, and when less information is given, the LSI improvement is not distinct. Therefore, it is not as effective in creating trace links for products in requirements engineering.

F. Threshold Selection

The IR model is used to calculate the similarity in reverse order by size to generate the final trace links **(9)**. For the threshold selection, the number of candidate links is selected using Selectivity Rate (SR). The equation is as follows:

$$SR = \frac{A}{B} \quad (4)$$

A represents the number of candidate trace links and B represents the total number of trace links. As the number of candidate links increases, the variation of its precision and recall becomes smaller[17]. Therefore, when SR $\in$ [0,0.2], 0.01 is used as the interval point, when SR $\in$ [0.2,1], 0.05 is used as the interval point, which is a total of 36 points.

G. Quality Measures

In the field of IR, there are many different metrics that can be used to assess the quality of IR models. Among them, the most commonly used are $precision$ (P) and $recall$ (R). When the R is 100%, it means that all recovered trace links have been found; when the P is 100%, it means that all recovered trace links are correct. The equation for these two metrics is as follows:

$$Precision = \frac{|N_{total} \cap N_{rel}|}{|N_{rel}|} \quad (5)$$

$$Recal = \frac{|N_{total} \cap N_{rel}|}{|N_{total}|} \quad (6)$$

Where N_{total} represents the total number of candidate trace links. N_{rel} represents the total number of trace links in the true set. The intersection of both indicates which candidate trace links are in the true set.

When P and R are equally effective, there is a need for another metric to measure. Therefore, a third metric is used: $F - measure$ (F). This metric is the summed average of P and R. The equation is as follows:

$$F - measure = \frac{1}{\frac{1}{P(recall)} + \frac{1}{P(precision)}} \quad (7)$$

In order to systematically measure automated RT techniques, the criteria are shown in Table 5. The metric is a R and P measure proposed by Hayes et al.[17], which is based on industrial practice. The criterion measures the results obtained by the IR model without any refinement strategy and considers R to be more important than P. In addition, P and R cannot be separated and need to meet the criteria at the same time.

Table 5. Standards from Hayes[17]

Measure	Acceptable	Good	Excellent
Recall	60% — 69%	70% — 79%	80% — 100%
Precision	20% — 29%	30% — 49%	50% — 100%

III. RESULTS AND ANALYSIS

A. RQ1: What is the impact of code feature extraction on the trace link generation between requirements and source code?

For RQ1, JavaParser is used to extract VN, CN, and MN in the code, and regular expressions to extract line annotations and block annotations in the code, and then combine them into 15 kinds of patterns of feature combinations to build trace links through IR model. Instead of code slicing, code files are regarded as text files and then establish trace links through IR model. This experiment considers R as more important by Hayes et al.[17]. The intercept points close to "Acceptable", "Good", and "Excellent" are taken separately for the R, and then the P and F are compared.

Table 6 represents the results of the code feature extraction experiment for VSM, while Table 7 represents the results of the experiment for LSI. The first row of "VSM" and "LSI" in the table indicates that no feature extraction is done. From the experimental results of both models, it can be seen that the results of generating trace links for IR model are generally better for extracting code features than for not extracting codes. Among the four datasets TOUR, iTrust, SMOS, and EasyClinic, the feature combination "CN_MN_CMT" performs better in the VSM model. In LSI, the feature combination with "CN_CMT" works better. In addition, the combination of the features with CMT generally better. Therefore, the extraction of features has reduced the noise of the code information and extracted the useful information more precisely. In particular, the comments provide a more standardized introduction to the code and their extraction helps to establish the links between requirements and source code.

For the Albergate, the effect of not extracting the code features is better than extracting the code features. Upon

analyzing the datasets, the annotations in Albergate are sparse, which plays a significant role in the poor trace link generation between requirements and source code.

Above all, for most datasets, code feature extraction can improve the effectiveness in trace link generation. The feature combination with "CN_MN_CMT" is better in the VSM model. In LSI, the feature combination with "CN_CMT" performs better.

Table 6. Code feature extraction experiment results for VSM

	Albergate			eTOUR			iTrust			SMOS			EasyClinic		
	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
VSM	***0.6226***	***0.1416***	***0.2308***	0.6006	0.1100	0.1859	0.6077	0.1073	0.1823	0.6339	0.1983	0.3021	0.6559	0.6224	0.6387
	0.7170	***0.1357***	***0.2282***	0.7435	0.0851	0.1527	0.7177	0.0780	0.1406	0.7108	0.1853	0.2940	0.7097	0.5238	0.6027
	0.8113	***0.1024***	***0.1818***	0.8214	0.0752	0.1378	0.8254	0.0466	0.0882	0.8043	0.1677	0.2776	0.8172	0.3858	0.5241
CN	0.6226	0.0642	0.1164	0.6071	0.0463	0.0861	0.6100	0.0157	0.0305	0.6290	0.1789	0.2786	0.6129	0.0899	0.1568
	0.7359	0.0643	0.1182	0.7143	0.0467	0.0877	0.7033	0.0153	0.0299	0.8130	0.1696	0.2806	0.7419	0.0816	0.1470
	0.8679	0.0656	0.1220	0.8084	0.0463	0.0875	0.8110	0.0143	0.0281	0.7352	0.1769	0.2852	0.8172	0.0771	0.1409
MN	0.6604	0.0750	0.1346	0.6007	0.0458	0.0852	0.6196	0.0159	0.0310	0.6232	0.1772	0.2760	0.6344	0.0697	0.1257
	0.7359	0.0759	0.1376	0.7046	0.0461	0.0865	0.7129	0.0155	0.0303	0.7322	0.1762	0.2841	0.7312	0.0690	0.1260
	0.8302	0.0673	0.1245	0.8182	0.0468	0.0886	0.8206	0.0145	0.0285	0.8121	0.1694	0.2803	0.8172	0.0674	0.1245
VN	0.6038	0.0775	0.1373	0.6234	0.0571	0.1046	0.6603	0.0155	0.0304	0.6222	0.1770	0.2755	0.6129	0.0674	0.1214
	0.7170	0.0691	0.1260	0.7013	0.0494	0.0923	0.7034	0.0153	0.0299	0.7274	0.1751	0.2822	0.7527	0.0662	0.1217
	0.8491	0.0577	0.1080	0.8409	0.0481	0.0910	0.8230	0.0145	0.0285	0.8062	0.1682	0.2783	0.8172	0.0674	0.1245
CMT	0.6038	0.0570	0.1042	0.6136	0.1124	0.1900	0.6053	0.1068	0.1816	0.6378	0.1995	0.3039	0.6344	0.6020	0.6178
	0.7925	0.0642	0.1188	0.7403	0.0847	0.1521	0.7034	0.0764	0.1378	0.7050	0.1838	0.2916	0.7097	0.5238	0.6027
	0.8302	0.0628	0.1167	0.8279	0.0689	0.1272	0.8206	0.0386	0.0738	0.8092	0.1688	0.2793	0.8280	0.3909	0.5310
CN_MN	0.6226	0.0707	0.1269	0.6039	0.0461	0.0856	0.6172	0.0158	0.0309	0.6280	0.1786	0.2781	0.6344	0.0837	0.1479
	0.7359	0.0695	0.1270	0.7046	0.0461	0.0865	0.7129	0.0155	0.0303	0.7361	0.1772	0.2856	0.7312	0.0804	0.1448
	0.8302	0.0673	0.1245	0.8214	0.0470	0.0889	0.8206	0.0145	0.0285	0.8121	0.1694	0.2803	0.8172	0.0771	0.1409
CN_VN	0.6038	0.0775	0.1373	0.6266	0.0574	0.1051	0.6603	0.0155	0.0304	0.6329	0.1800	0.2803	0.6344	0.1046	0.1796
	0.7170	0.0691	0.1260	0.7078	0.0499	0.0931	0.7057	0.0153	0.0300	0.7381	0.1776	0.2864	0.7204	0.0792	0.1427
	0.8113	0.0625	0.1161	0.8442	0.0483	0.0914	0.8230	0.0145	0.0285	0.8150	0.1700	0.2813	0.8172	0.0771	0.1409
CN_CMT	0.6038	0.0623	0.1129	0.6136	0.1124	0.1900	0.6172	0.1090	0.1852	0.6349	0.1986	0.3026	***0.6452***	***0.6122***	***0.6283***
	0.7547	0.0612	0.1132	0.7305	0.0836	0.1500	0.7034	0.0828	0.1481	0.7040	0.1835	0.2912	***0.7097***	***0.5893***	***0.6439***
	0.8679	0.0656	0.1220	0.8182	0.0681	0.1258	0.8206	0.0386	0.0738	0.8092	0.1688	0.2793	***0.8495***	***0.4010***	***0.5448***
MN_VN	0.6415	0.0728	0.1308	0.6266	0.0574	0.1051	0.6699	0.0158	0.0308	0.6271	0.1783	0.2777	0.6022	0.0662	0.1193
	0.7736	0.0731	0.1335	0.7078	0.0499	0.0931	0.7034	0.0828	0.1481	0.7352	0.1769	0.2852	0.7204	0.0680	0.1242
	0.8302	0.0673	0.1245	0.8409	0.0481	0.0910	0.8206	0.0386	0.0738	0.8111	0.1692	0.2800	0.8065	0.0665	0.1229
MN_CMT	0.6226	0.0642	0.1164	0.6169	0.1130	0.1910	0.6148	0.1240	0.2064	***0.6027***	***0.2095***	***0.3110***	0.6237	0.5918	0.6073
	0.7547	0.0612	0.1132	0.7013	0.0918	0.1623	0.7034	0.0903	0.1600	***0.7147***	***0.1863***	***0.2956***	0.7419	0.4894	0.5897
	0.8679	0.0656	0.1220	0.8052	0.0737	0.1351	0.8278	0.0390	0.0744	***0.8072***	***0.1684***	***0.2786***	0.8280	0.3909	0.5310
VN_CMT	0.6981	0.0660	0.1205	***0.6169***	***0.1130***	***0.1910***	0.6172	0.1090	0.1852	0.6368	0.1992	0.3035	0.6129	0.5816	0.5969
	0.7547	0.0659	0.1212	***0.7403***	***0.0847***	***0.1521***	0.7105	0.0717	0.1302	0.7079	0.1846	0.2928	0.7312	0.4387	0.5484
	0.8491	0.0642	0.1194	***0.8117***	***0.0743***	***0.1362***	0.8182	0.0385	0.0736	0.8092	0.1688	0.2793	0.8065	0.3555	0.4934
CN_MN_VN	0.6415	0.0728	0.1308	0.6266	0.0574	0.1051	0.6675	0.0157	0.0307	0.6329	0.1800	0.2803	0.6129	0.1011	0.1735
	0.7736	0.0731	0.1335	0.7078	0.0499	0.0931	0.7105	0.0154	0.0302	0.7390	0.1779	0.2867	0.7097	0.0780	0.1406
	0.8113	0.0658	0.1216	0.8377	0.0479	0.0907	0.8325	0.0147	0.0289	0.8150	0.1700	0.2813	0.8172	0.0771	0.1409
CN_MN_CMT	0.5849	0.0603	0.1093	0.6104	0.1118	0.1889	***0.6412***	***0.1293***	***0.2153***	0.6329	0.1980	0.3016	0.6344	0.6020	0.6178
	0.7170	0.0677	0.1238	0.7305	0.0836	0.1500	***0.7057***	***0.0906***	***0.1606***	0.7108	0.1853	0.2940	0.7419	0.5476	0.6301
	0.8113	0.0613	0.1141	0.8020	0.0734	0.1345	***0.8038***	***0.0454***	***0.0859***	0.8043	0.1677	0.2776	0.8280	0.4208	0.5580
CN_VN_CMT	0.6415	0.0675	0.1221	0.6104	0.1118	0.1889	0.6005	0.1211	0.2016	0.6280	0.1965	0.2993	0.6452	0.6122	0.6283
	0.7547	0.0623	0.1151	0.7305	0.0836	0.1500	0.7081	0.0769	0.1388	0.7089	0.1848	0.2932	0.7097	0.5238	0.6027
	0.8113	0.0586	0.1093	0.8020	0.0734	0.1345	0.8182	0.0385	0.0736	0.8043	0.1677	0.2776	0.8065	0.4098	0.5435
MN_VN_CMT	0.6415	0.0662	0.1199	0.6071	0.1112	0.1879	0.6148	0.1240	0.2064	0.6018	0.2092	0.3105	0.6129	0.5816	0.5969
	0.7736	0.0676	0.1243	0.7305	0.0836	0.1500	0.7081	0.0769	0.1388	0.7137	0.1861	0.2952	0.7204	0.4752	0.5726
	0.8491	0.0642	0.1194	0.8020	0.0734	0.1345	0.8254	0.0389	0.0742	0.8072	0.1684	0.2786	0.8172	0.3602	0.5000
CN_MN_VN_CMT	0.6038	0.0685	0.1231	0.6071	0.1112	0.1879	0.6172	0.1245	0.2072	0.6339	0.1983	0.3021	0.6344	0.6020	0.6178
	0.7547	0.0659	0.1212	0.7208	0.0825	0.1481	0.7105	0.0772	0.1392	0.7108	0.1853	0.2940	0.7204	0.5317	0.6119
	0.8302	0.0628	0.1167	0.8312	0.0692	0.1277	0.8230	0.0387	0.0740	0.8043	0.1677	0.2776	0.8172	0.4153	0.5507

Table 7. Code feature extraction experiment results for LSI

	Albergate			eTOUR			iTrust			SMOS			EasyClinic		
	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
LSI	***0.6415***	***0.1459***	***0.2378***	0.6234	0.1141	0.1930	0.6077	0.1073	0.1823	0.6047	0.2102	0.3120	0.6237	0.5918	0.6073
	0.7547	***0.1223***	***0.2105***	0.7078	0.0926	0.1638	0.7177	0.0780	0.1406	0.7215	0.1881	0.2984	0.7204	0.5317	0.6119
	0.8302	***0.0942***	***0.1692***	0.8084	0.0740	0.1356	0.8254	0.0466	0.0882	0.8228	0.1716	0.2840	0.8065	0.4438	0.5725
CN	0.6415	0.0662	0.1199	0.6071	0.0463	0.0861	0.6124	0.1081	0.1838	0.6008	0.1709	0.2661	0.6774	0.0813	0.1452
	0.7170	0.0677	0.1238	0.7013	0.0459	0.0861	0.7081	0.0714	0.1298	0.7098	0.1586	0.2593	0.7097	0.0721	0.1308
	0.8868	0.0592	0.1110	0.8734	0.0470	0.0893	0.8206	0.0386	0.0738	0.8033	0.1571	0.2628	0.8710	0.0676	0.1255
MN	0.6604	0.0750	0.1346	0.6104	0.0466	0.0866	0.6292	0.0162	0.0315	0.6368	0.1811	0.2820	0.6129	0.0735	0.1313
	0.7359	0.0759	0.1376	0.7240	0.0474	0.0889	0.7249	0.0158	0.0308	0.7322	0.1762	0.2841	0.7204	0.0731	0.1328
	0.8868	0.0671	0.1247	0.8312	0.0476	0.0900	0.8206	0.0145	0.0285	0.8199	0.1710	0.2830	0.8065	0.0710	0.1304
VN	0.6415	0.0741	0.1328	0.6136	0.0511	0.0943	0.6890	0.0122	0.0239	0.6241	0.1627	0.2582	0.6022	0.0662	0.1193
	0.7170	0.0691	0.1260	0.7468	0.0488	0.0917	0.6172	0.0125	0.0244	0.7235	0.1617	0.2643	0.7312	0.0603	0.1114
	0.8491	0.0577	0.1080	0.8377	0.0479	0.0907	0.8062	0.0134	0.0263	0.8062	0.1577	0.2637	0.8065	0.0626	0.1162
CMT	0.6038	0.0570	0.1042	0.6169	0.1130	0.1910	0.6124	0.1081	0.1838	0.6660	0.1894	0.2950	0.6344	0.6020	0.6178
	0.7925	0.0642	0.1188	0.7013	0.0918	0.1623	0.7081	0.0714	0.1298	0.7020	0.1830	0.2904	0.7097	0.5238	0.6027
	0.8302	0.0628	0.1167	0.8247	0.0687	0.1267	0.8206	0.0386	0.0738	0.8179	0.1706	0.2823	0.8065	0.4098	0.5435
CN_MN	0.6226	0.0707	0.1269	0.6169	0.0471	0.0875	0.6005	0.0170	0.0330	0.6388	0.1817	0.2829	0.6022	0.0794	0.1404
	0.7359	0.0695	0.1270	0.7435	0.0486	0.0913	0.7345	0.0148	0.0290	0.7313	0.1760	0.2837	0.6989	0.0710	0.1288
	0.8302	0.0673	0.1245	0.8474	0.0485	0.0917	0.8325	0.0138	0.0272	0.8257	0.1722	0.2850	0.8065	0.0761	0.1390
CN_VN	0.6038	0.0775	0.1373	0.6169	0.0514	0.0948	0.6340	0.0119	0.0234	0.6368	0.1660	0.2634	0.6667	0.0879	0.1554
	0.7170	0.0691	0.1260	0.7727	0.0505	0.0949	0.7345	0.0122	0.0240	0.7390	0.1651	0.2700	0.7527	0.0827	0.1491
	0.8113	0.0625	0.1161	0.8182	0.0499	0.0941	0.9043	0.0134	0.0265	0.8296	0.1622	0.2714	0.8280	0.0781	0.1427
CN_CMT	0.6038	0.0623	0.1129	0.6201	0.1136	0.1920	0.6268	0.1106	0.1881	0.6329	0.1980	0.3016	***0.6452***	***0.6122***	***0.6283***
	0.7547	0.0612	0.1132	0.7013	0.0918	0.1623	0.7081	0.0769	0.1388	0.7030	0.1833	0.2908	***0.7097***	***0.5893***	***0.6439***
	0.8679	0.0656	0.1220	0.8247	0.0687	0.1267	0.8014	0.0453	0.0857	0.8160	0.1702	0.2816	***0.8065***	***0.4839***	***0.6048***
MN_VN	0.6415	0.0662	0.1199	0.6266	0.0522	0.0963	0.6148	0.0145	0.0283	0.6680	0.1608	0.2592	0.6129	0.0735	0.1313
	0.7547	0.0612	0.1132	0.7208	0.0508	0.0949	0.7416	0.0131	0.0257	0.7137	0.1595	0.2607	0.7097	0.0624	0.1148
	0.8491	0.0602	0.1124	0.8117	0.0495	0.0934	0.8158	0.0136	0.0267	0.8354	0.1634	0.2733	0.8280	0.0643	0.1193
MN_CMT	0.6038	0.0570	0.1042	***0.6201***	***0.1136***	***0.1920***	***0.6531***	***0.1153***	***0.1960***	***0.6056***	***0.2106***	***0.3125***	0.6129	0.5816	0.5969
	0.7359	0.0556	0.1034	***0.7078***	***0.0926***	***0.1638***	***0.7225***	***0.0785***	***0.1416***	***0.7108***	***0.1853***	***0.2940***	0.7097	0.5238	0.6027
	0.8679	0.0579	0.1086	***0.8344***	***0.0695***	***0.1282***	***0.8086***	***0.0457***	***0.0865***	***0.8160***	***0.1702***	***0.2816***	0.8065	0.4098	0.5435
VN_CMT	0.6793	0.0642	0.1173	0.6169	0.1130	0.1910	0.6148	0.1085	0.1845	0.6047	0.2102	0.3120	0.6129	0.5816	0.5969
	0.7736	0.0676	0.1243	0.7013	0.0918	0.1623	0.7010	0.0761	0.1374	0.7020	0.1830	0.2904	0.7312	0.4823	0.5812
	0.8679	0.0615	0.1149	0.8344	0.0695	0.1282	0.8182	0.0385	0.0736	0.8130	0.1696	0.2806	0.8065	0.4098	0.5435
CN_MN_VN	0.6793	0.0642	0.1173	0.6136	0.0511	0.0943	0.6029	0.0131	0.0256	0.6465	0.1556	0.2509	0.5699	0.0752	0.1328
	0.7547	0.0659	0.1212	0.7046	0.0496	0.0927	0.7177	0.0135	0.0265	0.7254	0.1513	0.2504	0.7097	0.0780	0.1406
	0.8302	0.0588	0.1099	0.8344	0.0509	0.0960	0.8254	0.0137	0.0270	0.8559	0.1575	0.2660	0.8065	0.0665	0.1229
CN_MN_CMT	0.5849	0.0603	0.1093	0.6201	0.1136	0.1920	0.6201	0.1136	0.1920	0.6290	0.1968	0.2998	0.6237	0.5918	0.6073
	0.7170	0.0677	0.1238	0.7013	0.0918	0.1623	0.7013	0.0918	0.1623	0.7059	0.1841	0.2920	0.7419	0.5476	0.6301
	0.8113	0.0613	0.1141	0.8344	0.0695	0.1282	0.8344	0.0695	0.1282	0.8169	0.1704	0.2820	0.8065	0.4438	0.5725
CN_VN_CMT	0.6415	0.0675	0.1221	0.6169	0.1130	0.1910	0.6244	0.1102	0.1874	0.6222	0.1946	0.2965	0.6452	0.6122	0.6283
	0.7547	0.0623	0.1151	0.7013	0.0918	0.1623	0.7034	0.0764	0.1378	0.7040	0.1835	0.2912	0.7312	0.5397	0.6210
	0.8113	0.0586	0.1093	0.8312	0.0692	0.1277	0.8038	0.0454	0.0859	0.8140	0.1698	0.2810	0.8065	0.4098	0.5435
MN_VN_CMT	0.6415	0.0662	0.1199	0.6136	0.1124	0.1900	0.6483	0.1144	0.1945	0.6339	0.1983	0.3021	0.6022	0.5714	0.5864
	0.7736	0.0676	0.1243	0.7110	0.0930	0.1645	0.7034	0.0764	0.1378	0.7098	0.1851	0.2936	0.7312	0.4823	0.5812
	0.8491	0.0642	0.1194	0.8377	0.0697	0.1287	0.8062	0.0455	0.0862	0.8189	0.1708	0.2826	0.8065	0.3807	0.5172
CN_MN_VN_CMT	0.6038	0.0685	0.1231	0.6104	0.1118	0.1889	0.7153	0.0777	0.1402	0.6300	0.1971	0.3002	0.6344	0.6020	0.6178
	0.7547	0.0659	0.1212	0.7013	0.0918	0.1623	0.8062	0.0455	0.0862	0.7079	0.1846	0.2928	0.7204	0.5317	0.6119
	0.8302	0.0628	0.1167	0.8344	0.0695	0.1282	0.6005	0.1211	0.2016	0.8111	0.1692	0.2800	0.8065	0.3807	0.5172

B. RQ2: What is the impact of annotations in the source code on the trace link generation between requirements and source code?

As a result of the Albergate in RQ1, which found that the annotation is sparse, RQ2 aims to further verify whether annotation density is an important factor in code feature extraction. RQ2 verified the effect of annotation by calculating the average annotation density and ablation experiments. In the ablation experiments, their experimental measures are selected by Hayes et al. [17], which considers the R more important. The intercept points close to "Acceptable", "Good", and "Excellent" for R are taken separately and then compared for P and F.

The density of annotation lines is calculated by the following formula:

$$Density\ of\ Annotation\ Lines = \frac{A}{A+B} \tag{8}$$

where A denotes the number of annotations and B denotes the number of codes. The average annotation density is the average of the annotation density for each document.

Table 8. Average annotation density values

	Albergate	eTOUR	iTrust	SMOS	EasyClinic
Average Annotation Density	0.0155	0.7057	0.2283	0.2123	0.6319

As shown in Table 8 and Figure 2, P(after), R(after), and F(after) represents represent the P, R, and F of trace link

generation after code removal of annotations respectively. Albergate has the lowest average annotation density, and its ablation experiments have a low impact on the experimental results. For SMOS, the annotation density is close to 0.2, so the annotation has a small impact on trace link generation for VSM, and the *P* and *F* decreased when the *R* reached the point of 0.8. For iTrust, eTOUR, and EasyClinic, the quality of the generated trace links is reduced after removing the annotations. When extracting the annotations, it is found that eTOUR has a high proportion of annotations, but most of them are annotations on useless code and contained less information to trace link generation. In EasyClinic, the comments contain descriptions of the code, which can better help to build links between the requirements and the source code.

Above all, when the average annotation density is greater than 0.2, the quality of annotations will have an impact on the established trace links. The better the quality of its annotations, the better the trace link generation using the method of code feature extraction.

Figure 2. The experiment results after removing annotation for VSM and LSI

C. RQ3: What is the impact of annotation redundancy on the trace link generation between requirements and source code?

For RQ3, an example of code feature extraction is shown in Figure 3, where the extraction of block annotations. The annotations is extracted directly by using JavaParser and regular expressions. The CAJP method is used to further divide the block annotations, and the redundancy of block annotations is removed by subsequent extraction of Context and Doclet of value. The Doclet of name is considered annotation redundancy.

Because the annotation specifications of eTOUR, EasyClinic, and Albergate do not conform to the java writing specification, RQ3 extracts the annotations for iTrust and SMOS, which conform to the java writing specification.

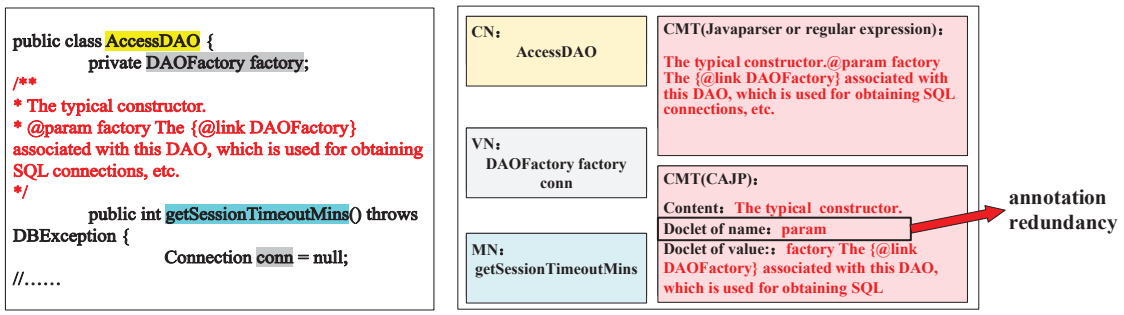

Figure 3. An example for code features, the left is the source code, the right is the result after feature extraction

Table 9. Comparison of experimental results for VSM without annotation redundancy and with annotation redundancy removal

	iTrust			iTrust (new)			SMOS			SMOS (new)		
	R	P	F	R	P	F	R	P	F	R	P	F
VSM	0.6077	0.1073	0.1823				0.6339	0.1983	0.3021			
	0.7177	0.0780	0.1406				0.7108	0.1853	0.2940			
	0.8254	0.0466	0.0882				0.8043	0.1677	0.2776			
CMT	0.6053	0.1068	0.1816	0.6029	0.1216	0.2024	0.6378	0.1995	0.3039	0.6271	0.1962	0.2988
	0.7034	0.0764	0.1378	0.7105	0.0772	0.1392	0.7050	0.1838	0.2916	0.7468	0.1798	0.2898
	0.8206	0.0386	0.0738	0.8158	0.0384	0.0733	0.8092	0.1688	0.2793	0.8286	0.1620	0.2711
CN_CMT	0.6172	0.1090	0.1852	0.6053	0.1221	0.2032	0.6349	0.1986	0.3026	0.6105	0.1910	0.2910
	0.7034	0.0828	0.1481	0.7010	0.0900	0.1595	0.7040	0.1835	0.2912	0.7507	0.1807	0.2913
	0.8206	0.0386	0.0738	0.8158	0.0384	0.0733	0.8092	0.1688	0.2793	0.8335	0.1630	0.2727
MN_CMT	0.6148	0.1240	0.2064	0.6005	0.1413	0.2288	***0.6027***	***0.2095***	***0.3110***	***0.6280***	***0.1965***	***0.2993***
	0.7034	0.0903	0.1600	0.7057	0.0997	0.1747	***0.7147***	***0.1863***	***0.2956***	***0.7488***	***0.1802***	***0.2905***
	0.8278	0.0390	0.0744	0.8014	0.0453	0.0857	***0.8072***	***0.1684***	***0.2786***	***0.8354***	***0.1634***	***0.2733***
VN_CMT	0.6172	0.1090	0.1852	0.6053	0.1221	0.2032	0.6368	0.1992	0.3035	0.6261	0.1959	0.2984
	0.7105	0.0717	0.1302	0.7081	0.0769	0.1388	0.7079	0.1846	0.2928	0.7488	0.1802	0.2905
	0.8182	0.0385	0.0736	0.8158	0.0384	0.0733	0.8092	0.1688	0.2793	0.8286	0.1620	0.2711
CN_MN_CMT	***0.6412***	***0.1293***	***0.2153***	***0.6124***	***0.1441***	***0.2334***	0.6329	0.1980	0.3016	0.6319	0.1865	0.2881
	0.7057	***0.0906***	***0.1606***	***0.7081***	***0.1000***	***0.1753***	0.7108	0.1853	0.2940	0.7537	0.1814	0.2924
	0.8038	***0.0454***	***0.0859***	***0.8014***	***0.0453***	***0.0857***	0.8043	0.1677	0.2776	0.8335	0.1630	0.2727
CN_VN_CMT	0.6005	0.1211	0.2016	0.6172	0.1245	0.2072	0.6280	0.1965	0.2993	0.6105	0.1910	0.2910
	0.7081	0.0769	0.1388	0.7010	0.0825	0.1476	0.7089	0.1848	0.2932	0.7527	0.1812	0.2920
	0.8182	0.0385	0.0736	0.8110	0.0382	0.0729	0.8043	0.1677	0.2776	0.8335	0.1630	0.2727
MN_VN_CMT	0.6148	0.1240	0.2064	0.6411	0.1293	0.2153	0.6018	0.2092	0.3105	0.6261	0.1959	0.2984
	0.7081	0.0769	0.1388	0.7033	0.0903	0.1600	0.7137	0.1861	0.2952	0.7478	0.1800	0.2901
	0.8254	0.0389	0.0742	0.8182	0.0385	0.0736	0.8072	0.1684	0.2786	0.8335	0.1630	0.2727
CN_MN_VN_CMT	0.6172	0.1245	0.2072	0.6100	0.1436	0.2325	0.6339	0.1983	0.3021	0.6018	0.1961	0.2958
	0.7105	0.0772	0.1392	0.7057	0.0906	0.1606	0.7108	0.1853	0.2940	0.7537	0.1814	0.2924
	0.8230	0.0387	0.0740	0.8158	0.0384	0.0733	0.8043	0.1677	0.2776	0.8335	0.1630	0.2727

Table 10. Comparison of experimental results for LSI without annotation redundancy and with annotation redundancy removal

	iTrust			iTrust (new)			SMOS			SMOS (new)		
	R	P	F	R	P	F	R	P	F	R	P	F
LSI	0.6077	0.1073	0.1823				0.6047	0.2102	0.3120			
	0.7177	0.0780	0.1406				0.7215	0.1881	0.2984			
	0.8254	0.0466	0.0882				0.8228	0.1716	0.2840			
CMT	0.6124	0.1081	0.1838	0.6244	0.1102	0.1874	0.6660	0.1894	0.2950	0.6241	0.1952	0.2974
	0.7081	0.0714	0.1298	0.7081	0.0714	0.1298	0.7020	0.1830	0.2904	0.7468	0.1798	0.2898
	0.8206	0.0386	0.0738	0.8062	0.0455	0.0862	0.8179	0.1706	0.2823	0.8393	0.1751	0.2897
CN_CMT	0.6268	0.1106	0.1881	0.6029	0.1216	0.2024	0.6329	0.1980	0.3016	0.6076	0.1901	0.2896
	0.7081	0.0769	0.1388	0.7033	0.0764	0.1378	0.7030	0.1833	0.2908	0.7488	0.1802	0.2905
	0.8014	0.0453	0.0857	0.8110	0.0458	0.0867	0.8160	0.1702	0.2816	0.8004	0.1789	0.2924
MN_CMT	0.6531	0.1153	0.1960	0.6148	0.1240	0.2064	***0.6056***	***0.2106***	***0.3125***	***0.6261***	***0.1959***	***0.2984***
	0.7225	0.0785	0.1416	0.7081	0.0833	0.1491	***0.7108***	***0.1853***	***0.2940***	***0.7507***	***0.1807***	***0.2913***
	0.8086	0.0457	0.0865	0.8134	0.0459	0.0870	***0.8160***	***0.1702***	***0.2816***	***0.8442***	***0.1761***	***0.2914***
VN_CMT	0.6148	0.1085	0.1845	0.6196	0.1094	0.1859	0.6047	0.2102	0.3120	0.6232	0.1949	0.2970
	0.7010	0.0761	0.1374	0.7033	0.0764	0.1378	0.7020	0.1830	0.2904	0.7478	0.1800	0.2901
	0.8182	0.0385	0.0736	0.8062	0.0455	0.0862	0.8130	0.1696	0.2806	0.8130	0.1696	0.2806
CN_MN_CMT	***0.6201***	***0.1136***	***0.1920***	***0.6364***	***0.1284***	***0.2137***	0.6290	0.1968	0.2998	0.6164	0.1928	0.2937
	0.7013	***0.0918***	***0.1623***	***0.7057***	***0.0906***	***0.1606***	0.7059	0.1841	0.2920	0.7498	0.1805	0.2909
	0.8344	***0.0695***	***0.1282***	***0.8182***	***0.0462***	***0.0875***	0.8169	0.1704	0.2820	0.8247	0.1720	0.2847
CN_VN_CMT	0.6244	0.1102	0.1874	0.6053	0.1221	0.2032	0.6222	0.1946	0.2965	0.6086	0.1904	0.2900
	0.7034	0.0764	0.1378	0.7057	0.0767	0.1383	0.7040	0.1835	0.2912	0.7498	0.1805	0.2909
	0.8038	0.0454	0.0859	0.8062	0.0455	0.0862	0.8140	0.1698	0.2810	0.8345	0.1740	0.2880
MN_VN_CMT	0.6483	0.1144	0.1945	0.6029	0.1216	0.2024	0.6339	0.1983	0.3021	0.6232	0.1949	0.2970
	0.7034	0.0764	0.1378	0.7057	0.0831	0.1486	0.7098	0.1851	0.2936	0.7498	0.1805	0.2909
	0.8062	0.0455	0.0862	0.8134	0.0459	0.0870	0.8189	0.1708	0.2826	0.8325	0.1736	0.2873
CN_MN_VN_CMT	0.7153	0.0777	0.1402	0.6459	0.1303	0.2169	0.6300	0.1971	0.3002	0.6144	0.1922	0.2928
	0.8062	0.0455	0.0862	0.7105	0.0836	0.1496	0.7079	0.1846	0.2928	0.7507	0.1807	0.2913
	0.6005	0.1211	0.2016	0.8110	0.0458	0.0867	0.8111	0.1692	0.2800	0.8179	0.1706	0.2823

As shown in Table 9 and Table 10, the first row of the table, "VSM" and "LSI", indicates the datasets without code feature extraction. The "dataset name (new)" represents the result of removing annotation redundancy.

In the iTrust, the combination of code features with annotation redundancy removed results for VSM and LSI better than without annotation redundancy. The combination with "CN_MN_CMT" performs best. This experiment considers R to be more important. Comparing P and F with approximately the R. R and P are negatively correlated and P is lost when R is increased. When the gap between P and R increases, it leads to a decrease in F. Therefore, in the SMOS, the combination with "CN_MN" performs best, and because the R for annotation redundancy removal is higher than the results without annotation redundancy removal and code feature exaction, which resulted in a loss of P and F, the experiments concluded that there is no degradation in the quality of trace link generation. In addition, analysis of the code annotation shows that there is more annotation redundancy in iTrust than in SMOS.

The results show that there is no reduction in the effectiveness of the trace link generation for the IR model after removing the annotation redundancy. For the dataset with more annotation redundancy, removing the annotation redundancy can significantly improve the performance of the IR-based traceability recovery method.

IV. Discussion

A. Research Suggestions

This empirical study applies feature extraction for source code to improve the performance of IR-based trace links generation. It is shown that feature extraction works best for source code with different annotation densities, because annotations contain a large amount of information that can improve the quality of trace links, but not all features are equally effective in recovering traceability links.

As shown in Table 11, to improve the effectiveness of trace links, different feature extraction methods are used depending on the code features. When the average annotation density is higher than 0.2, it is more effective to perform feature extraction and then generate trace links. The code feature combination with "CN_MN_CMT" is recommended for VSM, and "CN_CMT" is recommended for LSI. In terms of the feature extraction method, JavaParser or CAJP is used to extract code that conforms to the java writing specification. Otherwise, regular expressions are used, where the CAJP can help remove annotation redundancy. For developers, the specification and number of code annotations can greatly improve the effectiveness of generating trace links.

Table 11. Recommendations for the treatment of different code feature

Code Specification		Average Annotations Density		Annotations Redundancy
Yes	No	>0.2	<0.2	
JavaParser, CAJP	regular expressions	code feature extraction	not code feature extraction	CAJP

B. Validity Threats Discussion

There are a number of threats that could limit the validity of the experiments, so this section is primarily devoted to discussing potential threats to the experiments and how we can alleviate or mitigate the following four validity threats.

Conclusion validity: A detailed research design is devised for this empirical study. As shown in Section II, the research questions, datasets selection, data preprocessing, threshold selection, and quality measure selection are reasonably designed to ensure the validity of the conclusions.

Construct validity: The IR-based approach includes pre-processing phase, links generation phase, links refinement phase, etc. For the model selection, VSM and LSI are the most used in IR[18], whereas pure VSM is the most effective in trace link generation[17]. In this study, the links generation stage, as shown in Figure 1, is devised in accordance with the research objectives and research questions, which can effectively support empirical studies on code feature extraction, annotation density, and annotation redundancy. In addition, three widely adopted metrics of P, R, and F are used to analyze the results of this experiment and they are effective to quantify the various situations.

Internal validity: The experimental results show that code feature extraction, annotation density, and annotation redundancy have impacts on the trace link generation from requirements to code, and the empirical study conducted on these three aspects does not have the problem that the impacting factors are inappropriately selected.

External validity: In the experimental section, five commonly used datasets are selected, which reduces the threat of this empirical study producing different results on different systems or projects. In addition, the experimental results show that the study achieves consistent conclusions on both "small" and "large" scale datasets. Therefore, the empirical conclusions presented have a great opportunity and potential to be extended to real software systems and projects. In order for reproducibility verifications, the source code of the experiments can be downloaded from WTU-intelligent-software-development/code-feature-extraction (github.com).

C. Related Works

Researchers have done a great deal of research on trace link generation between requirements and source code, and many IR-based approaches have been proposed. These works are described below.

Kuang et al. target the refinement of the recovery links generated by the IR model through class invocation, class inheritance, class usage, and class data basis[11]. Shao et al. improve the documentation based on LSI by weighting each code document by the inclusion relationship of classes in the code document and refining the generated candidate links through class clustering[8]; all of the above methods are used to class names in the code improves the quality of the IR model for trace link generation. Others argue that annotations would be richer than the information contained in pure code. Therefore, Diaz et al. propose to improve the quality of trace link between code and artifacts by using code ownership information, i.e., developer name and text information pairs in annotations[10]. Nagano et al. use a source code parser to

create syntax trees as a way to establish the link between identifier keywords and keywords in annotations[5]. Shen et al. investigated different types of annotations to compensate for the problem of lexical matching between pure source codes and requirements through the contribution of different types of annotations[7]. All of the above methods are analyzed through a single code feature, which can lead to some important information in the code being overlooked. Therefore, these empirical conclusions provide developers with an empirical study on how to extract code features.

V. Conclusion and Future Work

This paper aims to draw empirical conclusions about code feature extraction to help improve the quality of trace links between requirements and source code generated by IR-based approach. Five datasets are used in this experiment, and three experiments are designed for the IR-based approach in three aspects: code feature extraction, annotation importance assessment, and annotation redundancy removal.

Firstly, it is concluded that code feature extraction can improve the quality of trace links when the average annotation density is higher than 0.2. The code feature combination with "CN_MN_CMT" is recommended for VSM, and "CN_CMT" is recommended for LSI.

Secondly, the method called CAJP is proposed that can help remove annotation redundancy. The results show that annotation redundancy removal does not reduce the quality of trace links. For datasets with high redundancy in code annotations, the annotation redundancy removal can significantly improve the quality of trace link generation.

Finally, the study concludes suggestions for researchers and developers about code writing specifications and feature extraction methods.

In the future, the following two aspects will be worked on. Firstly, the code features will be analyzed with other programming languages, because the code artifacts selected for this paper are all java source code. Secondly, the quality of the IR model may be improved by using different weighting methods or relationships between the code features.

Acknowledgment

This work is supported by the National Natural Science Foundation of China Project (No. 62102291), the Young Talents Programmer of Scientific Research Program of the Hubei Education Department (Project No. Q20211711) and the Opening Foundation of Engineering Research Center of Hubei Province for Clothing Information (No. 2022HBCI02, No. 2022HBCI05).

References

[1] O. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," Proceedings of IEEE International Conference on Requirements Engineering, pp. 94-101, 1994.

[2] B. Wang, R. Peng, Y. Li, et al.,"Requirements traceability technologies and technology transfer decision support: A systematic review," Journal of Systems and Software, vol.146, pp.59-79, 2018.

[3] D. Li, W. E. Wong, S. Pan, L.-S. Koh, S. Li, and M. Chau, "Automatic test case generation using many-objective search and principal component analysis," IEEE Access, vol. 10, pp. 85518–85529, 2022.

[4] S. Maza and O. Megouas, "Framework for trustworthiness in software development," International Journal of Performability Engineering, vol. 17, no. 2, pp. 241–252, 2021.

[5] S. Nagano, Y. Ichikawa and T. Kobayashi, "Recovering Traceability Links between Code and Documentation for Enterprise Project Artifacts," 2012 IEEE 36th Annual Computer Software and Applications Conference, 2012, pp. 11-18.

[6] D. Li, W. E. Wong, M. Jian, Y. Geng, and M. Chau, "Improving search-based automatic program repair with Neural Machine Translation," IEEE Access, vol. 10, pp. 51167–51175, 2022.

[7] G. Shen, H. Wang, Z. Huang, Y. Yu, K. Chen, "Supporting Requirements to Code Traceability Creation by Code Annotations," International Journal of Software Engineering and Knowledge Engineering, vol.31, pp.1099-1118, 2021

[8] J. Shao, W. Wu and P. Geng, "An Improved Approach to the Recovery of Traceability Links between Requirement Documents and Source Codes Based on Latent Semantic Indexing," 13th International Conference on Computational Science & Its Applications, Springer Berlin Heidelberg, 2013.

[9] R. Tsuchiya, H. Ishizaki, Y. Fukazawa, T. Kato, M. Kawakami, K Yoshimura, "Recovering Traceability Links between Requirements and Source Code Using the Configuration Management Log," IEICETransactions on Information and Systems, vol.98-D, pp, 852-862 ,2015.

[10] D. Diaz, G. Bavota, A. Marcus, et al., "Using code ownership to improve IR-based Traceability Link Recovery," 2013 21st International Conference on Program Comprehension (ICPC), pp. 123-132, 2013.

[11] H. Kuang, J. Nie, H. Hu, et al., "Analyzing closeness of code dependencies for improving IR-based Traceability Recovery". 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 68-78, 2017.

[12] Nasir Ali, Yann-Gael Gu ¨ eh´ eneuc, Giuliano Antoniol, "Requirements Traceability for Object Oriented Systems by Partitioning Source Code", IEEE, 2011.

[13] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," Communications of the ACM, vol. 18, no. 11, pp. 613–620, Nov. 1975

[14] Mahmoud, A., Niu, N. "On the role of semantics in automated requirements tracing". Requirements Eng 20, 281–300 (2015).

[15] S. Eder, H. Femmer, B. Hauptmann and M. Junker, "Configuring Latent Semantic Indexing for Requirements Tracing," 2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing, pp. 27-33, 2015.

[16] B. Wang, R. Peng, Z. Wang, et al., "An Automated Hybrid Approach for Generating Requirements Trace Links," International Journal of Software Engineering and Knowledge Engineering, 2020.

[17] J. H. Hayes, A. Dekhtyar and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods", IEEE Trans. Software Eng. Vol.32, pp.4-19, 2006.

[18] B. Wang, H. Wang, R. Luo, S. Zhang, Q. Zhu, A Systematic Mapping Study on Information Retrieval Approaches in Requirements Traceability, pp.1-6, July, 2022.