

MEIM: A Multi-Source Software Knowledge Entity Extraction Integration Model

Wuqian Lv¹, Zhifang Liao^{1,*}, Shengzong Liu² and Yan Zhang³

¹School of Computer Science and Engineering, Central South University, Changsha, 410075, China

²School of Information Technology and Management, Hunan University of Finance and Economics, Changsha, 410205, China

³School of Computing, Engineering and Built Environment, Glasgow Caledonian University, Glasgow, G4 0BA, UK

*Corresponding Author: Zhifang Liao. Email: zfliao@csu.edu.cn

Received: 01 July 2020; Accepted: 25 July 2020

Abstract: Entity recognition and extraction are the foundations of knowledge graph construction. Entity data in the field of software engineering come from different platforms and communities, and have different formats. This paper divides multi-source software knowledge entities into unstructured data, semi-structured data and code data. For these different types of data, Bi-directional Long Short-Term Memory (Bi-LSTM) with Conditional Random Field (CRF), template matching, and abstract syntax tree are used and integrated into a multi-source software knowledge entity extraction integration model (MEIM) to extract software entities. The model can be updated continuously based on user's feedbacks to improve the accuracy. To deal with the shortage of entity annotation datasets, keyword extraction methods based on Term Frequency–Inverse Document Frequency (TF-IDF), TextRank, and K-Means are applied to annotate tasks. The proposed MEIM model is applied to the Spring Boot framework, which demonstrates good adaptability. The extracted entities are used to construct a knowledge graph, which is applied to association retrieval and association visualization.

Keywords: Entity extraction; software knowledge graph; software data

1 Introduction

In the construction of knowledge graphs, knowledge entity extraction is a fundamental step. The quantity and accuracy of knowledge entities have an important impact on subsequent steps such as relationship establishment, knowledge fusion and knowledge graph application. For example, if the number of entities is too small, the entity relationship will be limited and the results obtained from knowledge graph retrieval could be ineffective. Designing a method to extract knowledge entity reasonably and effectively is important for knowledge graph construction.

Traditional knowledge entity extractions mainly target unstructured data and use deep learning models to extract knowledge entities. This type of method has been widely used in medical field and journalism field. In the medical field, researchers identify and extract medical entities in electronic medical records and use



This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

them to construct medical knowledge graphs. In the journalism field, researchers usually focus on extracting entities in three categories: Person, Location, and Organization.

In the field of software engineering, software is composed of code and documents which exist in a large number of different types of data, so the extracted data sources are diverse, including Source Code, eXtensible Markup Language (XML) files, JavaScript Object Notation (JSON) files, Question and Answer (Q&A) records, Version Control System records, etc. They come from different software communities, such as GitHub, StackOverflow, Sourceforge, etc. These data sources not only contain unstructured data, but also a lot of semi-structured data and code data.

At present, the majority of the research on software knowledge entity extraction is aimed at a single type of data. To build a complete and meaningful software knowledge graph, software knowledge entities should be extracted from multiple data sources. This paper proposes MEIM: A multi-source software knowledge entity extraction integrated model, which is used to implement entity extraction for data input in different formats.

Main contributions presented in this paper include: 1) The definition of entity categories in the field of software engineering. 2) A method combining TF-IDF, TextRank and K-Means is proposed for software entity data annotation, which can efficiently obtain a large number of datasets. 3) Rules of template matching are defined for semi-structured data, and analysis tool is used to achieve the extraction of entities. 4) Different types of data extraction methods are integrated and a software entity extraction integration model, which is able to improve its accuracy incrementally, is built.

2 Related Work

Current research in the field of knowledge graph and entity extraction mainly focuses on the following aspects:

Zhao et al. [1] proposed the Harvesting Domain Specific Knowledge Graph (HDSKG) framework to discover domain specific concepts and their relation triples from the content of webpages. They incorporated dependency parser with rule-based method to chunk the relation triple candidates. Then advanced features of these candidate relation triples were extracted to estimate the domain relevance by a machine learning algorithm.

Guo [2] optimized the HDSKG framework and proposed a strategy for the extraction of Wiki pages in the field of software engineering. In this work, web page titles were used to construct domain dictionaries, then rules based on entity conceptual features were designed in the field of software engineering. Finally, the constructed domain dictionaries were used to improve the accuracy of subsequent entity recognition. Researchers [3–5] use different methods to extract entities in web. In the field of open source software, Liao et al. [6,7] expanded the scope of software knowledge to open source software domain and proposed recommendation and prediction methods for social networks and ecosystems.

Ye et al. [8] analyzed the challenges of entity recognition in software engineering, and proposed a method based on machine learning for social content of software engineering. They combined labeled data, unlabeled data and other social content data of a Q&A website to train the model, which can be applied to various software entities in different popular programming languages and platforms.

Hang et al. [9] proposed the DeepLink framework to realize the link recovery of Issue and Commit in GitHub. A code knowledge graph was constructed and text semantic information of Issue and Commit is combined to complete the link recovery.

Xiao et al. [10] applied knowledge graph to the field of software security. He integrated heterogeneous software security concepts and examples of different databases into a knowledge graph, and then developed a knowledge graph embedding method which embeds symbolic relational and descriptive information of

software security entities into a continuous vector space. The generated results can be used to predict software security entity relationships. knowledge graphs have also been used to find the defects in software [11]. Chen et al. [12] proposed a method of combining recurrent neural networks with dependency parser to extract error entities and their relationships from error reports.

Lin et al. [13] built an intelligent development environment based on the software knowledge graph and realized software text semantic search. Based on the work of Lin, Wang et al. [14] proposed a method to convert natural language questions into structured Cypher queries. These queries can be used in graph database Neo4j to return the corresponding answers. Ding et al. [15] identifies the primary studies on knowledge-based approaches in software documentation.

3 Methodology

In order to solve the problem of multi-source software entity knowledge extraction, an integrated extraction model is designed in this paper. The model integrates data classification, source code data extraction, semi-structured data extraction and unstructured data extraction. In this section, the framework of the proposed multi-source software knowledge entity extraction integration model and the specific implementation methods of each functional module are introduced.

3.1 Framework for Integrated Models

The framework of multi-source software knowledge entity extraction integration model (MEIM) is presented in Fig. 1.

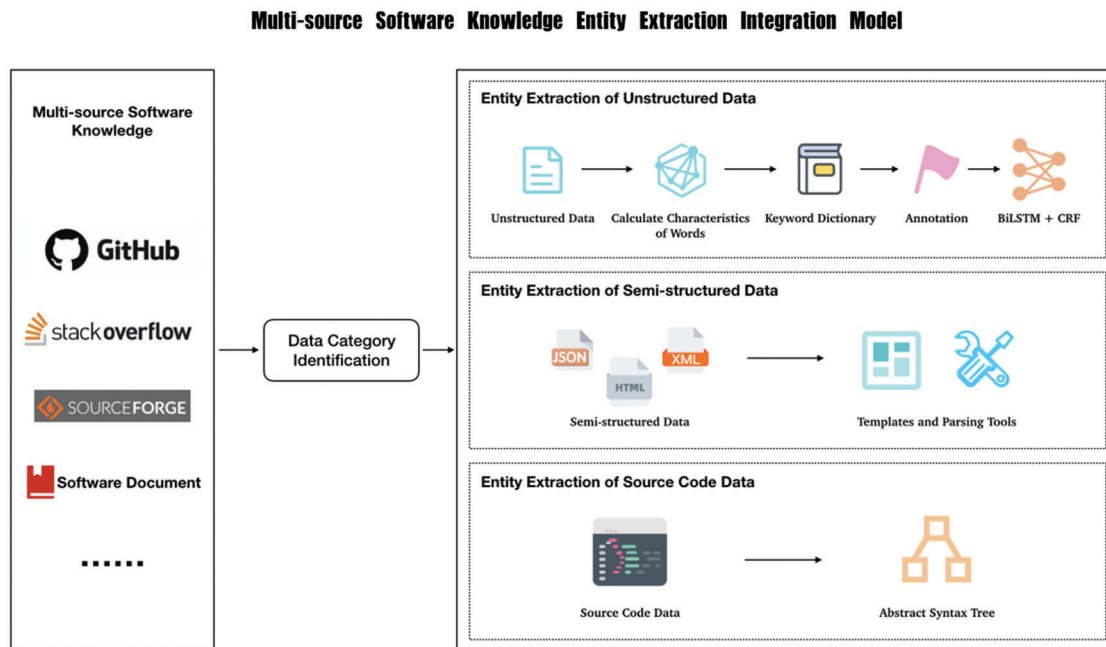


Figure 1: Framework of multi-source software knowledge entity extraction integration model

The first part is the input of multi-source software knowledge. Open source software usually consists of code and documentation, which exist in a large number of software platforms and communities, such as, GitHub, StackOverflow, Sourceforge, etc.

The second part is the entity extraction module for various types of data, which includes three extraction sub-modules of unstructured data, semi-structured data and source code data. For unstructured data, characteristics of the words are used to extract keywords from data, then these keywords are manually labelled. After that, Bi-LSTM + CRF method is used to train a model. Finally, the model is used to extract entities from the unstructured data. For semi-structured data, template matching and parsing tools are applied to mine data patterns and extract entities. For source code data, the code is parsed into abstract syntax trees and the tree is traversed to obtain the code entities.

3.2 Unstructured Data Extraction

3.2.1 Dataset Processing

At present, there is no good open source entity dataset for sequence labeling in the field of software engineering, so an entity dataset is constructed. We crawled 20,000 posts from the well-known IT technology question and answer site, Stackoverflow, and selected 500 posts for dataset construction. Based on TF-IDF, TextRank, and K-Means methods, more than 3000 keywords are extracted from them, and these keywords are manually labelled using BIO labeling methods. Finally, keywords set is used to annotate the original dataset.

Text preprocessing: Firstly, all text is converted to lowercase and natural language toolkit (NLTK) is used to label part-of-speech (POS) tag. And the tokenized text (mainly the nouns and adjectives) is normalized by lemmatization tool. For example, “classes” is replaced by “class.” We used the English Stopwords List provided by Ranks NL to remove the stop words in the text. Then, based on the position of the stop words, the text was segmented to generate words and phrases. The phrases are also candidates for keywords to be extracted.

TF-IDF: TF-IDF algorithm is a classic algorithm based on word frequency statistics. TF (Term Frequency) refers to the frequency with which a given word appears in the current text. IDF (Inverse Document Frequency) is related to the total number of texts containing a given word. The smaller the total number of texts, the greater the IDF value. The basic idea of TF-IDF is that the importance of a word increases proportionally with the number of times it appears in the text, but at the same time it decreases inversely with the frequency of its appearance in the text library. For example, in a piece of text, the word “the” may appear frequently, but it appears frequently in all texts, so the IDF value is low. So final TF-IDF score is also low. All posts are used to build a text library to improve the accuracy of the IDF value and calculate the score of each word according to the Eqs. (1)–(3).

$$S(W_i) = TF(W_i) * IDF(W_i) \quad (1)$$

$$TF(W_i) = \frac{C(W_i)}{n} \quad (2)$$

$$IDF(W_i) = \log\left(\frac{m}{H(W_i) + 1}\right) \quad (3)$$

where $S(W_i)$ represents the TF-IDF score of the i -th word, $TF(W_i)$ represents the TF score of the i -th word, $IDF(W_i)$ represents the IDF score of the i -th word, $C(W_i)$ represents the number of times the i -th word appears in a text, n represents the total number of words in this text, $H(W_i)$ represents the number of texts where the i -th word appears in all text libraries, m represents the total number of texts in the text library.

Then words are sorted by TF-IDF score in descending order to obtain the keyword set for each post.

TextRank: TextRank algorithm is a graph-based model based on the idea of PageRank algorithm. The basic idea is: If a word is linked by a large number of words or by a highly ranked word, it means that the

word is more important. Therefore, we build a graph. Each word serves as a node of this graph. And edges are constructed between related words.

Firstly, a dictionary is built for the text, and all words in the dictionary become nodes in the graph. Then edges are generated in the graph using a Window which slides from the beginning of the original text to the end of the original text. Weighted edges are generated between words in a Window, which are determined by the distance between them. Since the relationship of words is mutual, we construct undirected edges. A two-dimensional array is used to store the weights of the edges between all vocabularies. For each occurrence of two related words, the weight of the corresponding edge increases according to Eq. (4).

$$W = \frac{1}{|I_1 - I_2|} \quad (4)$$

where W represents weight, I_1 represents the absolute position of the first word, I_2 represents the absolute position of the second word.

After each calculation, the absolute position of two words are stored in a set to avoid repeated calculations in the same Window. After traversing the original text, a word graph is constructed, and the score of each word node is calculated based on the word graph.

Initializing the score of all word nodes to 1, the score of each word node is iteratively updated according to Eq. (5).

$$S(V_i) = (1 - \alpha) + \alpha * \sum_{j \in E(V_i)} \frac{S(V_j)}{W(V_j)} \quad (5)$$

where $S(V_i)$ represents the score of the i -th node, α represents the damping factor, $E(V_i)$ represents the set of connected point numbers of the i -th word node, $W(V_j)$ represents the sum of weights of all edges of the j -th word node.

The score of the word node is calculated iteratively until it converges to the given threshold or reaches the preset number of iterations. Word nodes are sorted according to the score in descending order to obtain the keyword set.

K-Means: K-Means is a clustering algorithm. We give K cluster starting center points. The algorithm calculates the Euclidean distance from each point to the center point, dividing the points into clusters that contain the nearest center point. Then the algorithm recalculates the center point of each cluster, repeating the above steps until convergence.

Word2vec tool is used to load Google open source English pre-trained word vectors to convert all words into word vectors. To use the K-Means clustering algorithm, the K value needs to be set, and the determination of the K value depends on the Calinski–Harabasz Score (Eqs. (6)–(8)).

$$S(K) = \frac{tr(B_K)}{tr(W_K)} * \frac{N - K}{K - 1} \quad (6)$$

$$B_K = \sum_{q=1}^K n_q (c_q - c_E)(c_q - c_E)^T \quad (7)$$

$$W_K = \sum_{q=1}^K \sum_{x \in P_q} (x - c_q)(x - c_q)^T \quad (8)$$

where $S(K)$ represents K cluster scores, $\text{tr}(x)$ is used to take the diagonal elements of the matrix, N is the number of all sample points, B_K is the inter-class discrete matrix, W_K is the discrete matrix within the class, n_q is the total number of sample points of class q , c_q is the center point of the class, c_E is the center point of all sample points, P_q is all sample points of class q .

A part of the words which are closest to each central point is used to form a keyword set.

Generation of the keywords set: Based on the experimental results, the top ranked keywords are extracted from the three types of keyword set at a ratio of 3:2:1. After deduplication, the final keywords set to be annotated is obtained.

Entity category definition: We collected developers' suggestions for the classification of software entities, and finally determined several categories of software entities as shown in [Tab. 1](#). They are File, Programming Language, Application Programming Interface, External Tools and Dependencies and Standard.

Table 1: Software entity category

Category	Tag	Examples
File	F	Files: hello.java Folders: src/main/java
Programming Language	PL	Programming language: C, Java, JavaScript, Python Classes: String Packages: java.util.List
Application Programming Interface	API	Functions and methods: printf("Hello"), System.out.print("Hello") Interfaces: EventListener Platforms: Ubuntu, Android Tools: Neo4j, MySQL
External Tools and Dependencies	ETD	Libraries: NumPy, Pandas Frameworks: Spring Boot Others: Visual Studio, etc Formats: JSON, XML Protocols: HTTP
Standard	S	Technology acronyms: USB, URL Others: Observer Pattern, etc

Annotation of the keywords set: 5 developers with rich development experience annotated the keyword set according to the entity category table in the form of BIO. The BIO annotation method labels each element as "B-X," "I-X," or "O." Among them, "B-X" means that the word belongs to the X type and is located at the beginning of a phrase. "I-X" means that the word belongs to the X type but do not appear at the beginning of a phrase. "O" means it does not belong to any type. [Fig. 2](#) shows an example of BIO annotation. After that, the keywords set is used to annotate the original dataset.

3.2.2 Model Training

After the sequence labeling, we preprocess the data and obtain word vectors for word and POS tag, and use these two kinds of vectors as input of the model. The core of the model is Bi-LSTM + CRF, and the output is a sequence annotation of each word.

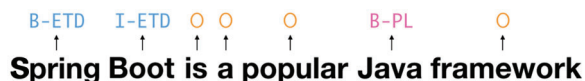


Figure 2: An example of BIO annotation

Data preprocessing: Firstly, NLTK is applied to the dataset for POS tag and filtering out sentences without nouns. Then we divide the dataset into a training set and a test set, where the test set is accurately annotated manually and accounts for 15%. The sentences in the training set are used to create the word and POS tag vectors.

Bi-LSTM-CRF: Bi-LSTM-CRF is currently one of the most widely used sequence labeling models. For sequence labeling, it is effective to consider the contextual content of each word and the legal order of the tag sequence. Bi-LSTM can add context features to the training process, and CRF can output globally optimal sequences. The combination of them can complete the task of sequence labeling effectively. In this paper, the structure of the Bi-LSTM-CRF model includes three layers as shown in Fig. 3.

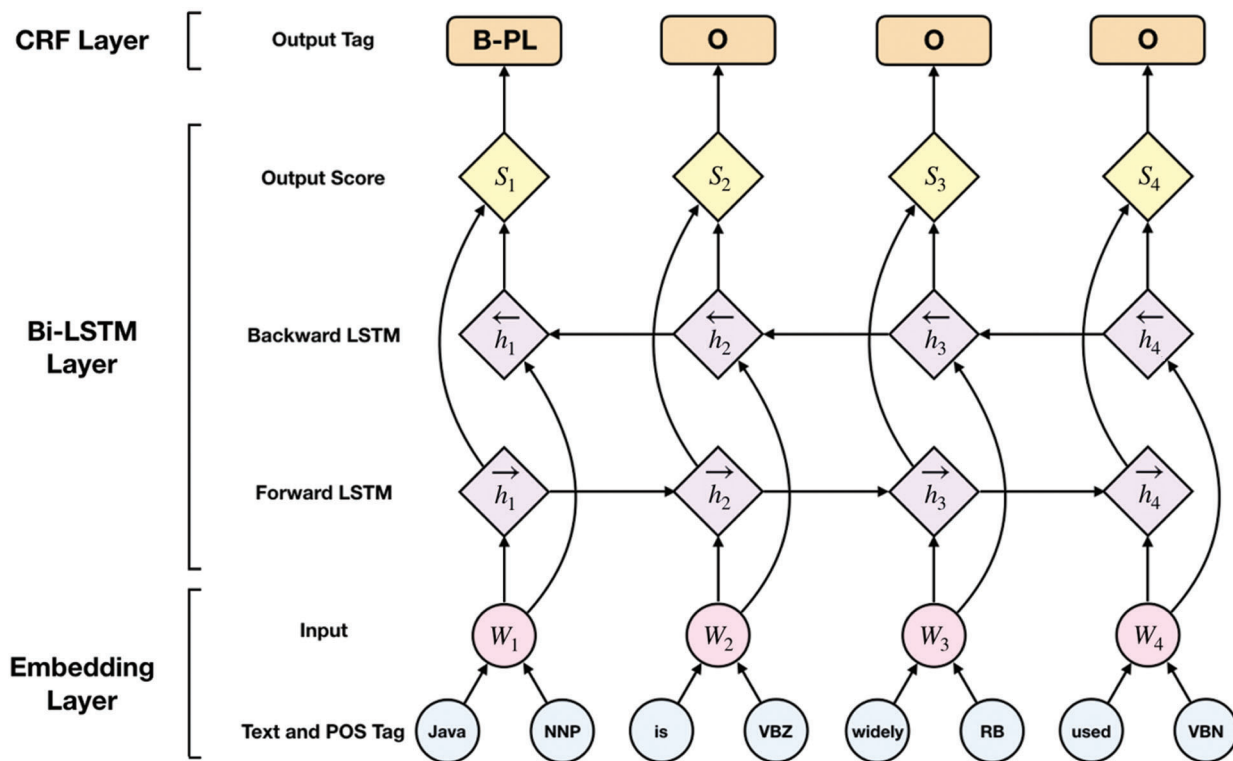


Figure 3: Bi-LSTM + CRF model structure

The first layer is the Embedding Layer, which inputs word and POS tag vectors and adds Dropout to the two categories of word vectors to prevent overfitting.

The second layer is composed of a Forward LSTM Layer and a Backward LSTM Layer, and Dropout is also added. Forward LSTM adds information before the word, and Backward LSTM adds information after the word. In this way, contextual information can be used with word order and meaning combined. LSTM calculates the current time value h_t by combining the cell state C_{t-1} , output value h_{t-1} and current time input x_t . The sequence $(\vec{h}_1, \vec{h}_2, \dots, \vec{h}_n)$ calculated by Forward LSTM and the sequence $(\overleftarrow{h}_1, \overleftarrow{h}_2, \dots, \overleftarrow{h}_n)$ calculated by

Backward LSTM are combined into $([\vec{h}_1 : \overleftarrow{h}_1], [\vec{h}_2 : \overleftarrow{h}_2] \dots [\vec{h}_n : \overleftarrow{h}_n])$ and outputted to the LSTM Output Layer. The output of this layer is a score for each sequence corresponding to various sequence labels.

The third layer is the CRF Layer, which adds constraints to the last predicted label to ensure that the predicted label is legal. These constraints are learned in the training set through the CRF layer. For example, each sentence must start with “B-label” or “O,” and “O” cannot be followed by “I-label,” etc. Each label is used as a node to construct linear chain CRFs, and a two-dimensional matrix is used to store the transfer score from one label to another, determining the output sequence.

3.3 Semi-Structured Data Extraction

Semi-structured data has a certain structure. A large amount of semi-structured data exists in open source software, such as operation manual in HyperText Markup Language (HTML) format, configuration files in XML format, data storage files in JSON format, etc. We take HTML, XML and JSON data as examples, introducing the extraction methods of semi-structured data in the integrated model.

3.3.1 HTML

HTML files in open source software include operation manuals, user guides and static front pages. According to the experience, operation manuals and user guides contain more valuable entities than static front pages. Therefore, our extraction of HTML is focused on these two types of documents.

The HTML parser and template matching method are used to extract entities in HTML. The templates in [Tab. 2](#) are used to extract some software entities in HTML.

Table 2: Extraction templates

Entity Type	Regular Expression	Example
File	$\backslash s(\backslash w^*/)(\backslash w^+)(\backslash .)\{0,1\} \backslash w^* \backslash s$	src/test/java/com/example/springboot/ HelloControllerIT.java
File	$\backslash s(\backslash w^+\backslash .)\backslash w^+ \backslash s$	org.springframework.boot.autoconfigure
Application Programming Interface	$(\backslash w^+\backslash .)^* \backslash w^+ \backslash ((\backslash w)^*)$	System.out.println(message)
Other	<code><code>\S*</code></code>	<code><code>spring-boot-gradle-plugin</code></code>

BeautifulSoup is used to parse HTML files. 1) All `<a>` tags are found through BeautifulSoup, using the text related to the `<a>` tag as an entity, and the link address as an attribute of the entity. 2) The `<code>` tags and their contents with the data-lang attribute are found. Then all HTML tags are cleared and the code is spliced. After that, the data-lang attribute value and the spliced code text are input to the source code data entity extraction module. 3) All the code already used in the HTML file and all HTML tags are cleared. The remaining text are input to the unstructured data entity extraction module.

User guide of the famous development framework Spring Boot is taken as an example to explain our entity extraction method. As shown in [Fig. 4](#), this is a part of the user guide and its corresponding HTML code.

For the “src/main/java/com/example/springboot/HelloController.java,” it will be recognized by regular expressions. And we extract “HelloController.java” as a File entity. Regular expressions can also identify entities such as “@RestController” and “@RequestMapping” in web pages.

Create a Simple Web Application

Now you can create a web controller for a simple web application, as the following listing (from

src/main/java/com/example/springboot/HelloController.java) shows:

```
package com.example.springboot;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!";
    }

}
```

COPY

The class is flagged as a `@RestController`, meaning it is ready for use by Spring MVC to handle web requests. `@RequestMapping` maps `/` to the `index()` method. When invoked from a browser or by using curl on the command line, the method returns pure text. That is because `@RestController` combines `@Controller` and `@ResponseBody`, two annotations that results in web requests returning data rather than a view.

```
...
<h2 id="initial">Create a Simple Web Application</h2>
...
<p>Now you can create a web controller for a simple web
application, as the following listing (from <code>src/main/java/
com/example/springboot/HelloController.java</code>) shows:</p>
</div>
...
<code class="language-java" data-lang="java">package
com.example.springboot;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!";
    }

}</code>
...
<p>The class is flagged as a <code>@RestController</code>,
meaning it is ready for use by Spring MVC to handle web
requests. <code>@RequestMapping</code> maps <code>/</code> to
the <code>index()</code> method. When invoked from a browser or
by using curl on the command line, the method returns pure
text. That is because <code>@RestController</code> combines <
code>@Controller</code> and <code>@ResponseBody</code>, two
annotations that results in web requests returning data rather
than a view.</p>
```

Figure 4: User guide and its HTML code

For the code block, its HTML code is “`<code class="language-java" data-lang="java"> package </code>`.” We will locate it through the `<code>` tag and the `data-lang` attribute in HTML files, then input them to code data extraction module. The remaining text is input to unstructured data extraction module.

3.3.2 XML

In software, most configurations are implemented by static XML configuration files. Therefore, a software usually contains a lot of XML files which consist of software entities.

XML files are mainly parsed by Python Dom. For all nodes, the entity library is searched based on the node name (the entity library consists of labeled data and manual confirmation data). If the category of the corresponding node can be found, the node is extracted as an entity of this category, and the content of the node is extracted as the attribute. If the category to which the corresponding node belongs cannot be found, the node is extracted as an “Other” entity, and the content of the node is extracted as the attribute. For example, the node “`<modelVersion>4.0.0</modelVersion>`” and its content is obtained through the parser. “modelVersion” is used as a keyword to search in the entity library. If the corresponding category can be found, “modelVersion” is extracted as an entity of this category, and “4.0.0” is used as an attribute of the entity. If there is no corresponding result, “modelVersion” is extracted as “Other” category.

After parsing an XML file, the user can choose whether to classify the extracted “Other” entities. Through this step, the user can manually divide these entities into corresponding categories. And these entities will also be used to expand the entity library and to improve the accuracy of extraction model further.

3.3.3 JSON

In software, developers commonly use JSON files to store content and deliver messages. In fact, JSON is a light form of XML. JSON and XML can be mutually converted to some extent. Therefore, the same method as XML is used for JSON entity extraction.

3.4 Source Code Data Extraction

Source code is a unique data source for software. It has a fixed format and a large number of software entities. It is an important data source in the field of software engineering. Taking Java code as an example, we implemented a module for automatically extracting source code entities. QDox open source plug-in is used to parse the code. For the input source code file or folder, QDox iterates automatically and stores the obtained content into a JavaProjectBuilder object to form a tree structure. Using the JavaProjectBuilder object, we can get source files, packages, classes, methods, parameters, etc.

The extraction results of entities and their attributes are shown in [Tab. 3](#).

Table 3: Source code entity and attribute

Entity	Attribute
Class	name, package, fullName, isInterface, isPublic, isAbstract, isFinal, isStatic
Field	name, type, isArray, isStatic
Method	name, return, isArray, isStatic, isPublic
Exception	name
Parameter	name, type
Interface	name, package, fullName
Package	name
Source	name, content

3.5 Module Integration

All input files are classified according to the suffix names and parse the files using corresponding modules to extract all software entities in the files. Users can input software-related files that they want to extract according to the prompts through the user interface. During the extraction process, users can optionally participate in editing and manually determine some “Other” type entities. The entity library is continuously updated with manual confirmation, and the annotation of the source dataset can be updated by calling a new entity library at intervals to improve the accuracy of the unstructured entity extraction model.

3.6 Construction and Application of Knowledge Graph

In order to show the application value of MEIM, a knowledge graph is constructed using the extracted entities. “from” relations is established between all extracted entities and their source file entities. And “related to” relations are established between entities of the same name from different source files. In the source code entities, detailed relations are established according to the relationships in code, which are {extend, implement, have_field, declare_exception, have_parameter, have_method, import}.

The established knowledge graph can realize fast retrieval and association visualization.

4 Experiment

4.1 Keyword Extraction

4.1.1 Comparing the Keyword Extraction Results from the Three Methods

In [Fig. 5](#), the first paragraph includes keywords extracted using the TF-IDF algorithm. This method is good at extracting important vocabularies in text. Overall, it performs best among the three methods.

TF-IDF: mediastore, canvas, stylesheet, wordpress, url, css, pw, joptionpane, restart, fw, printwriter, proguard, cordova, gradle, android, max_length, charfield, ios, asp, net, html, wx, echo, sh, macosx

TextRank: mvc project, targeting net, legacycasmodel true, image, image eta, img bg_damask jpg, jpg, style cs, bg_damask, url, img, dev tool, jpg cs, bitmap, small web project, process eclipse, war, small web, eclipse project, content, sitepage, function, timeouterror function, phinstance, http publ

K-Means: alternative, cluster, claim, instream, request, switch, draw, cool, advance, apollo, fb, alice, py, textbox, stylesheets, imac, software, goto, mf, qt, posix, neu, strip, zip, inline

Figure 5: Keyword extraction results

The second paragraph presents a part of the keywords extracted using the TextRank algorithm. The extracted keywords are slightly worse than TF-IDF. And the importance of keywords is not as high as that of TF-IDF.

The third part is a part of the keywords extracted using the K-Means algorithm. Comparing with the other two methods, the extracted keywords perform poorly in the importance of vocabulary. It extracts some less distinctive words like “software” and “project.” However, the K-Means algorithm has more types of words and includes some words ignored by the other two methods because they only focus on importance. This makes the keyword dictionary more complete.

In the end, the TF-IDF algorithm is used to extract 6000 keywords. The TextRank algorithm is used to extract more than 4000 keywords. And the K-Means clustering algorithm is used to extract 2000 keywords. After deduplication, finally more than 6,000 keywords are extracted. We manually filter and annotate these keywords to create a set of annotated keywords.

4.1.2 Comparison of Keyword Annotation Data and Manual Annotation Data

Fig. 6 shows the results of the keyword annotation (in red) and the manual annotation (in blue). The keyword annotation is quite accurate. The result of keyword annotation is different from the one of manual annotation only in “WebSocket handshake” and “wss endpoint,” and both methods provide same categories. We observed a part of the dataset and noticed that the performance of keyword annotation is reliable and can meet the data requirements. In this way, we reduce repetitive labor and obtain a sequence labeling dataset in the field of software engineering.

This initial handshake is used to upgrade the HTTP(B-S) (B-S) connection to a WebSocket(B-S) (B-S) connection, and using Java(B-PL) (B-PL) you will be talking over secure WebSockets(B-S) (B-S) while you are directly trying to connect to host. Note that the STOMP(B-S) (B-S) client implementation you're using expects a connection frame upon sending connection, but since it does not perform the WebSocket(B-S) (B-S) handshake(I-S) it is not receiving it, hence the EOFException(B-API) (B-API). You can replace the headers with your credentials and also the wss(B-S) (B-S) endpoint(I-S) can be got from the AWS(B-ETD) (B-ETD) MQTT(I-ETD) (I-ETD) server, see the link I shared above for how the wss(B-S) (B-S) uri(I-S) (I-S) looks like.

Figure 6: Annotation result

4.1.3 K Value Selection

In Fig. 7, the distribution of the clustering results (when $K = 2$ and $K = 3$) of the word vectors is displayed with vocabularies reduced to two-dimensional. Fig. 8 illustrates the change of Calinski-Harabasz score along the K value, where the highest value is achieved when $K = 2$. So the final K value is 2.

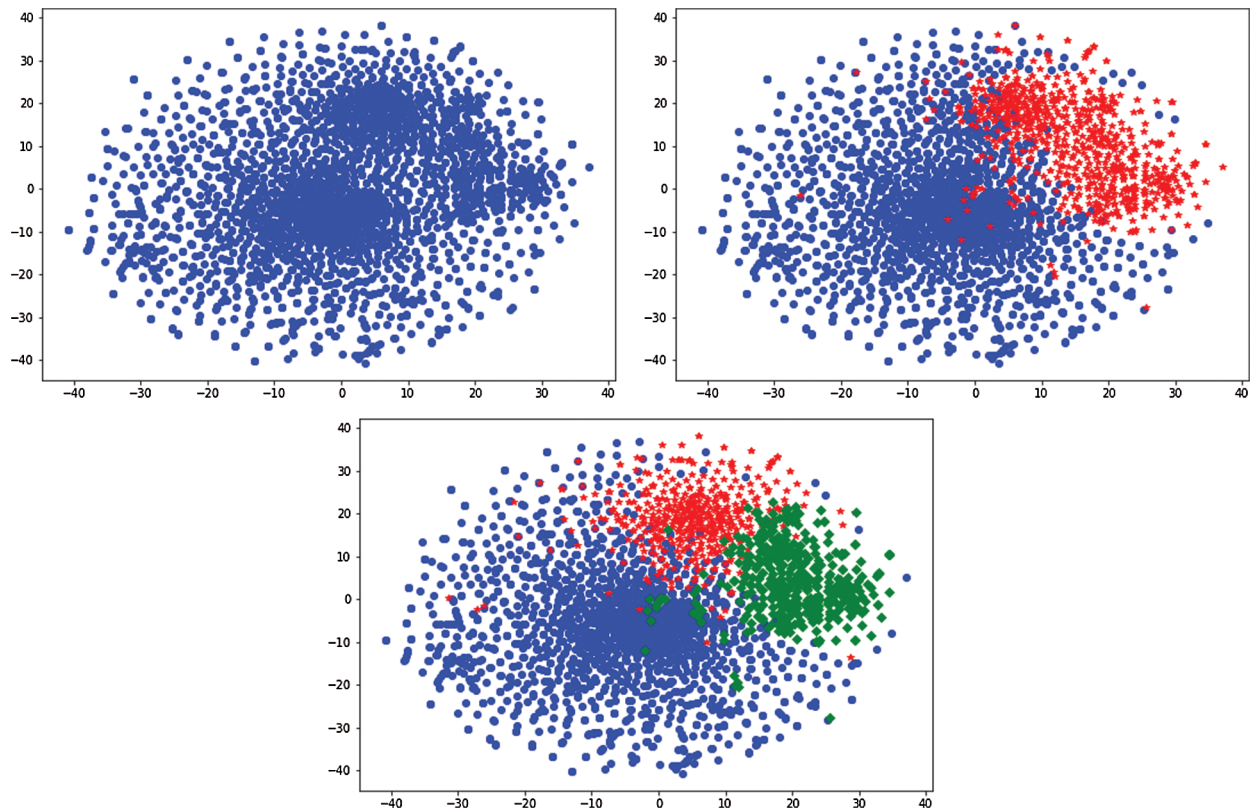


Figure 7: Distribution of word vectors and clustering results when $K = 2, 3$

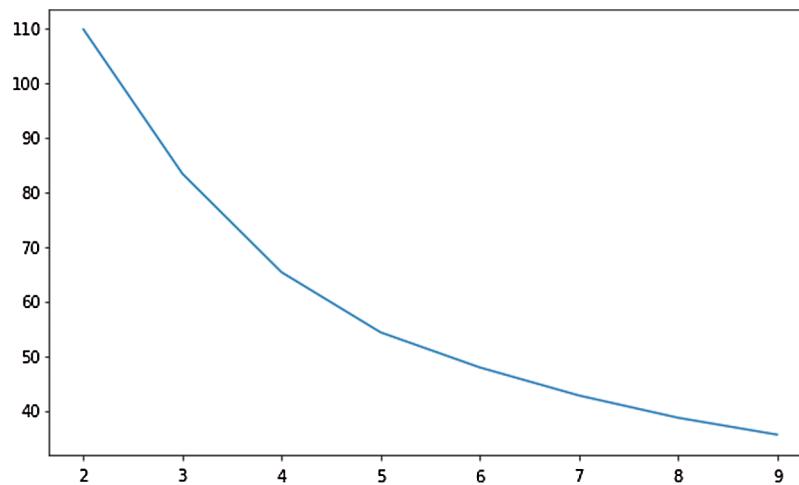


Figure 8: Calinski-Harabasz score curve

4.2 Application of Integrated Model in Spring Boot

4.2.1 Entity Extraction Results

We collected some codes and documents related to the Spring Boot framework from GitHub, StackOverflow and Spring Boot official website and use the proposed model to extract entities. We also crawled Posts from StackOverflow, Issues from GitHub, and documents from the Spring Boot official

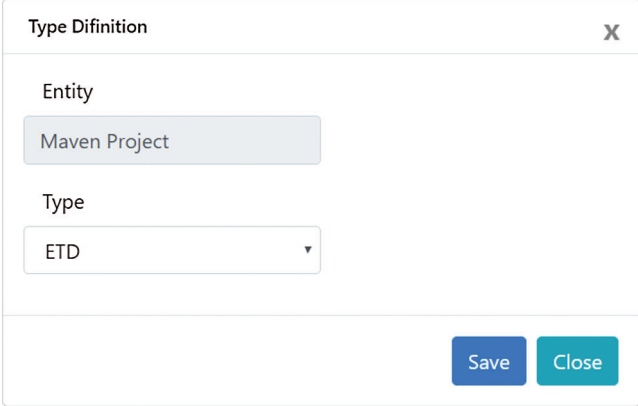
website. We crawled 15 documents in HTML format from the Spring Boot official website, collected files in XML or JSON format from GitHub repository, and downloaded the framework source code from GitHub. The quantity and types of entities finally extracted are shown in [Tab. 4](#).

Table 4: Entity extraction result

File Type	Quantity	File Type	Quantity
F	642	Field	7333
PL	350	Method	24487
API	1454	Exception	3807
ETD	3390	Parameter	13388
S	1606	Interface	361
Other	721	Package	559
Class	4554	Source	4915

4.2.2 Manually Confirm the Entity to Expand the Entity Library

The entities extracted from the semi-structured data are classified according to the entity library, but there are some entities that cannot be matched using the library and are labeled as “Other” type. Manual classification methods are provided for these entities, as shown in [Fig. 9](#). Users can selectively annotate some entities, and these entities will be added to the entity library. The expanded library can be used to improve the accuracy of unstructured data entity extraction models and semi-structured data entity classification.



The image shows a dialog box titled "Type Definition" with a close button (X) in the top right corner. Inside the dialog, there is a section labeled "Entity" with a text input field containing "Maven Project". Below this is a section labeled "Type" with a dropdown menu currently showing "ETD". At the bottom right of the dialog, there are two buttons: "Save" (blue) and "Close" (teal).

Figure 9: Manual classification UI

4.2.3 Construction and Application of Knowledge Graph

[Fig. 10](#) shows the construction result of the knowledge graph using the extracted entity set.

This knowledge graph is applied to related knowledge retrieval and association visualization. As shown in [Fig. 11](#), the Neo4j graph database query language Cypher is used to retrieve “additionalProperties” keywords. After expanding the associations, we can find various entities and relations related to “additionalProperties.”

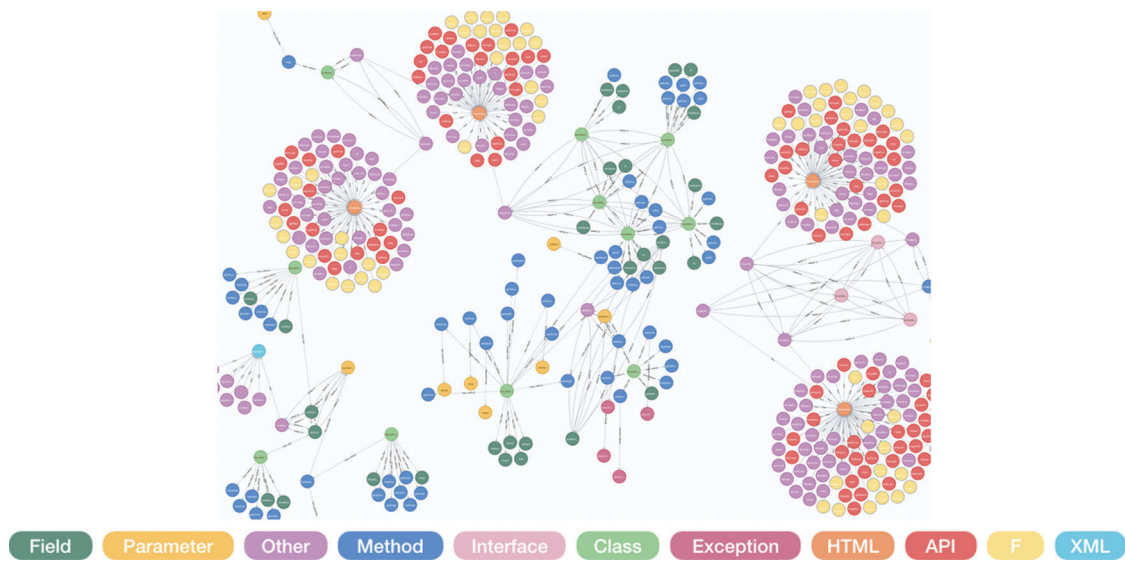


Figure 10: The result of knowledge graph construction

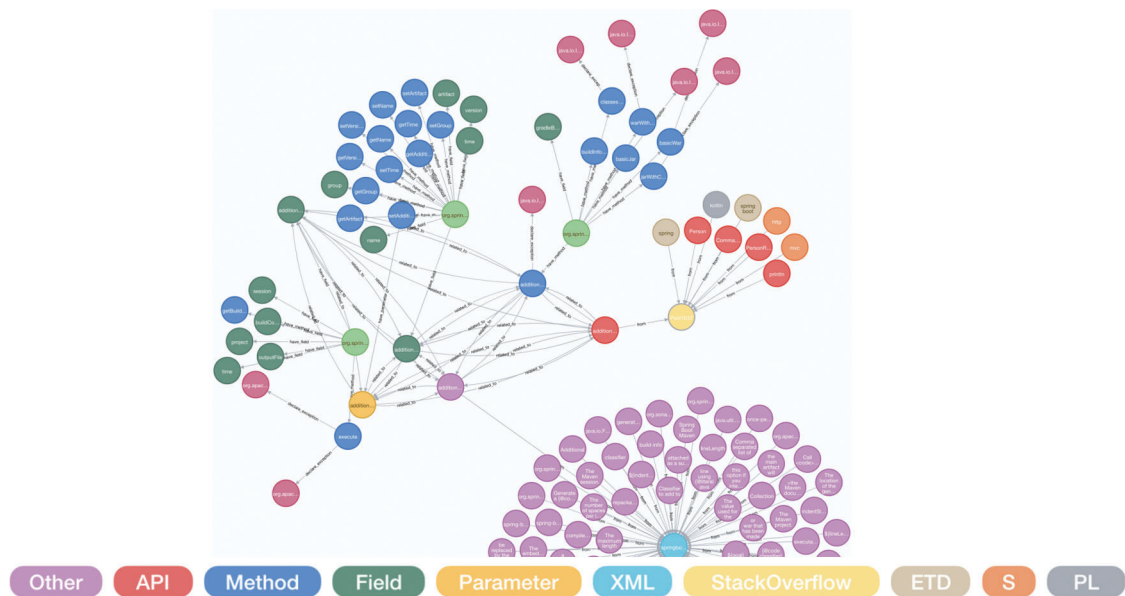


Figure 11: The result of knowledge retrieval and association visualization

The results demonstrate that “additionalProperties” appears in unstructured data, semi-structured data, and source code data. Through the association relations, users can find that there are a variable, a field and a method named “additionalProperties” in the source code. This method belongs to Class “BuildInfoDslIntegrationTests,” and the methods in the same Class also have “warWithCustomName,” “buildforProperties,” etc. “additionalProperties” is also related to a Post on StackOverflow. Users can get relevant information of this Post by association visualization, and learn about the questions and answers related to “additionalProperties.”

Through the application of the knowledge graph, developers and users can easily obtain the associated information of “additionalProperties” in different data sources and visualize knowledge relations. This provides great help for the learning of software development and iterative updating of software.

5 Conclusion and Future Work

This paper proposes MEIM: A multi-source software knowledge entity extraction integration model. In view of the shortage of entity annotation datasets in the field of software engineering, we define the categories of entities for software, and propose a keyword extraction method based on TF-IDF, TextRank and K-Means methods for the annotation task, which quickly expands the scale of labeling training set. Multi-source software knowledge entities are divided into unstructured data, semi-structured data, and code data. For these different types of data, software entities are extracted using Bi-LSTM + CRF, template matching and abstract syntax tree. These methods are integrated into a unique model, MEIM. Accuracy of the model can be improved literately based on user’s feedback. Our integrated model has been tested with Spring Boot framework. A knowledge graph is constructed from the extracted entities, which is used for related knowledge retrieval and association visualization.

In the future, we expect to further optimize the results of entity extraction, add entity alignment algorithms and merge parts of the same entity. At the same time, we will try to pre-extract the relations between entities to improve the accuracy of the later steps of knowledge graph construction.

Acknowledgement: The works that are described in this paper are supported by Ministry of Science and Technology: Key Research and Development Project (2018YFB003800), Hunan Provincial Key Laboratory of Finance & Economics Big Data Science and Technology (Hunan University of Finance and Economics) 2017TP1025 and HNSF 2018JJ2535. We are also grateful to corresponding author Shengzong Liu and his project NSF61802120.

Funding Statement: Zhifang Liao: Ministry of Science and Technology: Key Research and Development Project (2018YFB003800), Hunan Provincial Key Laboratory of Finance & Economics Big Data Science and Technology (Hunan University of Finance and Economics) 2017TP1025, HNSF 2018JJ2535. Shengzong Liu: NSF61802120.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

- [1] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li *et al.*, “Hdskg: Harvesting domain specific knowledge graph from content of webpages,” in *IEEE 24th Int. Conf. on Software Analysis, Evolution and Reengineering*, pp. 56–67, 2017.
- [2] J. Guo, H. Luo and Y. Sun, “Research on extracting named entities in software engineering field from wiki webpage,” in *IEEE Int. Conf. on Consumer Electronics-Taiwan*, pp. 1–2, 2019.
- [3] B. Niu and Y. Huang, “An improved method for web text affective cognition computing based on knowledge graph,” *Computers Materials & Continua*, vol. 59, no. 1, pp. 1–14, 2019.
- [4] H. Lee, J. An, J. Yoon, K. Bae and Y. Ko, “A method to solve the entity linking ambiguity and NIL entity recognition for efficient entity linking based on wikipedia,” *Journal of KIISE*, vol. 44, no. 8, pp. 813–821, 2017.
- [5] Z. Nie, J. R. Wen and W. Y. Ma, “Statistical entity extraction from the web,” *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2675–2687, 2012.
- [6] Z. Liao, Z. X. Wu, Y. Li, Y. Zhang, X. Fan *et al.*, “Core-reviewer recommendation based on pull request topic model and collaborator social network,” *Soft Computing*, vol. 24, no. 8, pp. 5683–5693, 2020.

- [7] Z. Liao, B. Zhao, S. Liu, H. Jin, D. He *et al.*, “A prediction model of the project life-span in open source software ecosystem,” *Mobile Networks and Applications*, vol. 24, no. 4, pp. 1382–1391, 2019.
- [8] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang and J. Li, “Software-specific named entity recognition in software engineering social content,” *IEEE 23rd Int. Conf. on Software Analysis, Evolution, and Reengineering*, vol. 1, pp. 90–101, 2016.
- [9] H. Ruan, B. Chen, X. Peng and W. Zhao, “DeepLink: Recovering issue-commit links based on deep learning,” *Journal of Systems and Software*, vol. 158, 110406, 2019.
- [10] H. Xiao, Z. Xing, X. Li and H. Guo, “Embedding and predicting software security entity relationships: A knowledge graph based approach,” in *Int. Conf. on Neural Information Processing*, pp. 50–63, 2019.
- [11] L. Zhang, M. Gao, P. Li and Y. Jiang, “Construction and applications of embedded aerospace software defect knowledge graph,” *Signal and Information Processing, Networking and Computers*, vol. 628, pp. 470–477, 2020.
- [12] D. Chen, B. Li, C. Zhou and X. Zhu, “Automatically identifying bug entities and relations for bug analysis,” in *IEEE 1st Int. Workshop on Intelligent Bug Fixing*, pp. 39–43, 2019.
- [13] Z. Q. Lin, B. Xie, Y. Z. Zou, J. F. Zhao, X. D. Li *et al.*, “Intelligent development environment and software knowledge graph,” *Journal of Computer Science and Technology*, vol. 32, no. 2, pp. 242–249, 2017.
- [14] M. Wang, Y. Zou, Y. Cao and B. Xie, “Searching software knowledge graph with question,” in *Int. Conf. on Software and Systems Reuse*, pp. 115–131, 2019.
- [15] W. Ding, P. Liang, A. Tang and H. V. Vliet, “Knowledge-based approaches in software documentation: A systematic literature review,” *Information and Software Technology*, vol. 56, no. 6, pp. 545–567, 2014.