A Learning-based Approach for Automatic Construction of Domain Glossary from Source Code and Documentation

Chong Wang*
Fudan University
China

Xin Peng*†
Fudan University
China

Mingwei Liu*
Fudan University
China

Zhenchang Xing
Australian National University
Australia

Xuefang Bai*
Fudan University
China

Bing Xie
Peking University
China

Tuo Wang*
Fudan University
China

ABSTRACT

A domain glossary that organizes domain-specific concepts and their aliases and relations is essential for knowledge acquisition and software development. Existing approaches use linguistic heuristics or term-frequency-based statistics to identify domain specific terms from software documentation, and thus the accuracy is often low. In this paper, we propose a learning-based approach for automatic construction of domain glossary from source code and software documentation. The approach uses a set of high-quality seed terms identified from code identifiers and natural language concept definitions to train a domain-specific prediction model to recognize glossary terms based on the lexical and semantic context of the sentences mentioning domain-specific concepts. It then merges the aliases of the same concepts to their canonical names, selects a set of explanation sentences for each concept, and identifies "is a", "has a", and "related to" relations between the concepts. We apply our approach to deep learning domain and Hadoop domain and harvest 5,382 and 2,069 concepts together with 16,962 and 6,815 relations respectively. Our evaluation validates the accuracy of the extracted domain glossary and its usefulness for the fusion and acquisition of knowledge from different documents of different projects.

CCS CONCEPTS

• **Software and its engineering → Documentation**; • **Computing methodologies → Information extraction**.

*C. Wang, X. Peng, M. Liu, X. Bai, and T. Wang are with the School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, China.
†X. Peng is the corresponding author (pengxin@fudan.edu.cn).

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3338963

KEYWORDS

documentation, domain glossary, concept, knowledge, learning

ACM Reference Format:
Chong Wang, Xin Peng, Mingwei Liu, Zhenchang Xing, Xuefang Bai, Bing Xie, and Tuo Wang. 2019. A Learning-based Approach for Automatic Construction of Domain Glossary from Source Code and Documentation. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338906.3338963

1 INTRODUCTION

Software projects usually belong to specific domains, for example, TensorFlow [18] and PyTorch [14] are deep learning libraries, Hadoop [6] and HBase [7] are distributed database systems. Each domain has a set of business or technical concepts. These concepts are frequently mentioned in source code and documentation of software projects in a domain. They can be mentioned in full names as well as various aliases (e.g., abbreviations and other morphological variants). Many concepts are correlated, such as hypernym-hyponym, whole-part, etc. Organizing domain-specific concepts, their aliases and relations in a domain glossary is essential for knowledge acquisition and software development. Moreover, based on a domain glossary we can semantically link the elements (e.g., classes/methods, APIs, Q&A posts, document fragments) from different artifacts. These links can facilitate many software engineering tasks such as developer question answering [50, 57], traceability recovery and maintenance [32, 45, 55, 56], feature location [29, 54], API recommendation [44, 49], and code search [39].

Due to the complexity and fast development of a domain, it is often laborious and expensive to manually construct a comprehensive domain glossary. Therefore, researches have been advocated to investigate automatic extraction of glossary terms or domain concepts from different kinds of software artifacts. Some researches [22, 30, 40] extract glossary terms from requirements documents. These approaches use linguistic heuristics or term-frequency-based statistics to identify domain specific terms from noun phrases, and thus the accuracy is often low. Moreover, these approaches only cluster relevant terms together and cannot identify the aliases and

relations of a specific concept. Some researches [26, 60] extract domain concepts and relations from Q&A websites such as Stack Overflow, but rely on Stack Overflow tags and human labelled data for the estimation of domain relevance.

In this paper, we aim at extracting domain-specific concepts, their aliases and relations from both source code and documentation of software projects in the same domain in an unsupervised way. The challenges for this task lie in several aspects (see examples in Section 2). First, knowledge about domain concepts and their relations is often scattered in different documents or even different projects. Second, the same concepts are often mentioned in different aliases in different places. Third, the relevance of concepts to a specific domain cannot be reliably determined by lexical heuristics or term frequency.

In this work, we propose a learning-based approach for automatic construction of domain glossary from source code and software documentation. The basic idea of our approach is twofold. First, domain-specific concepts in documentation are often used as identifiers in source code and the relations between concepts may be inferred from structural relations between corresponding code elements. Second, using the sentences that mention the high-quality seed terms identified from code identifiers and natural language concept definitions as training data, we can learn a domain-specific prediction model to recognize more glossary terms based on the lexical and semantic context of the sentences mentioning domain-specific concepts. Based on these two ideas, our approach first extracts a set of candidate terms from the source code and documentation of different projects of the target domain, and then merges the aliases of the same concepts to their canonical names. After that, our approach selects a set of explanation sentences for each concept, and further identifies "is a" (hypernym-hyponym), "has a" (whole-part), "related to" relations between the concepts.

We implement the approach and conduct an empirical study with two technical domains (deep learning and Hadoop). The study leads to the following findings. First, our approach outperforms a state-of-the-art approach [22] in documentation based domain glossary extraction. Second, domain glossary extraction provides an effective way to fuse the domain knowledge from different documents of different projects. Third, the extracted domain glossary complements general knowledge bases such as WikiPedia. Fourth, the extracted domain glossary can help developers to acquire the required knowledge from documents more efficiently.

This work makes the following contributions.

1) We propose a learning-based approach for automatic construction of domain glossary from source code and documentation.

2) We apply the approach to the deep learning domain and the Hadoop domain and harvest 5,382 and 2,069 concepts together with 16,962 and 6,815 relations respectively.

3) We evaluate the effectiveness of the approach for domain glossary extraction and the usefulness of the extracted domain glossary for knowledge fusion and software knowledge acquisition.

2 MOTIVATION

Domain concepts are often mentioned in various aliases. When encountering an alias in code or documentation that developers do not understand, the first question to ask is what this alias represents. For example, "NAS" is the abbreviations of "Neural Architecture Search" in deep learning domain, but it is hard to find explanations for it by searching Google. In this situation, just knowing the full name of the alias' corresponding domain concept is already very helpful for developers. Knowing the name of a domain concept is of course not enough. Because of the lack of domain knowledge, developers further need to know the domain-specific meaning of the concept. For example, in deep learning domain, "LSTM" is the abbreviation of "Long Short Term Memory" which is an advanced neural network structure that many developers may not know. An explanation of the concept will greatly help the developers understand it.

In addition to understand the meaning of domain-specific concepts, it is also very important for the developers to establish an understanding of the relationships between domain concepts, such as hypernym-hyponym, whole-part, or semantically correlated. Knowing the relationships between domain concepts can help developers understand domain knowledge in source code and documentation, and help them make informed decisions in adopting certain techniques. For example, a developer is learning to use Seq2Seq model. When she knows the relation {Seq2Seq, has a, RNN}, {RNN, has a, RNN Cell}, and {LSTM Cell, is a, RNN Cell}, she can naturally think of using LSTM Cell to replace the RNN Cell in the Seq2Seq model. Furthermore, when she knows more variants of RNN Cell (e.g., GRU Cell), she will soon be able to build up her understanding of Seq2Seq. This knowledge may change the developer's choice of more advanced model in her work.

Therefore, a solution to build domain glossary automatically is needed. This solution can exploit the domain concepts mentioned in source code and documentation. However, the following challenges must be addressed. First, domain concepts and their explanations and relationships are often scattered in many documents of different projects. For example, Pytorch [14] has a package *nn*, but we cannot find a sentence for explaining what *nn* is in its tutorials or reference documentation. Meanwhile, Tensorflow also has a package *nn* and we can find a sentence in its API reference documentation that explains the purpose of the package: "Wrappers for primitive Neural Net (NN) Operations". From this sentence we may infer that *nn* in PyTorch also stands for Neural Net. Thus it is important to fuse the same concepts (may be in different aliases) mentioned in different projects of the same domain. Second, the same concepts are often mentioned in various aliases in different parts of the documents. For example, "neural network" is mentioned in different forms in Deeplearning4j, TensorFlow, and PyTorch (e.g., "NN", "Neural Net", "neural nets", "neural networks", "nn"). The alias can be different abbreviations of the full names or different morphological variants. We need to identify these aliases and resolve them to canonical forms before we can reliably extract explanation sentences and relationships of domain concepts. Third, the relevance of a term to a specific domain cannot be reliably determined by simple lexicon heuristics or term frequency metrics. For example, some lexical heuristics may assume that acronyms are glossary terms, but many acronyms such as "USE", "SUPPORT", and "NOTE" in deep learning domain are not. Term frequency metrics are not reliable either because some frequent noun phrases are irrelevant to the target domain, while some relevant concepts are only mentioned a few times. The lexical and semantic context of the sentences in which

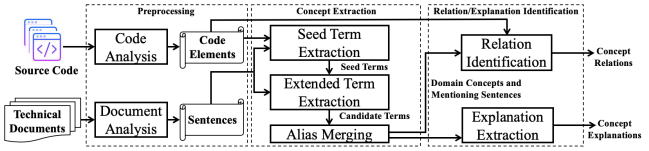

Figure 1: Approach Overview

domain-specific concepts are mentioned must be considered. Moreover, source code can be used as an auxiliary resource, for example, many class names and package names represent the domain concepts.

3 APPROACH

An overview of the approach is presented in Figure 1. It includes three phases, i.e., preprocessing, concept extraction, and relation/explanation identification.

The purpose of preprocessing is to extract useful information from the input corpus and prepare the required data for the subsequent phases. It includes two steps, i.e., code analysis and document analysis. Code analysis parses the source code and extracts code elements (e.g., packages and classes) and their relations (e.g., containment, inheritance, and aggregation) from the code. Document analysis parses the documents and extracts sentences from them.

The purpose of concept extraction is to extract domain concepts from source code and documents. It includes three steps, i.e., seed term extraction, extended term extraction, and alias merging. Seed term extraction uses heuristics to identify a set of initial seed terms from the sentences with the aid of code elements. By treating the seed terms as the initial labelled data, extended term extraction uses a semi-supervised learning method to iteratively identify more candidate terms from the sentences. Note that these candidate terms may be the aliases of the same concepts. Therefore, alias merging identifies and merges concept aliases (e.g., morphological synonyms and abbreviations) that are extracted from different sentences.

The purpose of relation and explanation identification is to provide relations and explanations for the extracted domain concepts. It includes two steps, i.e., relation identification and explanation extraction. Relation identification identifies "is a", "has a", and "related to" relations between concepts with the aid of code elements. Explanation extraction selects a set of explanation sentences for each concept.

We implement our approach in Python and the natural language processing tool used in our implementation is spaCy [17]. The remaining of the section details the steps of the approach and corresponding implementation. All the examples used in this section are from the deep learning domain and its projects Deeplearning4j, TensorFlow, and PyTorch.

3.1 Preprocessing

Our implementation includes code analyzers for Java and Python, which are implemented based on javalang [9] (a Java parser implemented in Python) and the build-in module *ast* of Python respectively. The code analyzers extract the packages/modules, classes, and various relations between them from the source code.

The technical documents of many projects are web pages that are available online. Therefore, we implement a crawler based on Scrapy [16] to obtain the documents. We use BeautifulSoup [1] (a Python parser for HTML and XML files) to parse the obtained web pages. We then clean the obtained web page content by handling different kinds of special HTML elements (for example replacing code fragments with "__CODE__", replacing tables with "__TABLE__") and recovering the sentences that are broken by tags like "<p>" and "<li>". Finally we extract the plain text from the content and split the text into sentences based on punctuations.

3.2 Seed Term Extraction

Seed term extraction automatically identifies a small set of terms with high confidence using heuristics. To find an optimal set of heuristics, we try different combinations of heuristics (e.g., extracting all-caps words in text and local variable/parameter names in code as seed terms) and adjust the heuristics based on the results. Finally we choose the following two heuristic rules that can be used to extract high-quality terms from source code and documents.

1. Acronym Appears with Its Full Name in Document

If an acronym appears together with its full name in a document, the acronym is likely a term. This rule is embodied by the following three sentence patterns.

1) [full name] ([acronym]), for example "By design, the output of a recurrent neural network (RNN) depends on arbitrarily distant inputs.";

2) [acronym] ([full name]), for example "Feed forward neural networks are comprised of dense layers, while recurrent neural networks can include Graves LSTM (long short-term memory) layers.".

3) [acronym]: [full name], for example "Enumeration used to select the type of regression statistics to optimize on, with the various regression score functions - MSE: mean squared error - MAE: mean absolute error - RMSE: root mean squared error - RSE: relative squared error - CorrCoeff: correlation coefficient.".

To enforce this rule, we first match a sentence with the above three patterns and then identify the phrase for the full name based on the acronym. For example, based on the acronym "RNN' we can identify the three-word phrase "recurrent neural network" before the bracket as its full name. For each matched acronym both its full name and itself are recognized as seed terms.

2. Package/Class Name Appears in Document

If the name of a package or class appears as ordinary text (i.e., not code elements) in a document, the name is likely a term. For example, in the sentence "During NN training, each X iterations, executors will send encoded dense updates with lower threshold" from Deeplearning4j, the term "NN" is the name of the package *org.deeplearning4j.nn*. The term "NN" here means neural network, thus can be recognized as a seed term. Considering finer-grained code elements such as parameters and local variables may produce vast terms, but most of them are false ones. To ensure the quality of seed terms we only consider course-grained code elements (i.e., package and class) in seed term extraction.

To enforce this rule, we obtain the names of all the packages and classes in the code elements and split them by camel case; we then search for the split words or phrases in all the sentences (case insensitive) and treat all the matched ones as seed terms.

Seed terms extracted by the above two rules may be ordinary phrases that happen to be used as a package/class name or general technical terms that are not specific to the target domain. For example, in the sentence "Computes the aggregate score as a sum of all of

the individual scores of each of the labels against each of the outputs of the network" from Deeplearning4j, the phrase "a sum" happens to be the name of the class *org.nd4j.linalg.api.ops.impl.accum.ASum*. But obviously the phrase is not a term. Examples of general technical terms include those related to file or string operations. These ordinary phrases and general technical terms need to be eliminated to ensure the quality of the extracted seed terms. To this end, we use the following three rules to further eliminate irrelevant terms. The seed terms that satisfy any condition are eliminated.

- the term only appears in the documents of one project;
- the term includes a single word and the word is a common word included in WordNet [20];
- the term includes a stop word at the beginning or end.

The stop words we use in our implementation are the English stop words defined by NLTK [12] (a natural language toolkit) plus some special stop words related to programs (including "get", "set", "return", "test", "util").

3.3 Extended Term Extraction

Extended term extraction uses machine learning to identify more terms from the sentences. It treats the term identification as a sequence tagging task [46], which is to predict the corresponding tag sequence of an observation sequence. There are many NLP problems that are solved as sequence annotations, such as part of speech tagging (POS), chunking, and named entity recognition (NER). The tagging scheme we use for this task is the IOBES scheme [46], which is widely used for sequence tagging tasks. In the schema, "B" ("Beginning"), "I" ("Inside"), and "E" ("End") respectively indicate that the current token is the beginning, middle, and end of a term; "S" ("Single") indicates that the current token itself constitutes a term; and "O" ("Outside") indicates a normal token. For example, the tagging of a sentence that includes two terms (i.e., "recurrent neural network" and "RNN') is presented below.

"O: By O: design O: , O: the O: output O: of O: a B: recurrent I: neural E: network O: (S: RNN O:) O: depends O: on O: arbitrarily O: distant O: inputs O: ."

Our machine learning model for term identification is the widely used LSTM-CRF model [36], which combines LSTM (Long-Short Term Memory) networks and a CRF (Conditional Random Fields) layer. It is an end-to-end learning model that does not require handcrafted features. We use the implementation of the LSTM-CRF model provided by Guillaume et al. [11]. To train the model we generate a sequence pair (an input token sequence and its corresponding tag sequence) from each sentence in the labelled data and use these sequence pairs as the training data. When used for term identification (i.e., prediction), the model takes as input a token sequence generated from an input sentence and produces as output a tag sequence. The words that are tagged with "S" and the phrases that are tagged with "B", "I", and "E" in the output tag sequence are identified as glossary terms.

In both the training and prediction we use pretrained word vectors to represent the tokens in the input token sequence. The word vectors are pretrained by fine tuning the GloVe [43] word vectors. Our implementation is based on glove.840B.300d [5], which has 840 billion tokens, 2.2 million cased vocabulary, and 300 vector dimensions. And the word2vec implementation we use is gensim [4].

We use a semi-supervised method to train the model and use an iterative process to identify more terms. The process starts with the seed terms by treating the sentences that include the seed terms as the labelled data and all the other sentences as unlabelled data. After each iteration, a term identification model is trained and used to identify more terms from the sentences. Similar to seed term extraction, this iterative term extraction process also uses the three rules (i.e., term appearing in only one project, common word in WordNet, or including stop words) to filter out irrelevant terms. All the unlabelled sentences with newly identified terms are then added into the labelled data to start the next iteration. This filtering strategy ensures the quality of the sentences that are iteratively added into the labelled data.

The whole iterative process ends when any of the following conditions is met: 1) no new terms are identified by the trained model; 2) the number of iterations reaches a predefined limit (10 in our implementation); 3) there are no unlabelled sentences. In the last iteration all the newly identified terms are accepted except those including stop words at the beginning or end of their names.

For example, in this step "Activation Layer" is identified as a term from the sentence "Imports an Activation layer from Keras." from Deeplearning4j.

To ensure the quality of the extracted terms, we further refine the results of seed and extended term extraction to produce the candidate terms. First, eliminate general technical terms. We use the following equation to estimate the generality of a term, where $freq_g(t)$ ($freq_d(t)$) and $TN_g(t)$ ($TN_d(t)$) are the frequency and occurrence number of a term t respectively and N_g (N_d) is the token number in the general technical corpus (domain corpus). We eliminate the terms whose generality is higher than a threshold (0.2 in our implementation), which is chosen based on trial term refinement using different thresholds. The general technical corpus used in our implementation includes the reference documentations of JDK 8 [10] and Python 3.6 [13]. For example, "ACM", "App", and "number values" are eliminated using this rule.

$$Generality(t) = \frac{freq_g(t)}{freq_d(t)} = \frac{\frac{TN_g(t)}{N_g}}{\frac{TN_d(t)}{N_d}} \quad (1)$$

Second, for each term that includes an acronym, add the acronym into candidate terms if it is not included. For example, "AIS data" is a candidate term, thus "AIS" (which means automatic identification system) is also added as a candidate term.

3.4 Alias Merging

The purpose of this step is to identify the aliases of the same concept from the candidate terms and merge them together to form a concept. This step handles two kinds of concept aliases that widely exist in technical documents, i.e., morphological synonym and abbreviation. We first lemmatize all the candidate terms, then identify and merge morphological synonyms, and finally identify and merge abbreviations.

3.4.1 Identification and Merging of Morphological Synonyms. Typical morphological synonyms in technical documents include:

1) words or phrases that are only different in cases or singular/plural forms, e.g., "Deeplearning4J" and "Deeplearning4j", "RNN" and "RNNs";

2) words or phrases that have different spelling or misspelling, e.g., "Deeplearning4J" and "Deeplearinng4J";

3) words or phrases that use hyphens in different ways, e.g., "t-SNE" and "tSNE".

The first type of morphological synonyms can be directly identified after lemmatization. For the other two types, we use edit distance to determine whether two words or phrases are morphological synonyms. In particular, we use Damerau-Levenshtein distance (DL distance) [28] to measure the similarity of two words or phrases. The DL distance is the minimum number of operations (insert, delete, or substitute a single character, or transpose two adjacent characters) required to convert one word or phrase into the other. Considering words or phrases of different lengths we also calculate the relative distance between two words or phrases s_1 and s_2 as the following equation, where $Distance(s_1, s_2)$ is the DL distance of s_1 and s_2, $length(s)$ is the length of a string s.

$$RDistance(s_1, s_2) = \frac{Distance(s_1, s_2)}{max(length(s_1), length(s_2))} \quad (2)$$

For example, "Deeplearinng4J" can be converted into "Deeplearning4J" by a transposition operation of "i" and "n", thus the DL distance of these two words is 1; similarly the DL distance of "RNN" and "NN" is also 1. These two pairs of words have the same DL distance, but the relative distance of "RNN" and "NN" (1/3) is much bigger than that of "Deeplearinng4J" and "Deeplearning4J" (1/14). For two words or phrases, if both of their DL distance and relative DL distance are lower than predefined thresholds (3, 1/4 respectively in our implementation), we regard them as a pair of morphological synonyms. The thresholds are chosen based on trail selection of morphological synonyms: we first fix the DL distance threshold to 2 and experimentally search for an optimal threshold for relative DL distance; we then search for an optimal threshold for DL distance by fixing the relative DL distance threshold.

Based on the identified morphological synonyms, we construct alias sets of domain concepts by calculating transitive closures based on morphological synonyms, i.e., iteratively merging morphological synonyms together. For any candidate term that has no morphological synonyms, we construct an alias set that includes only the term itself.

3.4.2 Identification and Merging of Abbreviation Relationships. Typical abbreviation relationships in technical documents include:

1) a word is the acronym of a phrase, e.g., "NN" and "neural network";

2) a word is the prefix of another one, e.g., "net" and "network";

3) two phrases satisfy one of the above two relationships after eliminating their common head or tail words, e.g., "neural net" and "neural network".

Any two words or phrases that satisfy one of the above three conditions are treated as a candidate abbreviation relationship. Thus we can further merge the alias sets based on the identified candidate abbreviation relationships. A difficulty here is that a candidate term may have abbreviation relationships with multiple other candidate terms, but it is usually an abbreviation of only one other candidate term. For example, "RNN" is a candidate abbreviation for both "Recurrent Neural Network" and "Recursive Neural Network", but it is usually the abbreviation of the former in deep learning.

Table 1: Hearst Patterns

C_1 such as C_2	C_2 "is a" C_1
such C_1 as C_2	C_2 "is a" C_1
C_2[,] or other C_1	C_2 "is a" C_1
C_2[,] and other C_1	C_2 "is a" C_1
C_1[,] including C_2	C_2 "is a" C_1
C_1[,] especially C_2	C_2 "is a" C_1

Therefore, we need to determine only one full name for an abbreviation in alias merging. For an alias set AS that has candidate abbreviation relationships with a set of other alias sets $ASSet$, we determine an only alias set in $ASSet$ as the full name of AS based on context similarity in the following way. We calculate the context similarity between AS and each alias set in $ASSet$ as the cosine similarity of their vectors. The vector of an alias set is obtained by averaging the vectors of all the sentences that include a candidate term from the alias set, while the vector of a sentence is obtained by averaging the pretrained vectors of all the words in the sentence. Based on the context similarities, we select an alias set from $ASSet$ that has the highest context similarity with AS as the full name of AS.

Based on the finally determined abbreviation relationships, we iteratively merge the alias sets that have abbreviation relationships together. After alias merging, each of the alias set is treated as a domain concept with all the candidate terms in the set as its aliases. For example, we can obtain the following domain concepts and their alias: "RNN" ("RNNs", "Recurrent Neural Networks", "Rnn", "recurrent neural network"", "recurrent neural networks", "rnn"); "neural network" ("NN", "neural net", "Neural Net", "Neural Network", "Neural Networks", "Neural net", "Neural network", "Neural networks", "neural nets", "neural networks").

3.5 Relation Identification

Relations between the extracted domain concepts are often implied in specific sentence patterns in the documents and structural relations between code elements. Based on the document and code analysis, we identify "is a", "has a", and "related to" relations between the extracted concepts. For two concepts C_1 and C_2, we identify "is a" and "has a" relations between them from the following aspects.

First, identify "is a" relations from the sentences that mention C_1 and C_2 based on Hearst Patterns [33], which are widely used in the automatic acquisition of hyponymy lexical relations from unrestricted text. The patterns used in our current implementation are shown in Table 1, where C_1 and C_2 can be any alias of the corresponding concepts.

Second, identify "is a" and "has a" relations based on the structural relations between packages and classes. For two packages or classes E_1 and E_2 whose names are the aliases of C_1 and C_2 respectively:

1) if E_1 contains E_2 or aggregates E_2 as a property, add a "has a" relation from C_1 to C_2;

2) if E_1 inherits E_2, add an "is a" relation from C_1 to C_2.

Third, identify "is a" and "has a" relations based on prefix/suffix relations between concept aliases:

1) if an alias of C_1 is the prefix of an alias of C_2, add a "has a" relation from C_1 to C_2;

2) if an alias of C_1 is the suffix of an alias of C_2, add an "is a" relation from C_2 to C_1;

To provide richer concept relations, we further identify "related to" relations between the extracted concepts, which are weaker than "is a" and "has a" relations. The identification of "related to" relations is based on the similarity between two concepts: if the similarity between two concepts is higher than a predefined threshold, add a bidirectional "related to" relation between them. The context similarity is calculated as the following equation, which combines the lexical similarity (Sim_{lex}) and the context similarity ($Sim_{context}$) of two concepts (w_1 and w_2 are two weights satisfying $w_1 + w_2 = 1$).

$$Sim(c_1, c_2) = w_1 \times Sim_{lex}(c_1, c_2) + w_2 \times Sim_{context}(c_1, c_2) \quad (3)$$

The lexical similarity of c_1 and c_2 is the Jaccard similarity [34] between the token sets of their aliases (i.e., $Token(c_1)$ and $Token(c_2)$).

$$Sim_{lex}(c_1, c_2) = \frac{|Token(c_1) \cap Token(c_2)|}{|Token(c_1) \cup Token(c_2)|} \quad (4)$$

The context similarity of c_1 and c_2 is calculated in a similar way to the context similarity used in alias merging: generate a vector for each concept by averaging the vectors of all the sentences that mention the concept, then calculate the cosine similarity between concept vectors. In our implementation, the similarity threshold is set to 0.5, and w_1 and w_2 are set to 0.25 and 0.75 respectively. The threshold and weights are determined based on trail identification of "related to" relations: we first fix the threshold to 0.5 and experimentally search for an optimal value for w_1 ($w_2 = 1 - w_1$); we then search for an optimal threshold by fixing w_1.

Some examples of concept relations extracted from different aspects are as follows.

- {SGD, is a, optimizer} extracted from the sentence "torch.optim: Contains optimizers such as SGD." based on Hearst Patterns;
- {NN, has a, Activation Layer} extracted based on the package containment relation between *org.deeplearning4j.nn* and *org.deeplearning4j.nn.layers.ActivationLayer*;
- {RNN, has a, RNN Layer} and {recurrent neural network, is a, neural network} extracted based on the prefix/suffix relations between concept names;
- {L1, related to, Regularizer} extracted based on context similarity of concepts.

Note that some "related to" relations may be "is a" or "has a" relations. For example, L1 is a common method for regularization to avoid overfitting in machine learning, but only a "related to" relation is extracted for them because there lack supports from sentences and code elements for the "is a" relation between them. There may be more than one relation between two concepts. For example, there are both "is a" and "has a" relations between "RNN" and "neural network".

3.6 Explanation Extraction

Concept extraction provides for each extracted domain concept a list of sentences that mention the concept. The purpose of explanation extraction is to select for each domain concept a set of sentences that can help users to understand the concept. These sentences usually can be classified into the following categories.

- **Concept Definition:** provide definition for a concept and related techniques, for example taxonomy, explanation of the meaning;
- **Technical Comment:** comment on the characteristics of a concept and related techniques, for example benefits and drawbacks, comparisons with other techniques;
- **Usage Guidance:** suggest the proper way to use a concept and related techniques, for example applicable scenarios, guidance on related settings, common problem solutions.

We need to filter out sentences that mention a concept but are useless for the understanding of the concept. These sentences may be low-quality sentences that are incomplete or sentences talking about low-level implementation (e.g., using the concepts to explain the functionalities of code elements). We treat all the sentences that mention a domain concept as the candidate sentences for the concept and use the following heuristic rules to filter out useless sentences. Candidate sentences that satisfy any of the rules are filtered out.

- **Incomplete Sentences**: the sentence has no subject or predicate, or has incomplete punctuations (e.g., the right parenthesis is missing);
- **Code Elements**: the sentence includes code elements;
- **Question**: the sentence is a question;
- **Subordinate Clause**: the sentence mentions the concept in its subordinate clauses.

For example, for the concept "Activation Layer" the following two sentences are selected as its explanation sentences: 1) "Activation layer is a simple layer that applies the specified activation function to the input activations."; 2) "Activation Layer Used to apply activation on input and corresponding derivative on epsilon.". While the following two sentences are filtered out: 1) "Imports an Activation layer from Keras."; 2) "Advanced Activation Layers.".

4 EMPIRICAL STUDY

We conduct an empirical study with two technical domains (deep learning and Hadoop). Based on the results, we evaluate the effectiveness of our approach for domain concept extraction and the usefulness of the extracted domain concepts for software development tasks. All the data and results of the study have been included in our replication package [15].

4.1 Study Design

The two subject domains that we choose for the study represent two kinds of technical domains. Deep learning domain includes software libraries that are written in different languages and provide similar functionalities for the development of deep learning applications. Hadoop domain is a software ecosystem that consists of interdependent software systems.

The subject projects we choose for the deep learning domain are three popular deep learning libraries: Deeplearning4j 1.0.0-beta3 [2], Tensorflow 1.12 [18], PyTorch 1.0.0 [14]. Among them, Deeplearning4j is written in Java, Tensorflow and PyTorch are written in Python.

The subject projects we choose for the Hadoop domain are Hadoop 2.9.2 [6], HBase 2.1.0 [7], Hive 2.3.4 [8]. Hadoop is a distributed data storage and processing framework based on the MapReduce model. HBase is a distributed database that runs on top of HDFS (Hadoop Distributed File System) and provides Bigtable-like capabilities for Hadoop. Hive provides a SQL-like interface to query

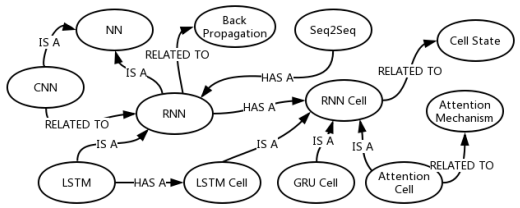

Figure 2: Concepts and Relations Extracted for Deep Learning Domain

data stored in various databases and file systems that integrate with Hadoop. All these three projects are written in Java.

All these six projects are open source. We crawl the source code of these projects and the documents available at their official websites. The documents include official project introductions, user guides/manuals, tutorials, API references. For each of the two domains, we use our approach to extract domain concepts, concept relations, and explanation sentences.

Based on the results, we design a series of experiments to answer the following research questions.

- **RQ1**: How accurate are the extracted domain concepts, relations, and explanations? Can the approach outperform existing approaches for domain glossary extraction?
- **RQ2**: How does the extracted domain glossary fuse the knowledge from different projects and documents? How does the extracted concepts complement general knowledge base such as WikiPedia?
- **RQ3**: Can the extracted domain glossary help developers to obtain the required knowledge?

4.2 Basic Results

For the deep learning domain, we identify 471 seed terms and 6,645 additional terms in extended term extraction. These candidate terms are finally transferred into 5,382 concepts after alias merging. These concepts have 7,116 aliases in total and each concept has 1 to 22 aliases (1.32 on average). For these concepts 16,962 relations are extracted, including 119 "is a" relations, 709 "has a" relations, 16,134 "related to" relations, and each concept has 1 to 115 relations (6.30 on average). For these concepts 1,689 explanation sentences are extracted and each concept has 0 to 69 sentences (0.31 on average).

For the Hadoop domain, we identify 202 seed terms and 2,537 additional terms in extended term extraction. These candidate terms are finally transferred into 2,069 concepts after alias merging. These concepts have 2,739 aliases in total and each concept has 1 to 12 aliases (1.32 on average). For these concepts 6,815 relations are extracted, including 135 "is a" relations, 479 "has a" relations, 6,201 "related to" relations, and each concept has 1 to 150 relations (6.59 on average). For these concepts 1,311 explanation sentences are extracted and each concept has 0 to 68 sentences (0.63 on average).

A snippet of the concepts and relations extracted for deep learning domain is shown in Figure 2. This snippet explains the relationships between a set of deep learning concepts such as NN, RNN, CNN, LSTM, LSTM Cell, RNN Cell, Seq2Seq. The aliases and explanation sentences of a part of the concepts are shown in Table 2.

Table 2: Aliases and Explanations of Concepts in Figure 2

Concept	Alias	Explanation
NN	Neural Network, Neural Net	Neural networks can find complex relationships between features and the label.
RNN	Recurrent Neural Network	Recurrent neural networks (RNN's) are used when the input is sequential in nature.
CNN	Convolutional Neural Network	In practice, convolutional neural networks (CNN's) are better suited for image classification tasks.
Seq2Seq	–	Sequence to Sequence network, or seq2seq network, or Encoder Decoder network, is a model consisting of two RNNs called the encoder and decoder.
Back Propagation	Backpropagation, Backprop	Backpropagation will compute the gradients automatically for us.
LSTM	Long Short-Term Memory	A LSTM is well suited for this type of problem due to the sequential nature of the data.
RNN Cell	–	A fused RNN cell represents the entire RNN expanded over the time dimension.

4.3 Accuracy (RQ1)

To answer the question, we evaluate the accuracy of the key steps of our approach (i.e., term extraction, alias merging, relation identification, explanation extraction) separately for each of the two domains. As the numbers of terms, concepts, relations, and sentences are large, we adopt a sampling method [48] to ensure that the estimated population is in a certain confidence interval at a certain confidence level. According to the sampling method the minimum number of data instances to be examined can be calculated by the formula $MIN = n_0/(1 + (n_0 - 1)/populationsize)$. The variable n_0 in the formula can be calculated by the formula $n_0 = (Z^2 \times 0.25)/e^2$, where Z is the z-score of the desired confidence level and e is the desired error margin. For each step, we randomly select MIN samples for the error margin $e = 0.05$ at 95% confidence level: for term extraction we examine the accuracy of candidate terms, i.e., whether the sampled terms represent domain specific concepts; for alias merging we examine the accuracy of alias relationships, i.e., whether the two aliases in the sampled alias pairs represent the same concepts; for relation identification we examine the accuracy of concept relations, i.e., whether the two concepts in the sampled relations have the corresponding relations; for explanation extraction we examine the accuracy of explanation sentences, i.e., whether the sampled sentences provide useful explanations or guidance for the corresponding concepts. We invite three master students who are experts in the two domains (with 3 years' experience) to examine the accuracy. For each examination two experts independently make the decision based on both their own knowledge and online resources of the domains. When there is disagreement on a decision, the third expert gives an additional judgement.

Then we compare our approach with a state-of-the-art approach for documentation based domain glossary extraction by Arora et al. [22]. Arora et al.'s approach extracts glossary terms and their related terms (i.e., the terms that belong to the same categories), but does not identify concept aliases or relations. Therefore, we can only compare the accuracy of the extraction of glossary terms with the approach. For each target domain we use the same sampling method to randomly select MIN sentences from the corpus of technical documents and ask the experts to annotate the glossary terms that are included in these sentences. For each sentence two experts independently identify the glossary terms in it and their agreement on each identified term is checked. When there is disagreement on an identified term, the third expert gives an additional judgement. These identified terms are used as the golden set. We implement

Table 3: Accuracy of Key Steps

Step	Deep Learning		Hadoop	
	Agreement	Accuracy	Agreement	Accuracy
Term Extraction	0.773	0.747	0.755	0.771
Alias Merging	0.943	0.961	0.940	0.956
Relation Identification	0.807	0.753	0.698	0.828
Explanation Extraction	0.729	0.807	0.831	0.802

Arora et al.'s approach and use the implementation to extract glossary terms from the corpus of each domain. We then identify the extracted terms that are included in the sampled sentences and treat these terms as the result set of the approach. The result set of our approach is produced in a similar way. Based on the result sets of the two approaches and the golden set, we calculate the precision, recall, and F1-measure (the harmonic mean of precision and recall) of glossary term extraction for the two approaches.

In the above process the agreement rate is always calculated between the first two experts before resolving their disagreements together with the third one.

According to the sampling method, we select 384 samples for the accuracy evaluation of each step in each domain. The results of the evaluation are shown in Table 3. For each step in each domain we provide the agreement rate of the human annotations and the accuracy.

It can be seen that our approach achieves high accuracy in term extraction (0.747 and 0.771), alias merging (0.961 and 0.956), relation identification (0.753 and 0.828), and explanation extraction (0.807 and 0.802). For term extraction, the falsely extracted terms include: non-concept phrases, e.g., "network learn" (deep learning) and "table exists (Hadoop); concepts that are not relevant to the domain, e.g., "Google Cloud" (deep learning), "Webapp" (Hadoop); concepts with meaningless qualifiers, e.g., "second tensors" (deep learning), "given rows" (Hadoop). For alias merging, the falsely identified alias relationships include: similar terms with different meanings, e.g., "opencl" and "opencv" (deep learning); falsely merged abbreviation and full name, e.g., "RR" and "random row" (Hadoop, the right full name is "Record Reader"). For relation identification, the falsely extracted concept relations include: false relations of false concepts, e.g., {contrib, has a, Early Stopping} (deep learning, "contrib" is not a term); false "related to" relations between lexically similar terms, e.g., {specified nodes, related to, specified metric} (Hadoop, the two terms are lexically similar but not relevant). For explanation extraction, the falsely extracted sentences do not provide useful explanations or guidance, e.g., the sentence "The reparameterized sample therefore becomes differentiable." for "reparameterized sample" (deep learning).

According to the sampling method, we select 384 sentences from each domain for the comparison with the approach in [22]. The agreement rates of the human annotations are 0.776 and 0.805 in the deep learning domain and the Hadoop domain respectively. The results of the comparison are shown in Table 4. For each approach we provide the precision, recall, and F1-measure of the extraction of glossary terms in each domain.

It can be seen that in deep learning domain our approach significantly outperforms Arora et al.'s approach in terms of both precision (by 0.498) and recall (by 0.117). Their approach is based on the identification of noun phrases, thus may extract many irrelevant words or phrases, especially acronyms, e,g., "specified value", "AND" and "IT" in deep learning domain, and "Hadoop example

Table 4: Comparison with Arora et al.'s Approach [22]

Method	Deep Learning			Hadoop		
	Precision	Recall	F1	Precision	Recall	F1
Our Approach	0.698	0.668	0.683	0.642	0.326	0.432
Arora et al.'s Approach [22]	0.200	0.551	0.293	0.210	0.606	0.312

code", "ONLY", "OR" in Hadoop domain. In Hadoop domain our approach outperforms Arora et al.'s approach in terms of precision (by 0.432) and F1-measure (by 0.120), but is inferior to their approach in term of recall (by 0.280). We find that the reason why our approach achieves a much lower recall in Hadoop lies in the difference between the two domains. In our study, deep learning domain consists of three alternative libraries with similar functionalities and these projects share many glossary terms; while Hadoop domain consists of three interdependent software systems that provide different functionalities and these projects share much less glossary terms. Our approach relies on the rule of shared terms in different projects to identify the training data for term extraction, so it can identify much less terms for training in Hadoop domain and thus extract much less glossary terms finally. To conclude, our approach outperforms Arora et al.'s approach in general; and it performs better when the target domain consists of projects that provide similar functionalities.

4.4 Knowledge Fusion (RQ2)

Our approach harvests glossary terms, concepts, and explanations from different projects and documents. To evaluate how our approach fuses the knowledge from different projects and documents, we analyze the distribution of the extracted terms, concepts, and explanations among different projects and documents. Table 5 shows the statistics for the knowledge fusion of different projects, i.e., how much of the terms, concepts, and explanation sentences are covered by different documents of different projects. For deep learning domain, if one reads a single document of a specific project she can learn at most 58.0% terms, 53.7% concepts, and 29.5% explanation sentences. For Hadoop domain, the ratios are 45.7%, 40.9%, and 40.6% respectively. From the analysis, it can be seen that it is necessary to fuse the knowledge from different documents of different projects to have a comprehensive understanding of domain specific concepts. For example, the abbreviation "GAN" appears in the documents of TensorFlow, but its full name "Generative Adversarial Networks" can only be found in Deeplearning4j and its explanations can only be found in PyTorch.

We further analyze the complementarity with Wikipedia based on the 384 terms sampled in each domain for the evaluation of term extraction (see Section 4.3). For each term we manually examine Wikipedia to check whether it is included with the same meaning. Note that as Wikipedia may include the same terms with different meanings this examination can only be performed manually. Table 6 shows the results of the complementarity analysis, including the number of glossary terms that are confirmed, the number of glossary terms that are included in Wikipedia and the ratio. It can be seen that only 23.0% and 28.0% of the glossary terms extracted by our approach are included in Wikipedia, indicating a high complementarity with Wikipedia. Wikipedia includes some domain specific terms, e.g., "RNN" and "LSTM" in deep learning domain, but misses more domain specific terms. For example, our approach identifies "computation graph" as a term in deep learning domain and provides useful explanations, but it is not included

Table 5: Knowledge Fusion of Different Documents and Projects

	Deep Learning							Hadoop						
	Total	Deeplearning4j		TensorFlow		PyTorch		Total	Hadoop		Hbase		Hive	
		API Ref.	Guide	API Ref.	Guide	API Ref.	Guide		API Ref.	Guide	API Ref.	Guide	API Ref.	Guide
Term	7116	40.9%	26.3%	58.0%	29.1%	27.0%	30.6%	2739	32.7%	45.7%	34.5%	39.0%	33.8%	40.5%
Concept	5382	35.3%	21.1%	53.7%	24.1%	21.9%	25.5%	2069	28.2%	40.9%	30.4%	33.6%	29.5%	33.6%
Explanation	1689	16.2%	16.9%	29.5%	17.7%	8.2%	11.4%	1311	10.2%	40.6%	9.0%	16.9%	9.2%	14.1%

Table 6: Complementarity with Wikipedia

Domain	#Term	#In Wekipedia	Ratio
Deep Learning	287	66	23.0%
Hadoop	296	83	28.0%

in Wikipedia. Therefore, the concepts, relations, and explanations extracted by our approach can well complement general knowledge base such as WikiPedia.

4.5 Usefulness (RQ3)

We design an experiment to investigate whether the extracted domain glossary can help to answer real-world developer queries. Given a query, we use a document search engine to search the corpus of documents of the target domain. Then we perform the document search again using query expansion based on the extracted domain glossary and compare the performance before and after the query expansion.

In particular, we use Elasticsearch 6.0.6 [3] as the document search engine. The query expansion in implemented in the following way. We identify the domain concepts that are mentioned in the query. Assume C is the set of concepts that are mentioned in the initial query and T is the set of the other tokens in the initial query, then the expanded query consists of: 1) all the tokens in T, their weights are set to 0.5; 2) all the concepts that have "is a" or "has a" relations with any concept in C, their weights are set to 0.5; 3) the aliases of the concepts in C (each concept with one alias), their weights are set to 1. We try different combinations of aliases for the concepts in C and for each combination the expanded query is used to search the corpus of documents of the target domain with Elasticsearch. According to the suggestion in [31], we replace the aliases of up to three concepts each time (while the other concepts use the initial terms) to avoid combinatorial explosion and the distortion of results. Each time we collect the top 10 ranked sentences and record the score (given by Elasticsearch) of each sentence. After all the combinations for the query are executed we sum up the scores of each sentence to produce the final ranking of sentences.

We select 12 questions from the top 100 voted Stack Overflow questions with the tag "deep learning" or "Hadoop". These questions are related to domain concepts and can be answered by the documents. The selected sentences are shown in Table 7, among which Q1-Q8 are about deep learning and Q9-Q12 are about Hadoop. For each question we use its title as a query and calculate three metrics for the query results: precision (P), which indicates the proportion of relevant sentences in the top 10 ranked sentences; average precision (AP), the mean of the precision scores obtained after all the relevant results are retrieved (with relevant results that are not retrieved receiving a precision score of 0); reciprocal rank (RR), which is the reciprocal of the position of the first correct result.

Table 8 shows the results of the evaluation of query expansion, which compare the performance before and after query expansion for each question. It can be seen that supported with the domain glossary the query expansion significantly improves the performance of document searching for both deep learning and Hadoop.

Table 7: Developer Questions Selected from Stack Overflow

Question	ID	Title
Q1	17837871	How to set layer-wise learning rate in Tensorflow ?
Q2	38714959	Understanding Keras LSTMs
Q3	35687678	Using a pre-trained word embedding (word2vec or Glove) in TensorFlow
Q4	44747343	Keras input explanation: input_shape, units, batch_size, dim, etc
Q5	4151128	What are the differences between numpy arrays and matrices? Which one should I use?
Q6	5751114	Nearest neighbors in high-dimensional data ?
Q7	13610074	Is there a rule-of-thumb for how to divide a dataset into training and validation sets ?
Q8	2620343	What is machine learning ?
Q9	17837871	How to copy file from HDFS to the local file system
Q10	42735154	What is hadoop namenode command used for
Q11	28146411	What is the benefit of using CDH (cloudera)?
Q12	46684091	In hadoop,what is the difference and relationship between jobtracker tasktracker?

Table 8: Performance of Query Expansion

Question	Domain	Before Extension			After Extension		
		P	AP	RR	P	AP	RR
Q1	Deep Learning	0.80	0.80	1.00	0.90	0.88	1.00
Q2		0.30	0.51	1.00	0.70	0.83	1.00
Q3		0.50	0.70	1.00	0.70	0.69	0.50
Q4		0.80	0.78	1.00	0.60	0.57	0.30
Q5		0.30	0.46	0.50	0.60	0.49	0.25
Q6		0.80	0.78	1.00	0.90	1.00	1.00
Q7		0.30	0.24	0.20	0.50	0.73	1.00
Q8		0.40	0.35	0.33	0.50	0.93	1.00
Avg.		0.53	0.58	0.75	0.68	0.76	0.76
Q9	Hadoop	0.80	0.94	1.00	0.50	0.62	1.00
Q10		0.20	0.23	0.25	0.50	0.61	1.00
Q11		0.10	0.33	0.33	0.50	0.70	1.00
Q12		0.30	0.22	0.14	0.70	0.84	1.00
Avg.		0.35	0.43	0.43	0.55	0.69	1.00

The query expansion performs well for Q2, Q7, Q8, Q10, Q11, and Q12. We can see these queries are quite relevant to the comprehension of concepts (e.g., training, validation sets). In contrast, the query expansion is not so effective for Q3 and Q9. Both these two queries ask concrete technical solutions with the corresponding software libraries and systems (e.g., TensorFlow). One reason for this difference may be that concrete technical solutions are more relevant to specific projects (e.g., TensorFlow) and basic techniques (e.g., file copy) rather than the overall domain and high-level concepts. This inspires us to consider to integrate the domain glossaries with more general background knowledge bases (e.g., those for computing and Java programming) to provide better support for the acquisition of software knowledge.

4.6 Threats to Validity

A major threat to the internal validity is the subjective judgements involved in the data annotation and result assessment of the experiments. Although the final results are based on the agreement of different experts, some of the judgements may be incorrect and thus influence the evaluation results.

A major threat to the external validity is the limited number of target domains, projects, and documents used in the study. Different domains, projects, and documents may have different characteristics in their source code and documents (e.g., styles and formats) and the involved concepts (e.g., naming styles and usage modes in

code/text). The approach may be less effective for other domains or when considering more projects and documents of the domains. Another major threat to the external validity is the limited number of questions used in the usefulness evaluation. Different questions may involve different concepts and thus lead to different effects of improvement.

5 DISCUSSION

Researchers have advocated for a new vision for satisfying the information needs of developers, which is called On-Demand Developer Documentation (OD3) [47]. An OD3 system would automatically generate high-quality response for a user query based on a combination of knowledge extracted from a collection of artifacts such as source code, documentation, and Q&A posts. This capability needs to be built around the extraction and fusion of software development knowledge from a variety of artifacts. And a domain knowledge graph that provides domain specific concepts and their relationships is a key for the knowledge extraction and fusion. Actually we have seen the application of domain knowledge graph for the purpose of dynamic documentation generation [42] and improving API Caveats Accessibility [37].

The domain glossary constructed by our approach can be seen as an initial attempt for the construction of domain knowledge graph. Besides technical domains like deep learning and Hadoop, our approach can also be applied on business domains like e-learning and office given the required source code and documentation. The domain glossary extracted by our approach complement general knowledge graphs such as Wikidata [19] by providing domain specific concepts and relationships. It is useful that the domain glossaries of specific domains are integrated with general knowledge graphs (e.g., Wikidata) and the domain glossaries of general technical domains (e.g., Java and Python) to provide more comprehensive knowledge graphs for software engineering tasks.

Our approach can be improved from several aspects. First, other artifacts such as Q&A posts, issues, and commit messages can be added into the data sources of concept extraction. Second, more rules and algorithms can be used for the identification of seed terms. Third, the extracted relationships can be further refined by employing more advanced text processing and knowledge extraction techniques. This refinement means not only identifying more "is a" and "has a" relations, but also refining the general "related to" relations into concrete relations. Fourth, the extracted domain specific concepts can be connected with more general concepts (e.g., concepts about programming languages and computing) in background knowledge by various relationships.

6 RELATED WORK

Term extraction is a long studied topic in many domains such as knowledge engineering and information retrieval. Existing approaches use linguistic, statistical, and hybrid techniques to identify glossary terms from text corpus [23, 24, 35, 38, 41]. In software engineering area, researchers investigate automatic extraction of domain glossaries from requirements documentations to mitigate imprecision and ambiguity [22]. Anurag et al. [30] propose a hybrid approach for the extraction of glossary terms from natural language requirements. Menard and Ratte [40] propose a text mining approach to extract domain-relevant terms from internal business documents. Arora et al. [22] propose a linguistic approach to extract glossary terms and their related terms from requirements documents. These approaches rely on linguistic heuristics or term-frequency-based statistics to identify domain specific terms, thus the accuracy is often low. Moreover, these approach do not extract concept aliases or relations.

Some researches identify and extract software specific entities such as API elements and programming actions from software engineering text corpus. The researches on API recognition identify API elements mentioned in text corpus [53, 58]. Based on the recognition of API elements, we can further link APIs with other relevant artifacts. Treude and Robillard [51] propose an approach for automatically augmenting API documentation with insight sentences from Stack Overflow. Dagenais and Robillard [27] propose an approach for recovering traceability links between an API and its learning resources. Treude et al. [52] developed a technique for automatically extracting tasks from software documentation to help developers navigate documentation. The software specific entities extracted by these approaches complement the glossary terms extracted by our approach.

Some researches [21, 25] extract domain concepts and relations from source code. These approaches can only identify concepts from source code identifiers and cannot harvest knowledge from software documentation.

Recently, some researchers investigate learning based approaches for the identification of software specific concepts and relations from text content. Ye et al. [59] propose a learning based named entity recognition approach on software engineering social content (e.g., Stack Overflow). Zhao et al. [60] propose a learning based approach for discovering domain specific concepts and their relation triples from Stack Overflow posts. Chen et al. [26] propose an automatic approach for extracting software-specific terms and commonly used morphological forms from Stack Overflow posts. These approaches are customized for Stack Overflow and rely on Stack Overflow specific features such as post tags and timestamps. Moreover, Ye et al.'s approach [59] and Zhao et al.'s approach [60] rely on human labelled data for learning.

7 CONCLUSION

In this paper, we have proposed a learning-based approach for automatic construction of domain glossary from source code and software documentation. We have conducted an empirical study with the deep learning domain and the Hadoop domain. The study confirms the accuracy of the extracted domain glossary and its usefulness for the fusion and acquisition of knowledge from different documents of different projects. In the future, we plan to improve the approach by refining the extracted relationships and integrating with general knowledge graphs and apply the constructed domain glossaries to more software engineering tasks.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No. 2018YFB1004803.

REFERENCES

[1] 2019. *BeautifulSoup.* Retrieved February 20, 2019 from https://www.crummy.com/software/BeautifulSoup
[2] 2019. *Deeplearning4j.* Retrieved February 20, 2019 from http://deeplearning4j.org
[3] 2019. *Elasticsearch.* Retrieved February 20, 2019 from https://www.elastic.co
[4] 2019. *gensim.* Retrieved February 20, 2019 from https://radimrehurek.com/gensim
[5] 2019. *GloVe.* Retrieved February 20, 2019 from https://nlp.stanford.edu/projects/glove
[6] 2019. *Hadoop.* Retrieved February 20, 2019 from https://hadoop.apache.org
[7] 2019. *HBase.* Retrieved February 20, 2019 from https://github.com/apache/hbase
[8] 2019. *Hive.* Retrieved February 20, 2019 from https://github.com/apache/hive
[9] 2019. *javalang.* Retrieved February 20, 2019 from https://github.com/c2nes/javalang
[10] 2019. *JDK 8.* Retrieved February 20, 2019 from https://docs.oracle.com/javase/8
[11] 2019. *LSTM-CRF.* Retrieved February 20, 2019 from https://github.com/guillaumegenthial/tf_ner
[12] 2019. *NLTK.* Retrieved February 20, 2019 from http://www.nltk.org
[13] 2019. *Python 3.6.* Retrieved February 20, 2019 from https://docs.python.org/3.6
[14] 2019. *PyTorch.* Retrieved February 20, 2019 from https://pytorch.org
[15] 2019. *Replication Package.* Retrieved June 17, 2019 from https://fudanselab.github.io/Research-ESEC-FSE2019-DomainGlossary/
[16] 2019. *Scrapy.* Retrieved February 20, 2019 from https://scrapy.org
[17] 2019. *spaCy.* Retrieved February 20, 2019 from https://spacy.io/
[18] 2019. *Tensorflow.* Retrieved February 20, 2019 from https://www.tensorflow.org
[19] 2019. *Wikidata.* Retrieved February 20, 2019 from https://www.wikidata.org
[20] 2019. *WordNet.* Retrieved February 20, 2019 from https://wordnet.princeton.edu
[21] Surafel Lemma Abebe and Paolo Tonella. 2015. Extraction of domain concepts from the source code. *Sci. Comput. Program.* 98 (2015), 680–706.
[22] Chetan Arora, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. 2017. Automated Extraction and Clustering of Requirements Glossary Terms. *IEEE Trans. Software Eng.* 43, 10 (2017), 918–945.
[23] Sophie Aubin and Thierry Hamon. 2006. Improving Term Extraction with Terminological Resources. In *Advances in Natural Language Processing, 5th International Conference on NLP, FinTAL 2006, Turku, Finland, August 23-25, 2006, Proceedings.* 380–387.
[24] Didier Bourigault. 1992. Surface Grammatical Analysis For The Extraction Of Terminological Noun Phrases. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992.* 977–981.
[25] Nuno Ramos Carvalho, José João Almeida, Pedro Rangel Henriques, and Maria João Varanda Pereira. 2015. From source code identifiers to natural language terms. *Journal of Systems and Software* 100 (2015), 117–128.
[26] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017.* 450–461.
[27] Barthélémy Dagenais and Martin P. Robillard. 2012. Recovering traceability links between an API and its learning resources. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland.* 47–57.
[28] Fred Damerau. 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM* 7, 3 (1964), 171–176.
[29] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95.
[30] Anurag Dwarakanath, Roshni R. Ramnani, and Shubhashis Sengupta. 2013. Automatic extraction of glossary terms from natural language requirements. In *21st IEEE International Requirements Engineering Conference, RE 2013, Rio de Janeiro-RJ, Brazil, July 15-19, 2013.* 314–319.
[31] Fan Fang, Bo-Wen Zhang, and Xu-Cheng Yin. 2018. Semantic Sequential Query Expansion for Biomedical Article Search. *IEEE Access* 6 (2018), 45448–45457.
[32] Jin Guo, Marek Gibiec, and Jane Cleland-Huang. 2017. Tackling the term-mismatch problem in automated trace retrieval. *Empirical Software Engineering* 22, 3 (2017), 1103–1142.
[33] Marti A. Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992.* 539–545.
[34] Paul Jaccard. 1901. Distribution de la Flore Alpine dans le Bassin des Dranses et dans quelques régions voisines. *Bulletin de la Societe Vaudoise des Sciences Naturelles* 37 (1901), 241–72.
[35] Leslie P. Jones, Edward W. Gassie Jr., and Sridhar Radhakrishnan. 1990. INDEX: The statistical basis for an automatic conceptual phrase-indexing system. *JASIS* 41, 2 (1990), 87–97.
[36] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016.* 260–270.
[37] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. 2018. Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018.* 183–193.
[38] Yutaka Matsuo and Mitsuru Ishizuka. 2004. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools* 13, 1 (2004), 157–169.
[39] Collin McMillan, Denys Poshyvanyk, Mark Grechanik, Qing Xie, and Chen Fu. 2013. Portfolio: Searching for relevant functions and their usages in millions of lines of code. *ACM Trans. Softw. Eng. Methodol.* 22, 4 (2013), 37:1–37:30.
[40] Pierre André Ménard and Sylvie Ratté. 2016. Concept extraction from business documents for software engineering projects. *Autom. Softw. Eng.* 23, 4 (2016), 649–686.
[41] Maria Teresa Pazienza, Marco Pennacchiotti, and Fabio Massimo Zanzotto. 2005. *Terminology Extraction: An Analysis of Linguistic and Statistical Approaches.* Knowledge Mining, S. Sirmakessis, Ed. Berlin, Germany: Springer. 255—-279 pages.
[42] Xin Peng, Yifan Zhao, Mingwei Liu, Fengyi Zhang, Yang Liu, Xin Wang, and Zhenchang Xing. 2018. Automatic Generation of API Documentations for Open-Source Projects. In *IEEE Third International Workshop on Dynamic Software Documentation, DySDoc@ICSME 2018, Madrid, Spain, September 25, 2018.* 7–8.
[43] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL.* 1532–1543.
[44] Mohammad Masudur Rahman, Chanchal Kumar Roy, and David Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1.* 349–359.
[45] Michael Rath, Jacob Rendall, Jin L. C. Guo, Jane Cleland-Huang, and Patrick Mäder. 2018. Traceability in the wild: automatically augmenting incomplete trace links. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018.* 834–845.
[46] Lev-Arie Ratinov and Dan Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning, CoNLL 2009, Boulder, Colorado, USA, June 4-5, 2009.* 147–155.
[47] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil A. Ernst, Marco Aurélio Gerosa, Michael W. Godfrey, Michele Lanza, Mario Linares Vásquez, Gail C. Murphy, Laura Moreno, David C. Shepherd, and Edmund Wong. 2017. On-demand Developer Documentation. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017.* 479–483.
[48] Ravindra Singh and Naurang Singh Mangat. 2013. *Elements of survey sampling.* Vol. 15. Springer Science & Business Media.
[49] Ferdian Thung, Shaowei Wang, David Lo, and Julia L. Lawall. 2013. Automatic recommendation of API methods from feature requests. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013.* 290–300.
[50] Yuan Tian, Ferdian Thung, Abhishek Sharma, and David Lo. 2017. APIBot: question answering bot for API documentation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017.* 153–158.
[51] Christoph Treude and Martin P. Robillard. 2016. Augmenting API documentation with insights from stack overflow. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016.* 392–403.
[52] Christoph Treude, Martin P. Robillard, and Barthélémy Dagenais. 2015. Extracting Development Tasks to Navigate Software Documentation. *IEEE Trans. Software Eng.* 41, 6 (2015), 565–581.
[53] Gias Uddin and Martin P. Robillard. 2017. Resolving API Mentions in Informal Documents. *CoRR* abs/1709.02396 (2017). arXiv:1709.02396
[54] Jinshui Wang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. 2013. Improving feature location practice with multi-faceted interactive exploration. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013.* 762–771.
[55] Wentao Wang, Arushi Gupta, Nan Niu, Li Da Xu, Jing-Ru C. Cheng, and Zhendong Niu. 2018. Automatically Tracing Dependability Requirements via Term-Based Relevance Feedback. *IEEE Trans. Industrial Informatics* 14, 1 (2018), 342–349.
[56] Wentao Wang, Nan Niu, Hui Liu, and Zhendong Niu. 2018. Enhancing Automated Requirements Traceability by Resolving Polysemy. In *26th IEEE International Requirements Engineering Conference, RE 2018, Banff, AB, Canada, August 20-24, 2018.* 40–51.
[57] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: automated generation of answer summary to developersź technical questions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017.* 706–716.

[58] Deheng Ye, Lingfeng Bao, Zhenchang Xing, and Shang-Wei Lin. 2018. APIReal: an API recognition and linking approach for online developer forums. *Empirical Software Engineering* 23, 6 (2018), 3129–3160.

[59] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. 2016. Software-Specific Named Entity Recognition in Software Engineering Social Content. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. 90–101.

[60] Xuejiao Zhao, Zhenchang Xing, Muhammad Ashad Kabir, Naoya Sawada, Jing Li, and Shang-Wei Lin. 2017. HDSKG: Harvesting domain specific knowledge graph from content of webpages. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*. 56–67.