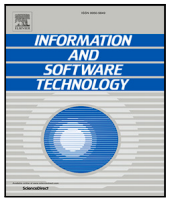
Information and Software Technology 143 (2022) 106770

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Automated data function extraction from textual requirements by leveraging semi-supervised CRF and language model

Mingyang Li [a,b], Lin Shi [a,b,*], Yawen Wang [a,b], Junjie Wang [a,b], Qing Wang [a,b,c], Jun Hu [a,b], Xinhua Peng [d], Weimin Liao [d], Guizhen Pi [d]

[a] Laboratory for Internet Software Technologies, Institute of Software, Beijing, China
[b] University of Chinese Academy of Sciences, Beijing, China
[c] State Key Laboratory of Computer Science, Institute of Software, Beijing, China
[d] China Merchants Bank, Shenzhen, China

ARTICLE INFO

Keywords:
Function point analysis
Size estimation
Conditional random field
Bootstrapping
Language model

ABSTRACT

Context: Function Point Analysis (FPA) provides an objective, comparative measure for size estimation in the early stage of software development. When practicing FPA, analysts typically abide by the following steps: data function (DF) extraction, transactional function extraction, function type classification and adjustment factor determination. However, due to lack of approach and tool support, these steps are usually conduct by human efforts in practice. Related approaches can hardly be applied in the FPA due to the following three challenges, i.e., FPA rule-driven extraction, domain-specific parsing, and expensive labeled resources.

Objective: In this paper, we aim to automate the extraction of DFs, which is the starting and fundamental step in FPA.

Method: We propose an automated approach named DEX to extract data functions from textual requirements. Specifically, DEX introduces the popularly-used conditional random field (CRF) model to predict the boundary of a data function. Besides, DEX employs the bootstrapping-based algorithm and DF-oriented language model to further boost the performance.

Results: We evaluate DEX from two aspects: evaluation on a real industrial dataset and a manual review by domain experts. The evaluation on the real industrial dataset shows that DEX could achieve 80% precision, 84% recall, and 82% F1, and outperforms three state-of-the-art baselines. The expert review suggests that DEX could increase 16% precision and 13% recall, compared with those produced by engineers.

Conclusion: DEX could achieve promising results under a small number of labeled requirements and outperform the state-of-the-art approaches. Moreover, DEX could help engineers produce more accurate and complete DFs in the industrial environment.

1. Introduction

In industrial practice, it is desirable to have a reliable size estimation method before software systems are built, so that the software development activities can be well planned and quantitatively managed [1]. One of the most intensively used approaches in the early stage is Function Point Analysis (FPA) that has been well developed over 30 years. FPA provides a measure of software size in function point that is defined as "*a synthetic metric that is comprised of the weighted totals of the inputs, outputs, inquiries, logical files or user data groups, and interfaces belonging to an application*" [2]. With the counted function points, reasonable size estimations can be measured, and indirect size estimations can be calculated [3]. Based on the FPA method, many measurement method standards, such as the IFPUG (ISO/IEC 20926), MKii(ISO/IEC 20968), NESMA (ISO/IEC 24570), COSMIC (ISO/IEC 19761) and FiSMA (ISO/IEC 29881), have been presented to support the software size estimation.

When practicing FPA, one of the most important steps is extracting "function" from requirements. Unfortunately, FPA has its unsolved problems that prevent it from being a valid measurement method with widespread acceptance. One significant problem is that functions need to be extracted from requirements documents. These documents are often totaling hundreds of pages, and the process heavily relies on human measurement to read those documents [4]. Another problem is that the estimation results are subjective to some degree. Analyzing and extracting functions follow a set of standard FPA rules, which are open

* Corresponding author.
E-mail address: shilin@iscas.ac.cn (L. Shi).

https://doi.org/10.1016/j.infsof.2021.106770
Received 2 December 2020; Received in revised form 28 October 2021; Accepted 2 November 2021
Available online 18 November 2021

to interpretation on many occasions, thus they often produce inconsistent estimation results [5]. The inconsistent estimation results may occur even in the same organization. Low and Jeffery [6] reported that 30% variance of estimation results was caused within one organization, and more than 30% variance was caused across organizations. Therefore, it is important to propose approaches to promote the automation, effectiveness, and quality of function point analysis. On the one hand, the automated function extraction approach can reduce the amount of manual work. On the other hand, automation approach could help preserve the extracted functions to be objective and unbiased [5].

A function in FPA is the smallest unit of activity that is meaningful to the users. According to the IFPUG standard, functions are classified into two categories, i.e., the data function (DF) and the transactional function. Specifically, the DFs represent the functionality provided to the user to meet internal and external data storage requirements [1,7]. They are described in requirements in the form of noun phrases. The transaction functions represent the functionality provided to the users to input and retrieve data from the application. They are typically described as "verb + DF" in requirements. In industrial practice, function extraction typically begins with the DFs, and then transactional functions by analyzing the operators acting on the DFs. Therefore, the extraction of DFs is the fundamental step, and mistakenly extracting them will lead to greater bias than the transactional ones [8]. Moreover, the concerns of extracting DFs are different from the transaction functions (e.g., "breaking elementary process" issue and "meaningless to users" issue when extracting transactional functions [5]). In this paper, we focus on the DF extraction.[1]

Requirements are typically written in natural language. Due to the complexity and ambiguity of natural language, there exist few studies to automatically extract DFs from textual requirements. The most related work is the approach proposed by Adem et al. [4] which leverages pre-defined rules to extract functions from requirements. However, it needs requirements in the format of a goal-scenario model [9] rather than free-format natural language. In addition, given that the DFs are noun phrases, there also exist studies in other fields, such as general entity extraction [10–12] and glossary term extraction [13,14]. However, these studies in the literature could not be effectively applied to extract DFs from textual requirements due to the following two aspects.

FPA rule-driven extraction. According to the FPA rules, DFs are derived from the functionalities provided to the user to meet internal and external data storage requirements. They are the business objects that are stored and maintained by the software systems [1]. In such cases, the entities or glossary terms extracted from previous studies [10–14] normally go beyond the scope of DFs. Take the requirement "I would like to receive the credit card bill when I open the mobile application in IOS12.4" as an example. Fig. 1 shows the four entities extracted by a popularly-used entity extractor, Stanford CoreNLP.[2] Among the four entities, only the "credit card bill" is the DF, since the other three entities describe general concepts that will not be stored and maintained by the software systems based on the requirement description. In the FPA, each DF will be calculated as a certain number for function points. If we consider the four general entities extracted by the existing methods as the DFs for the size estimation, it would lead to over-estimated size [8]. Thus, it is desirable to design novel solutions to accurately identify DFs from requirements.

Expensive labeling resources. There exist a plethora of machine learning-based approaches to address the general term extraction problem [10,11] which could be potentially applicable to the software requirements. However, these approaches rely on a large volume of labeled resources to train a promising model. It is expensive to offer sufficient labeled data since it requires FPA experts to read and label

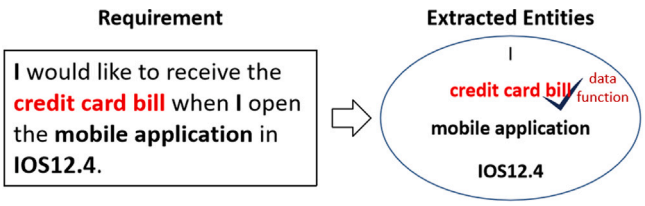


Fig. 1. The entities extracted by Stanford CoreNLP.

requirements containing enormous textual contents. With inadequate labeled data, the existing machine learning-based approaches would lead to inefficient models. Apart from the labeled data, there exists a large number of unlabeled data, and how to utilize the unlabeled requirements is valuable to explore.

In FPA, analysts typically starts with the DF extraction. Then, analysts recognize the operations on the DFs to produce transactional functions, and classify each function into specific type. In the end, analysts determine the value adjustment factor to produce the final size estimation [2]. However, due to lack of effective automated tool support, these steps are usually conduct by human efforts in practice. In this study, we focus on the starting and fundamental step, DF extraction, in FPA. We propose an automated **D**ata Function **EX**traction approach (named DEX) to extract DFs from textual requirements. Specifically, DEX takes the DF extraction as the sequence tagging task, and introduces Conditional Random Field (CRF) [15], which could consider the contextual information. The basic assumption is that DFs normally appear together with the indicative contexts. For example, there is a requirement "I would like to receive the credit card bill when I open the mobile application in IOS12.4", and corresponding DF is "credit card bill" which could be easier to extract with foregoing verb "receive" and following word "when". Not limited to the example, the contextual information usually provides clues for DF extraction. Motivated by this observation, DEX extracts the contextual features for each word, and employs CRF to capture the regularities from the training data for the DF extraction. Besides, to solve the problem of expensive labeling resources, DEX further adopts a semi-supervised technique and DF-oriented language model that leverages unlabeled data to boost the performance. DEX consists of four steps: (1) CRF instance building where DEX builds seeds and unlabeled instances from labeled requirements and unlabeled requirements; (2) Bootstrapping based CRF training where DEX trains the CRF model using bootstrapping based algorithm; (3) DF-oriented language model training using historical DFs; (4) DF extraction where DEX extracts DFs for new-coming requirements.

We evaluate DEX on 3586 requirements from 20 on-going software systems in a *China Merchants Bank (CMB)*, one of the largest joint-stock commercial banks in China, and compare DEX with three state-of-the-art baseline approaches. The results show that DEX could achieve 80% precision, 84% recall, and 82% F1, and significantly outperform the baseline approaches. Moreover, evaluation results also show that, with the help of semi-supervised technique and DF-oriented language model, unlabeled data can improve the extraction performance by 2% precision, 26% recall, and 16% F1. This further proves the usefulness of unlabeled data for DF extraction. Besides, we conduct an expert review to investigate the usefulness of DEX in a real-world application scenario. The results illustrate that DEX could achieve 82% precision and 83% recall on 11 projects, and outperform the manual extraction by engineers.

The major contributions of this paper are as follows:

- We propose a novel approach DEX to automatically extract DFs from textual requirements.
- We provide an effective way to train the DF extraction model with a small amount of labeled data and unlabeled domain-specific data, which can motivate other tasks with few labeled data.

[1] The extraction of transactional functions is discussed in Section 7.1.

[2] https://stanfordnlp.github.io/Core.

- We evaluate our approach on an industrial dataset and a real-world application scenario, and the results are promising.
- We make all the source code publicly accessible,[3] which could be applied to other industrial settings and related tasks.

The remainders of the paper are organized as follows: Section 2 introduces the background. Section 3 introduces the related work and their limitations. Section 4 elaborates the approach. Section 5 presents the experiment design. Section 6 describes the results. Section 7 discusses how to apply DEX to transactional function extraction, learned lessons, and threats to validity. Section 8 concludes our work.

2. Background

In this section, we introduce the nature of the function and the techniques used in the proposed approach.

2.1. The nature of function

FPA is firstly defined by Allan Albrech in 1979, which derived a functional size of product value distinct and disassociated from lines of code, technology, or software language [1]. In FPA, the requirements are modeled as a set of functions where each function maps to end-user business functionality. Once a function is identified, it will be categorized into a function type and assigned a specific number of function points according to different FPA standards. The number of function points is finally regarded as the size estimation.

The need for maintaining FPA counting practices led to founding the IFPUG,[4] which maintains the counting practices manual, provides guidelines and examples, and oversees the standardization of the measurement method. The standard maintained by IFPUG is the first-generation functional size measurements and has been an ISO standard (ISO/IEC 20926). In the IFPUG standard, the requirements are modeled as two types of functions: (1) DFs and transactional functions. A DF represents the functionality provided to the user to meet internal and external data storage requirements. The DF is a user-identifiable noun phrase within the requirements. A transaction function represents the functionality provided to the user to input and retrieve data from the application. They are the unique and user-recognizable elementary processes that satisfy functional user requirements and are expressed as the "verb + DF" in requirements. For example, given a DF "customer information", the corresponding transaction functions would typically be "add customer information", "delete customer information", "modify customer information" and "query customer information".

Given the temporal order when analyzing functions, DF extraction is the fundamental step. Besides, compared with transaction functions, DFs will be calculated as more function points. For example, in the IFPUG standard, the ILF and EIF are calculated as 10 and 7 function points respectively, while the EI, EO, EQ are calculated as 4, 5, 4 function points respectively [2]. Therefore, mistake extraction of DFs will lead to greater bias than the transactional ones [8].

Currently, DFs highly rely on manual extraction by experts, which is quite time-consuming and labor-intensive. Moreover, the manually extracted DFs are often biased by different people and vary considerably in estimation quality. Therefore, an automated solution to extract DFs is badly desired.

2.2. Technical background

We introduce two main techniques used in the proposed approach in the following.

2.2.1. Sequence tagging and conditional random field

Sequence tagging is a widely-used technique that is to predict the tag sequence given the input sequence [16]. Let $X = (x_1, x_2, x_3, x_4, \ldots, x_n)$ be the input sequence where x_i is the element in the input sequence X. It aims at finding the most optimal label sequence $Y = (y_1, y_2, y_3, y_4 ..., y_n)$ given X where y_i is the label given to corresponding x_i. The sequence tagging task could be solved with many machine learning algorithms such as Maximum Entropy Model [17], Hidden Markov Model [18], and CRF [15].

Specifically, CRF is a class of sequence tagging methods often applied in many pattern recognition and machine learning tasks. CRF is a type of discriminative undirected probabilistic graphical model. For the sequential data, whereas the typical discrete classifiers that predict a label for a single sample without considering "neighboring" information, CRF takes the input as the ordered sequence and predicts each label considering the previous elements. Thus, CRF is more suitable for tasks with abundant contextual information. Considering that natural language is a typical sequence of words, CRF has been popularly applied to many natural language processing tasks such as part-of-speech tagging [19], text chunking [20] and named entity recognition [21].

2.2.2. Semi-supervised learning and bootstrapping

Semi-supervised learning is a category of machine learning techniques that makes use of not only labeled data but also unlabeled data for training. It is an intermediate type of machine learning technique that is between supervised learning and unsupervised learning. Compared with supervised learning, it needs far less labeled data that is expensive to collect. Compared with unsupervised learning, it does not require the data following specific apriori assumptions, which enlarges its application scenarios. Due to its advantages in lower-level of human effort and higher accuracy, semi-supervised learning has gained great interests in both theory and practice [22].

Bootstrapping (also known as self-training or self-teaching) is a commonly used technique of semi-supervised learning [23]. In bootstrapping, a classifier is firstly trained with a small amount of labeled data. The classifier is then used to classify the unlabeled data. The most confident unlabeled instances, together with their predicted labels, are added to the training set. The classifier is re-trained, and the procedure is repeated until reaching promising results. Bootstrapping is a wrapper algorithm that could combine with different machine learning approaches depending on the specific tasks.

3. Related work

In this section, we firstly introduce the existing approaches for automated function point analysis. Considering that the function extraction is similar to the term extraction in the requirements engineering and natural language processing (NLP) fields, we further introduce the automated approaches for term extraction.

3.1. Function point analysis

Due to the significant challenges in understanding natural language requirements, few studies have addressed the automated extraction of functions from textual requirements. Adem et al. [4] proposed a rule-based approach to extract function points from requirements that are written in the goal and scenario model, e.g. "verb + target + direction + way", however, applying this method requires modeling free-format requirements into the goal-scenario model, which might be more time-consuming than the standard counting procedure. Thus, it is not practical for the software industry with a large volume of projects. To overcome that issue, we propose a learning-based approach that can automatically extract transaction functions from free-format requirements text. Shi et al. [5] proposed an approach to automatically extract transactional functions from textual requirements. It firstly

[3] https://github.com/iscaslmy/DEX.
[4] https://www.ifpug.org/.

trains a machine learning model to predict whether a word appears in the transactional functions, and then constructs transactional functions using the pre-defined rules. Differently, it focuses on the extraction of the transactional functions rather than the data functions in our study.

Moreover, there are several studies for the automation of FPA from design specifications or source code. Pow-Sang et al. [24] identified function point logic files from UML class diagrams that made use of association, composition, generalization, and association-class relationships. Irawati et al. [25] provided mapping rules between function point calculation and design documentation by referring to the information of Use Case Diagram and Class Diagram and associations between them. Edagawa and Akaike et al. [26] proposed a method to automatically identify data and transaction functions from Web applications using static code analysis. Ali SAG and Tarhan [27] derived UML sequence Diagrams from functional execution traces at runtime with the help of AspectJ technology and utilized it to measure the functional size. These studies for FPA mainly targeted design specifications and source code artifacts. Our study aims to automatically extract data functions from textual requirements, which can facilitate the automatic estimation of the system's size in the early stage.

3.2. Term extraction

According to the techniques used in the approaches, we divide them into two categories, i.e., linguistics based approaches and learning based approaches. Specifically, linguistics based approaches implement term extraction with text parsing (e.g., part-of-speech parsing and text chunking) and linguistic rules. learning based approaches aim to train models using machine learning algorithms and utilize the models for future predictions. In the following, we introduce the existing studies and their limitations of the two categories of approaches.

3.2.1. Linguistics based approaches

Typically, linguistics based approaches follow the following pipeline. First, they parsed texts leveraging NLP techniques such as part-of-speech parsing and text chunking. Second, they extract the target terms using the pre-defined linguistic rules, e.g., part-of-speech patterns and noun chunking. To our best knowledge, all the previous studies for requirements belong to this group. Bourigault et al. [28] described an approach for terminological noun phrases. With the part-of-speech tags returned by the NLP toolkit, it is implemented by regular expressions to extract certain combinations of noun phrases. Dwarakanath et al. [29] used linguistic rules to identify process nouns, abstract nouns, and auxiliary verbs from natural language requirements. It firstly parses part-of-speech tags to extract noun phrases and verb phrases and then builds the linguistic rules to handle co-ordinating conjunctions, adjectival modifiers, and nominalizations. Ménard et al. [30] proposed an approach to extract domain-specific concepts from business documents. It is implemented as a pipeline, including a candidate generation step based on part-of-speech patterns, several filtering rules to filter out the irrelevant terms by heuristics. Johann et al. [31] proposed an approach to extract key noun phrases and verb phrases that are related to features from app descriptions and app reviews. It first obtains part-of-speech tags using the NLP toolkit. Then it predefined 18 part-of-speech patterns to extract phrases. Arora et al. [13] developed an approach for extracting glossary terms and their related terms from requirements documents. It firstly extracts the candidate noun phrases via text chunking with the help of a popularly-used NLP toolkit NLTK. Then, three linguistic heuristics are utilized to refine the candidates.

When applying these approaches for data function extraction, there are mainly two limitations. First, linguistics based approaches rely on NLP toolkits to parse the requirements. Unfortunately, the NLP toolkits are typically trained on the general corpus. They produce errors when parsing some data functions that are named according to the business domains. The errors produced by NLP toolkits will accumulate into the linguistic based approaches, which makes negative impacts on the performance. Second, the pre-defined linguistic rules are not suitable for data function extraction. For example, the terms extracted by existing studies [29,31] refer to all general concepts contained in a textual document. For example, glossary terms, defined as "salient terms in documents" [13], aim to help stakeholders get familiar with business knowledge the technical terminologies in a domain. It is observed that the pre-defined rules are not for data functions, which results in extracting general terms loosely related to the data functions. Besides, it is expensive to rebuild rules for data functions whose extraction relies on contextual information. Rather than the pipelined approach, DEX directly trains a model to extract data functions, which could alleviate the first limitation. For the second limitation, DEX trains the semi-supervised CRF model using a small amount of labeled data to extract candidate DFs, which could avoid the overload of manually building expert rules. Moreover, DEX leverages the DF-oriented language model to filter out the irrelevant candidates, which could improve the precision of the DF extraction.

3.2.2. Learning based approaches

Learning based approaches adopt the machine learning algorithm to extract target information from texts, have been widely used for many general entity extraction tasks. The popularly used method for general entity extraction is supervised learning. For example, Finkel et al. [32] trained a CRF model for named entity recognition (NER) with 10 handcrafted features. Lample et al. [10] presented a widely used LSTM-CRF model for NER, which aims at extracting target information in terms of sequence tagging and eliminates the need for feature engineering. Chiu et al. [11] presented a novel neural network architecture to extract named entities from the general corpus. It detects word-level and character-level features using a hybrid bidirectional LSTM and CNN architecture.

With the help of supervised learning, general entity extraction has achieved promising performance under the condition of sufficient labeled data. However, supervised learning requires data labeling which is a labor-intensive activity, especially in the domains requiring extensive expertise. Thus, a growing number of researches focused on boosting performance by unlabeled data. Huang et al. [33] proposed an approach to extracted domain-specific concepts based on a semi-supervised CRF model. Using a small number of labeled data as seeds, it iteratively expands training samples from unlabeled data by bootstrapping. It is included in our study as the baseline approach. Sachan et al. [34] investigated how to use unlabeled text data to improve the performance of NER models. Specifically, they trained a bidirectional language model (BiLM) on unlabeled data and transfer its weights to pre-train a NER model with the same architecture as the BiLM, which results in a better parameter initialization of the NER model. Huang et al. [33] proposed an approach to extracted domain-specific concepts based on a semi-supervised CRF model. Using a small number of labeled data as seeds, it iteratively expands training samples from unlabeled data by bootstrapping. It is also included as the baseline approach. Sachan et al. [34] investigated how to use unlabeled text data to improve the performance of NER. Specifically, they trained a bidirectional language model (BiLM) on unlabeled data and transfer its weights to pre-train a NER model with the same architecture as the BiLM, which results in a better parameter initialization of the NER model.

Different from our study, these approaches aim at extracting entities from general corpus, and there exists a large volume of labeled and unlabeled data to train the model. When coming to the FPA context, the industrial requirements are ubiquitous difficult to access, and expensive to label. Our approach aims to train a promising machine learning model using limited requirements and labeling resources in the FPA context.

The most related work is our previous study [12] which proposed an approach RENE to extract entities from requirements. RENE leverages

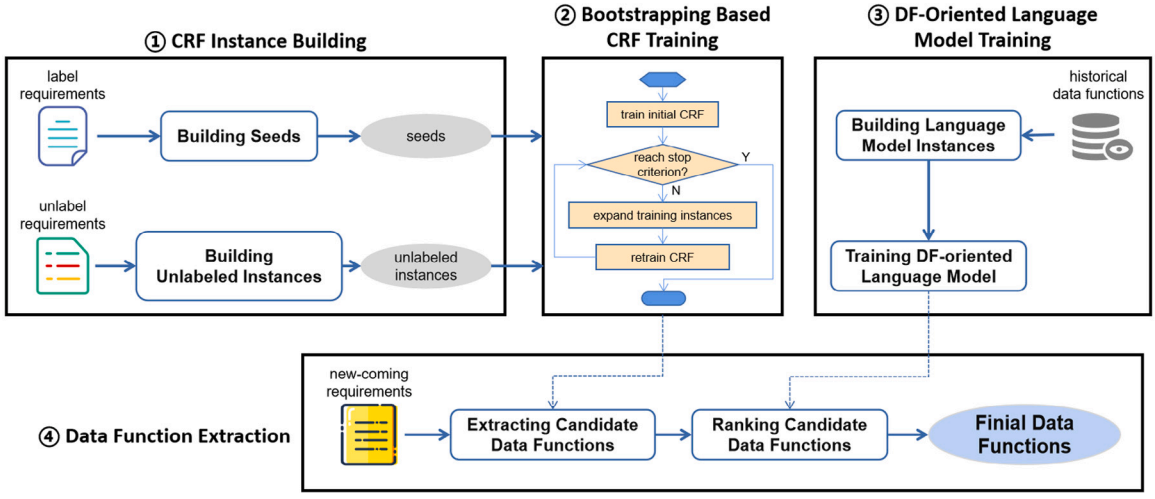


Fig. 2. The approach overview.

the LSTM-CRF network for entity extraction and utilizes a transfer learning framework to train the LSTM-CRF network to overcome the challenge of limited labeled data. Specifically, RENE pre-trains the embedding and LSTM layers in the LSTM-CRF model to capture the general knowledge using Wikipedia articles. Then, RENE fine-tunes the embedding and LSTM layers to adapt to the specific requirements domain using domain-specific unlabeled requirements. Different from our previous study, DEX adopts a relatively lightweight approach for data function extraction. We consider that DEX trains the model faster, which could adapt to application scenarios that need to frequently update the model.

4. Approach

We propose an automated approach DEX to extract DFs from textual requirements. In general, DEX considers DF extraction as the sequence tagging problem and trains the CRF model to avoid manually building expert rules. Besides, DEX uses a bootstrapping based algorithm to train the CRF model to solve the problem of limited labeling resources. Then, using the historical DFs, DEX trains a DF-oriented language model to rank the candidate DFs extracted by the CRF model. Fig. 2 illustrates the overview of DEX. It consists of four steps: (1) CRF instance building where DEX builds seeds and unlabeled instances from labeled requirements and unlabeled requirements; (2) Bootstrapping based CRF training where DEX trains the CRF model using bootstrapping based algorithm; (3) DF-oriented language model training using historical DFs; (4) DF extraction where DEX extracts DFs for new-coming requirements. The following introduces the details of each step.

4.1. CRF instance building

In this step, DEX builds the training instances for the CRF model. There are two sub-steps, i.e., building seeds from the labeled requirements and building unlabeled instances from unlabeled requirements.

Building seeds. Given a labeled requirement R_l, DEX first splits the textual contents into sentences $[s_1, s_2, \ldots, s_n]$ by period. For each sentence s, DEX tokenize it into a word sequence $s = [w_1, w_2, \ldots, w_m]$ using the NLP toolkit Stanford CoreNLP,[5] where w_i is the word in the sentence. Please note that DEX does not remove the stop words from sentences because we consider that they could provide information for predicting the locations of DFs.[6] Due to that DEX aims to predict the position of a DF, DEX extracts the features for each word in the word sequence. We have mentioned that contextual information is helpful for DF extraction. Therefore DEX extracts not only information of the current word but also its forward and backward words in the sentence. For each word, the feature vector captures a context window of n words to its forward and n words to its backward. For example, the following sub-sequence $[w_{i-n}, w_{i-(n-1)}, \ldots, w_{i-1}, w_i, w_{i+1}, \ldots, w_{i+(n-1)}, w_{i+n}]$ is the context window for current word w_i, where w_{i-1} is its forward 1st word, and w_{i+1} is its backward 1st word. Please note that the context window size n is the hyper-parameter that could be tuned by users. The default value of context window size n is 4 in DEX. For each word within the window, DEX extracts the following three features:

- *Lexical Feature*. The word itself. There might be common words appearing before and after DFs that are beneficial for DF extraction.
- *Part-of-Speech Feature*. The part-of-speech of the word. Since the DFs are noun phrases in the requirements, part-of-speech tagging results can contribute to DF extraction.
- *TF-IDF Feature*. The TF-IDF [35] of the word. TF-IDF could indicate the importance of the word and might have an impact on the probability of its appearance in DFs. The TF and IDF are calculated in the formula (1) and formula (2) respectively.

$$TF(w_i) = \frac{\#w_i \ in \ requirement}{\#all \ words \ in \ requirement} \tag{1}$$

$$IDF(w_i) = \log \frac{\#all \ requirements}{(\#requirements \ containing \ w_i) + 1} \tag{2}$$

If w_i is at the beginning or the end of a sentence, and the contextual words in the window are empty, DEX assigns the default feature values ($\langle NULL \rangle$ for lexical feature, $\langle NULL \rangle$ for part-of-speech features, 0 for TF-IDF feature) to the empty words.

After that, the sentence s is converted into a feature sequence $F_l = [f(w_1), f(w_2), \ldots, f(w_m)]$, where $f(w_i)$ is extracted features for each word. Then, DEX builds the corresponding label sequence $Y = [y_1, y_2, \ldots, y_n]$. For each labeled requirement, there are corresponding DFs which are manually labeled development teams and reviewed by FPA experts (the process of building labeled requirements corpus is described in Section 5.2). For each requirement, DEX splits each DF into a word sequence W_{DF}, and determines DF's location in s by sub-sequence matching. Then, DEX gives a label y_i to each word w_i, which

[5] https://stanfordnlp.github.io/CoreNLP/.

[6] Due to that our dataset is written in Chinese, we do not stem each word into its morphological root like other languages, e.g., English.

Word	I	would	like	to	receive	the	credit	card	bill	when	I	open	the	mobile	application	in	IOS12.4
Label	0	0	0	0	0	0	B	I	E	0	0	0	0	0	0	0	0

Fig. 3. A tagging example (translated into English).

indicates the location of DF in requirements. The y_i in represented in the *BIOES* format [36].

- *I-label (Inside)*: The word is inside DF but not the first or last within the DF.
- *O-label (Outside)*: The word is outside DF.
- *B-label (Beginning)*: The word is the beginning of DF.
- *E-label (End)*: The word is the end of DF.
- *S-label (Singleton)*: The singleton word is a DF.

For example, there is a sentence "I would like to receive the credit card bill when I open the mobile application in IOS12.4", and the corresponding DF is "credit card bill".[7] The tagging result is shown in Fig. 3. Finally, each pair $\langle F_l, Y \rangle$ is considered a seed for CRF training.

Building unlabeled instances. Given an unlabeled requirement R_u, each sentence is also converted into a word sequence $s = [w_1, w_2, \ldots, w_m]$. Same as building seeds, DEX extracts contextual for each w_i and builds the feature sequence $F = [f(w_1), f(w_2), \ldots, f(w_m)]$. Given that there are not DFs for R_u, DEX does not build the corresponding label sequence. Each feature sequence $\langle F \rangle$ is considered an unlabeled instance.

4.2. Bootstrapping based CRF training

To overcome the challenge of expensive label resources, DEX adopts a bootstrapping based algorithm for CRF training. In general, DEX trains an initial CRF model using seeds, iteratively expands the seeds from unlabeled instances, and retrains the CRF model until reaching the stop criterion. The details of the bootstrapping based algorithm are shown in Algorithm 1 First, DEX trains an initial CRF model (M_0) using seeds (Line 1). Second, DEX expands seeds and re-trains the CRF model iteratively in the way of bootstrapping (Line 3-15). In each iteration, DEX utilizes the CRF model to predict the label sequence Y_p of each unlabeled instance, and the CRF model calculates the confidence crf_{score} of prediction (Line 6). For each unlabeled instance, if the crf_{score} is greater than or equal to the corresponding threshold (T_{crf}), it will be added into the seeds (Line 7-9). When new instances are automatically added to the training set during bootstrapping, it is critically important that their labels are correct, otherwise, the performance of the CRF model will rapidly deteriorate. This suggests that DEX should employ a high threshold to add the instances with high confidences into the training set. On the other hand, a high threshold often yields a few new instances, which can cause the bootstrapping process to sputter and halt. In order to balance these competing demands, DEX uses a "sliding threshold" strategy. DEX sets a high T_{crf} in the beginning, and decreases T_{crf} by δ_{crf} (Line 11) with iterations. At the end of each iteration, DEX re-trains the CRF model with the expanded seeds (Line 12-13). When reaching the stop criterion ($T_{crf} < low\_limit_{crf}$), the bootstrapping process ends. The T_{crf}, low_limit_{crf} and δ_{crf} are the hyper-parameters which are set as 0.9, 0.7 and 0.01 by default.

Algorithm 1 Bootstrapping Based CRF Training Algorithm

Input:
D_{seeds}: seeds
$D_{unlabeled}$: unlabeled instances
T_{crf}: the CRF confidence threshold
δ_{crf}: the value decreasing T_{crf}
low_limit_{crf}: the low limit of T_{crf}
Output:
M_{final}: the final CRF model

```
1:  M_0 ⇐ train CRF model with D_seeds
2:  i ⇐ 1
3:  while T_crf ≥ low_limit_crf do
4:    T ⇐ { }
5:    for < F > in D_unlabeled do
6:      < F, Y_p >, crf_score ⇐ label s using M_{i−1}, and calculate the
        confidence
7:      if crf_score ≥ T_crf then
8:        T ⇐ T ∪ {< F, Y_p >}
9:      end if
10:   end for
11:   T_crf ⇐ T_crf − δ_crf
12:   T ⇐ T ∪ D_seeds
13:   M_i ⇐ train CRF model with T
14:   i ⇐ i + 1
15: end while
16: M_final ⇐ M_i
```

4.3. DF-oriented language model training

The language model is an attempt to capture regularities of natural language and amounts to estimating the probability distribution of various linguistic units, such as words, sentences, and whole documents [37]. We consider that there are linguistic regularities within words when describing DFs such as word co-occurrence and word combination. Thus, DEX leverages the DF-oriented language model to estimate the probability distribution of the words in each historical DF. With the historical DFs, DEX trains a DF-oriented language model and used the trained language model to determine whether an extracted phrase is in line with the regularities within the historical DFs. The basic assumption is that if an extracted is more consistent with the expression law of historical DFs, it is more likely to be a DF.

Building language model instances. Given a historical DF, DEX splits it into word sequence $[w_1, w_2, \ldots, w_n]$, where w_i is the word in the historical DF. Each word sequence is considered as a language model instance for training the DF-oriented language model. Second is **training DF-oriented language model.** For each language model instance $[w_1, w_2, \ldots, w_n]$, the DF-oriented language model models the joint probability $P(w_1, w_2, \ldots, w_n)$ using Eq. (3).

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)\prod_{k=3}^{m}(w_k|w_{k-2}w_{k-1}) \tag{3}$$

The DF-oriented language model is trained with the help of open-source library *Kenlm Language Model*.[8]

4.4. Data function extraction

When a new requirement arrives, DEX extracts the candidate DFs using the CRF model and ranks the candidates using the DF-oriented language model. The following introduces the two sub-steps.

Extracting candidate data functions. For each sentence in the new requirement, the trained CRF model receives its word sequence

[7] All the examples in our study are written in Chinese originally, we manually translate it to English.

[8] https://github.com/kpu/kenlm.

representation $[w_1, w_2, \ldots, w_m]$, extracts features for each word, and returns a label sequence $[y_1, y_2, \ldots, y_m]$, where y_i (in the *BIOES* format) is the label given to w_i. DEX forward scans the label sequence. Using the labels "B", "I", "E" and "S", DEX determines the location of each candidate DF in the word sequence. For example, given a sentence *"I want to add the system reminder service on my personal page"*, the word sequence and corresponding label sequence are shown in the DF extraction phrase in Fig. 3. The "Sequence" row corresponds to the word sequence of the sentence and the "Label" row represents the corresponding label sequence. In this example, the labels "B", "I" and "E" correspond to the words "system", "reminder" and "service", and the next sequence label "O" represents the corresponding word no longer belongs to the current candidate DF. Therefore, the extracted candidate DF is "system reminder service". Following the above process, all candidate DFs could be extracted after parsing through the labeled sequence.

Ranking candidate data functions. DEX ranks the candidate DFs by *perplexity* which is a measurement of how well a probability distribution or probability model predicts a sample [38]. Lower perplexity indicates that the candidate DF is more in line with the regularities of historical DFs. For each candidate DF, the DF-oriented language model calculate the perplexity using Eq. (4).

$$Perplexity(DF) = 2^{-P_{lm}(DF)\cdot \log P_{lm}(DF)} \tag{4}$$

After that, DEX keeps the candidate DF whose perplexity is smaller than the perplexity threshold as the final DFs. For the perplexity threshold, DEX also calculates the perplexity of each historical DF, ranks the perplexities in ascending order. The perplexity threshold is determined as $Q_3 + (Q_3 - Q_2)$, where Q_3 is the upper quartile, and Q_2 is the median. The candidate whose perplexity is larger than the threshold is considered an anomaly and removed from the final DF list.

5. Experiment design

In this section, we propose the research questions, introduce the studied subjects, present the experimental setup and baselines, and illustrate the measurements for evaluation.

5.1. Research questions

Our evaluation addresses the following five research questions:

RQ1: (Baseline Comparison) Can DEX outperform the state-of-art baselines methods? To evaluate the effectiveness of DEX, we evaluate its performance and compare it with four state-of-the-art baselines.

RQ2: (Component Evaluation) How do bootstrapping and DF-oriented language model contribute to DEX? To demonstrate how the bootstrapping based algorithm and DF-oriented language model contributions to DEX, we conduct comparative experiments to investigate the effects of the two components.

RQ3: (Hyper-parameter Sensitivity) Is DEX sensitive to hyper-parameters? There are two sets of model parameters in DEX needing to be well-tuned, i.e., context windows size and parameters for the bootstrapping based algorithm. We train DEX under different model parameters to investigate the sensitivity of the hyper-parameters for the performance.

RQ4: (Instance Size Sensitivity) To what extent is DEX sensitive to different instances sizes? We compare the performance of DEX under different sizes of unlabeled and labeled requirements to investigate whether the performance would decline if we decrease the number of instance sizes.

RQ5: (Expert Review) How does DEX work in real-world applications? We conduct an FPA expert review to investigate the usefulness of DEX in practices. we randomly sample the projects operating on the continuously evolved business systems. The DFs extracted by DEX and engineers are manually reviewed by requirement experts respectively.

Table 1
Summary of labeled and unlabeled requirements.

System ID	Business field	Labeled requirement		Unlabeled requirements
		Requirements	DFs	Requirements
1	Personal loan	805	534	801
2	Merchant management	108	146	124
3	Measurement tool	88	124	72
4	Payment	68	68	60
5	Data warehouse	64	88	92
6	Cloud computing	192	208	188
7	Marketing management	88	96	92
8	Private bank	869	1053	800
9	Retail	684	782	672
10	Settlement	156	222	153
11	Foreign trade	84	111	80
12	Charter business	48	32	59
13	Innovative product	21	10	23
14	Communation platform	18	5	21
15	Joint card	94	24	92
16	Block chain	42	10	42
17	Intelligent answering	8	5	8
18	Data analysis	33	18	33
19	Anti-fraud	14	28	13
20	Business analysis	102	46	75
Total	–	**3586**	**3610**	**3500**

5.2. Dataset

Our dataset comes from a financial company, China Merchants Bank (CMB). CMB is the largest joint-stock commercial bank wholly owned by corporate legal entities in China. CMB involves a large variety of banking business areas, including debit card management, credit card management, wealth management products, investment advisory products, cash management, online customs tax payment, online bill acceptance. By the end of 2019, with over 70,000 employees, CMB had set up a service network that consists of more than 1800 branches worldwide. To support its business, the IT department of CMB develops and maintains over 500 financial software, which involves huge budgets and employees. Most financial software is transaction-oriented applications with data persistence, which is well adapted to functional sizing. CMB has utilized FPA to estimate and measure system size over 10 years. Our approach is used in the enterprise to automatically extract DFs from requirements to support FPA. There are two corpora obtained from CMB, i.e., requirements corpus and historical DF corpus. The two corpora are both written in Chinese. We will introduce the two corpora in detail.

Requirements corpus. This requirement corpus contains 7086 requirements across 20 maintenance projects which were conduct between January 2018 and June 2019. The 20 maintenance projects are from 20 distinct software systems which are continuously maintained and evolved over the past 5 years. Meanwhile, to support function point measurement, CMB has maintained a DF list manually. Typically, the development teams are responsible for extracting and maintaining the DF list, after passing the review and verification by FPA experts. This dual effort is to ensure the quality of the DF list, avoiding reckless errors or individual bias from the development teams. However, due to the expensive expert-associated costs, not all requirements will go through this labeling process. In this study, the labeled requirements corpus contains 3586 requirements produced following the above process (DF extraction by development teams and review by FPA experts). The unlabeled requirements corpus contains 3500 requirements that are not reviewed and verified by FPA experts. The details of labeled requirements and unlabeled requirements are shown in Table 1.

Historical DF corpus. CMB manually maintains a DF dictionary which contains all historical DFs extracted by the development teams and verified by the FPA experts. The dictionary contains DFs involving over 6000 projects. The historical DF corpus is built from the DF

dictionary. To ensure that all the historical DFs had produced before the 20 projects in Table 1 were conduct, We exclude all the DFs from the 20 projects in Table 1, and randomly sample 6000 DFs before January 2018 as historical DF corpus.

5.3. Experimental setup

To answer RQ1, we randomly divide all labeled requirements into two sets, i.e., training set (90% of the labeled requirements) and test set (10% of the labeled requirements). For each experiment, we use the 90% labeled requirements, all the unlabeled requirements, and all the historical DFs to train DEX. The performance is evaluated on the test set. The experiment is repeated 10 times. The average of 10 experiments is used as the final performance. Meanwhile, we compare with four state-of-the-art baselines to investigate the advantage of DEX. Mann–Whitney tests are conducted between DEX and four baselines respectively to test the differences. In addition, we leverage the Scott-Knott Test [39] to rank the performance of DEX and baseline approaches.

To answer RQ2, we conduct the experiments under different settings:

- **CRF**: We remove the bootstrapping based algorithm and the DF-oriented language model from DEX. Under this setting, the labeled requirements are randomly divided into the training set and test set in the ratio of 9:1. We train the CRF model only using the training set and evaluate the model using the test set. This setting is used as the comparison benchmark.
- **CRF + Bootstrapping**: We remove the DF-oriented language model from DEX, i.e., only using the bootstrapping based algorithm to train the CRF model. Under this setting, we divide the labeled requirement into the training set and test set by the ratio of 9:1. We train the CRF model using 90% labeled requirements and all the unlabeled requirements and evaluate the performance on the test set.
- **CRF + Bootstrapping + LM**: the complete steps of DEX. Under the setting, we divide the labeled requirement into the training set and test set by the ratio of 9:1. We use the 90% labeled requirements, all the unlabeled requirements to train the CRF model. Then, we use all the historical DFs to train the DF-oriented language model to rank the candidates extracted by the CRF model. Finally, we evaluate the performance on the test set.

For each setting, the experiment is repeated 10 times. Mann–Whitney tests are conducted to test the differences between the settings.

To answer RQ3, there are two sets of parameters needing to be well-tuned: (1) context window size n; (2) T_{crf} (the CRF confidence threshold, in the range of [0,1]), low_limit_{crf} (the low limit of T_{crf} in the rage of [0, T_{crf}]) and δ_{crf} (the value decreasing T_{crf}) in the bootstrapping based algorithm. We adopt a greedy strategy to tune the two sets of parameters. First, we set three bootstrapping parameters as constant ($T_{crf} = 0.90$, $low\_limit_{crf} = 0.70$ and $\delta_{crf} = 0.01$), and tune the context window size n from 1 to 10 by the steps of 1. The context window size n is determined as the value with the best performance. Second, we investigate different combinations of three bootstrapping parameters. We set the context window size n as constant, and use 15 combinations of bootstrapping parameters.

Table 2 shows the 15 combinations, where combination 1-5 aim to tune the T_{crf}, combination 6-10 aim to tune low_limit_{crf} and combination 11–15 intend to tune δ_{crf}. Specifically, in the bootstrapping based algorithm, due to the small number of seeds, if a large number of noisy samples are added to the training set in the first few iterations, it will cause disastrous performance degradation. Thus, we set the candidate bootstrapping parameter T_{crf} as [1.00 (c1), 0.90 (c2), 0.80 (c3), 0.70 (c4), 0.60 (c5)] to ensure that highly reliable instances are added into training set in the beginning of the bootstrapping process. As for the low_limit_{crf}, it controls the lower confidence limit of instances added to the training set. We consider that a larger value (larger than 0.8) will result in an ineffective expansion of the training set, and a smaller value (smaller than 0.4) will introduce more noise. Therefore, we set the candidate low_limit_{crf} values as [0.80 (c6), 0.70 (c7), 0.60 (c8), 0.50 (c9), 0.40 (c10)] for parameter tuning. As for δ_{crf}, it control the iteration numbers. In order to achieve the balance of model performance and time efficiency, we empirically set the candidate δ_{crf} values as [0.20 (c11), 0.15 (c12), 0.10 (c13), 0.05 (c14), 0.01 (c15)] For each experiment, we randomly divide all labeled requirements into two sets, i.e., training set (10% labeled requirements) and test set (90% labeled requirements). We train DEX using the 10% labeled requirement, all the unlabeled requirements, and all the historical DFs, and evaluate the performance on the test set. For each parameter combination, the experiment is repeated 10 times, and the average of the 10 experiments is considered as the final performance.

Table 2
The 15 combinations of bootstrapping parameters.

Combination	T_{crf}	low_limit_{crf}	δ_{crf}
c1	1	0.5	0.01
c2	0.9	0.5	0.01
c3	0.8	0.5	0.01
c4	0.7	0.5	0.01
c5	0.6	0.5	0.01
c6	0.9	0.8	0.01
c7	0.9	0.7	0.01
c8	0.9	0.6	0.01
c9	0.9	0.5	0.01
c10	0.9	0.4	0.01
c11	0.9	0.7	0.20
c12	0.9	0.7	0.15
c13	0.9	0.7	0.10
c14	0.9	0.7	0.05
c15	0.9	0.7	0.01

To answer RQ4, we train DEX with different volumes of unlabeled and labeled requirements, and evaluate corresponding performance. We conduct two settings to evaluate the sensitivity of unlabeled and labeled requirements respectively. First, we use 90% randomly divided labeled requirements to train the CRF model, and all historical DFs to train the DF-oriented language model. The performance under different sizes of unlabeled requirements is evaluated using the remaining 10% labeled requirements. Second, we take the number of unlabeled requirements and historical DFs as constant, and randomly sample K% labeled requirements as seeds for training and remaining (100-K)% labeled requirements for evaluation. We set K as 10, 20, 30, ..., 90. Both settings are repeated 10 times respectively, and the average of 10 experiments is used as the final performance.

To answer RQ5, an expert review is assigned and conducted in September 2019 to evaluate the usefulness of DEX in the real-world application scenario. With the support of CMB, we apply DEX in 11 new projects, and compare with results produced following the manual labeling process, as introduced in Section 5.2. The process of the expert review is illustrated in Fig. 4. Specifically, the user study follows five steps: (1) first, we obtain the new requirements of the 11 projects and apply DEX to extract the DFs automatically; (2) next, for each project, CMB independently collects DFs extracted by corresponding development teams; (3) then, we also automatically extract the DFs from the 11 projects using DEX; (4) a certified FPA expert, who has been working on FPA for more than 5 years, manually reviewed the requirements and the DFs extracted by development teams and DEX respectively; (5) finally, the evaluation metrics are calculated and considered as expert review results.

5.4. Baselines

The following introduces the four baseline approaches.

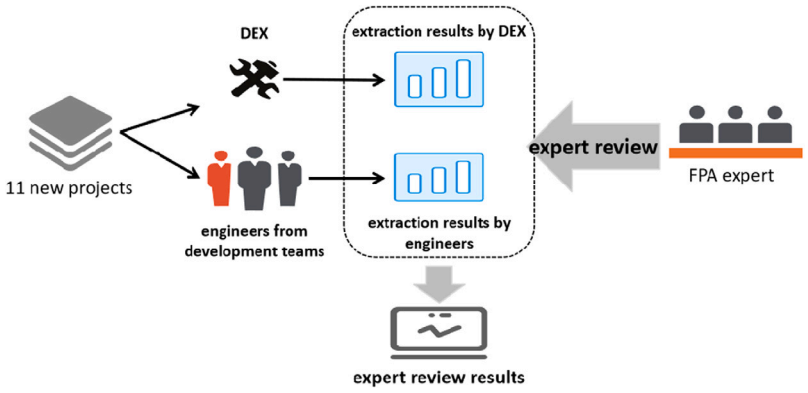


Fig. 4. The process of expert review.

Baseline 1-AERGT [13]: It is the state-of-the-art approach to automatically extract glossary terms from requirements. AERGT is a heuristics-based approach that does not require the training process and has achieved promising performance in the Satellites and embedded system domains. Following the steps introduced in AERGT, we firstly extract the candidate noun phrases from labeled requirements using the text chunking with the help of Standford CoreNLP. Then, we refine the candidates following the linguistic rules introduced in AERGT.

Baseline 2-CNER [11]: It is a supervised learning-based approach, which achieves promising performance for the general entity extraction task. It automatically detects word- and character-level features using a hybrid bidirectional LSTM and CNN architecture, eliminating the need for most feature engineering We make use of the implementation published on the Github,[9] and train the network using the labeled requirements.

Baseline 3-ACDO [33]: It is a semi-supervised learning-based approach to automatically extract domain-specific glossary terms from code documents. It also aims at building a machine learning model with insufficient labeled data. ACDO takes the glossary term as a sequence tagging task and also utilizes a bootstrapping based algorithm to train an LSTM-CRF model. Following the approach, we use the labeled requirements as the initial training samples and train an initial LSTM-CRF model. After that, we implement the iterative process introduced in ACDO to expand training samples from unlabeled requirements and retrain the CRF model until reaching the stop criteria.

Baseline 4-RENE [12]: It is our previous study to extract entities from requirements. RENE leverages the LSTM-CRF network for entity extraction, and employs a transfer learning framework to overcome the challenge of limited labeled data. There are two corpora for training RENE, i.e., general corpus and domain-specific requirements corpus including unlabeled requirements and labeled requirements. The unlabeled requirements and labeled requirements are the same as the requirements corpus in this paper (details in Table 1). Compared with DEX, RENE leverages extra 12,000 Wikipedia articles as a general corpus to pre-train the embedding layer and LSTM layer in the LSTM-CRF network. We use RENE as a baseline to investigate whether DEX outperforms RENE in the terms of extraction performance (details in Section 6.1), training cost (details in Section 6.1) and required data volume (details in Section 6.4).

5.5. Evaluation metrics

For all the experiments, we use three commonly-used measurements to evaluate the performance, i.e., *Precision, Recall, F1* [40]:

- *Precision*: the percentage that the DFs returned by DEX that are correct.

$$Precision = \frac{\#\ correctly\ extracted\ data\ functions}{\#\ all\ extracted\ data\ functions} \quad (5)$$

- *Recall*: the percentage of the ground truth DFs returned by DEX.

$$Recall = \frac{\#\ correctly\ extracted\ data\ functions}{\#\ all\ ground\ truth\ ground\ data\ functions} \quad (6)$$

- *F1*: is the harmonic mean of Precision and Recall.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (7)$$

Please note that, if an extracted DF is the same as anyone's ground truth DF in the requirement, it is considered as a correctly extracted DF.

6. Results

This section reports the analysis of the achieved results aiming at answering the five research questions.

6.1. Baseline comparison

Fig. 5 shows the box-plot comparison of DEX with four baselines. The results show that DEX reaches 84% precision, 80% recall, 82% F1 on average, and outperforms all four baselines. The details of average precision, recall, and F1 scores of all five approaches are listed in Table 3, along with the time cost of training the model (running on the personal computer with I7-8700K CPU, 32 GB memory, and GTX 2060 GPU). The numbers in the brackets are the differences compared to DEX. The "∗" symbol suffixed with the number indicates that the difference is significant (the significant level at 0.05). In general, DEX could reach 84% precision, 80% recall, and 82% F1. Moreover, we compare the performances of DEX and other baseline approaches using Scott-Knott Test (the significant level at 0.05). The performances from high to low are as such sequence: DEX, RENE, ACDO, CNER and AERGT. The results indicate that DEX outperforms all four baseline approaches with the least training cost. In the following, we compare DEX with four baselines respectively.

First, we compare DEX with AERGT. As for precision, the performance of AERGT is worse than DEX. It is due to that AERGT is a linguistics-based approach, and it extracts all noun phrases that match the linguistic rules no matter whether the noun phrases are DFs or not. Considering that the DFs describe the functional components in requirements, not all the noun phrases in requirements are DFs. For example, when processing a requirement *"As a credit card user, I want to receive the bill reminder message on Android 9.0, so that I could pay back my credit card on time"*. AERGT extracts not only the DF "bill reminder message" but also regular terms such as "credit card", "Android 9.0". It indicates that the training process is necessary to distinguish which ones of them

[9] https://github.com/kamalkraj/Named-Entity-Recognition-with-Bidirectional-LSTM-CNNs.

Table 3
The results of baseline comparison.

Approach	Precision	Recall	F1	Training cost (min)
DEX	84%	80%	82%	31 (CRF+Language Model)
AERGT	46% (−38%*)	60% (−20%*)	51% (−31%*)	–
CNER	56% (−28%*)	53% (−27%*)	55% (−27%*)	114
ACDO	64% (−20%*)	69% (−11%*)	67% (−15%*)	248
RENE	79% (−5%*)	81% (+1%)	80% (−2%)	1394

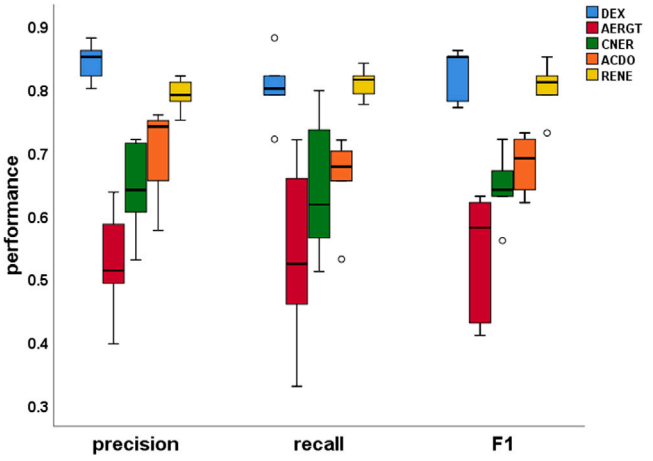


Fig. 5. The Performances of DEX and baselines.

are DFs, and the learning-based approaches are the better choice to achieve promising precision. As for the recall, DEX also outperforms AERGT, which indicates that DEX could retrieval more DFs than AERGT on average. The possible reason is that AERGT does not perform well for the domain-specific DFs which are the out-of-vocabulary words for the general natural language processing techniques. Specifically, the DFs are typically named according to business knowledge. AERGT is based on the noun chunking [41], a general natural language processing technique to discover noun phrases from texts, and produces parsing errors when handling DFs. As a result, some DFs cannot be extracted correctly, which affects the recall. DEX, which is built on the domain-specific contexts, could take not only terms themselves but also contextual information into consideration. Compared with a DF itself, it is easier to determine the boundary of the DF through contextual information. Thus, DEX could handle the issue better.

CNER is a deep learning approach, which shows poor performance, mainly due to the limited labeled data. Although ACDO also utilizes the bootstrapping based algorithm, DEX also outperforms ACDO. It is mainly due to two reasons: (1) ACDO utilizes the LSTM-CRF network that is a deep learning model, and it involves more model parameters to be trained. When there are few labeled data, the network still could not get effective training even using the bootstrapping based algorithm; (2) DEX uses an extra DF-oriented language model to rank the extracted candidates, and the evaluation results show that DEX could effectively filter out the irrelevant phrases extracted by the model (details in Section 6.2).

Compared with our previous RENE, DEX could increase 5% precision. Although the recall drops by 1%, the overall performance of DEX increases 2% in F1. Similarly, RENE also models the extraction as the sequence tagging task, and leverages the transfer learning framework [42] to boost the performance using general corpus and unlabeled requirements. Therefore, it also achieves promising performance in DF extraction. However, RENE is based on the LSTM-CRF network which is a deep learning model. When reducing the number of labeled requirements, the performance of RENE sharply declines and the advantage of DEX is more obvious. Specifically, when reducing the amounts of labeled requirements and unlabeled requirements to 361 and 1500 respectively, RENE achieves 62% precision and 32% recall [12]. While, DEX could also reach 76% precision and 78% recall (the analysis is described in Section 6.4). As for the training cost, due to that DEX adopts a simplified CRF model with hand-crafted features and a lightweight DF-oriented language model, DEX takes much less training time than RENE.

Summary: DEX could reach 84% precision, 80% recall and 82% F1 using limited labeled requirements. Moreover, DEX could significantly outperform four state-of-the-art baselines with the least training cost, which indicates the advantages of DEX.

Table 4
Summary of performance under different training settings.

Configuration	Precision	Recall	F1
CRF	78%	58%	66%
CRF + Bootstrapping	68% (−10%*)	85% (+27%*)	76% (+10%*)
CRF + Bootstrapping + LM	80% (+2%)	84% (+26%*)	82% (+16%*)

6.2. Component evaluation

Table 4 shows the performance of DEX under different training configurations. The figure in bracket is the difference compared to *CRF*. The symbol "∗" is suffixed with the figure if the Mann–Whitney Test shows the difference is significant.

We firstly compare "CRF" and "CRF + Bootstrapping + LM". The results indicate that adding the bootstrapping and the DF-oriented language model into the DEX could increase precision, recall, and F1 by 2%, 26%, 16% on average. It indicates that bootstrapping and DF-oriented could significantly improve the recall without loss of precision, to achieve a promising performance.

Then, we compare "CRF" and "CRF + Bootstrapping". The precision of "CRF" is significantly larger than the precision of "CRF + Bootstrapping". The decrease brought by bootstrapping indicates that some false positive instances are added into the seed set during the iterative process. However, the recall of "CRF + Bootstrapping" is significantly larger than that of "CRF". Correspondingly, the F1 also increases by 10% with the help of the bootstrapping based algorithm. The result implies that the bootstrapping based algorithm could significantly improve the performance for recall and F1, with a slight decrease in precision.

Considering that the DF-oriented language model is utilized to rank the candidate DFs after bootstrapping. We further investigate the performance differences before and after ranking. We find that the precision increases 12%, and recall decreases 1% on average. The overall performance F1 increase by 6%. This indicates that the DF-oriented language model could effectively filter the false positive candidates introduced by the bootstrapping based algorithm without significantly decreasing recall, which is in line with our purpose of using the DF-oriented language model to remove the noun phrases which are irrelevant to DFs.

Summary: Overall, DEX could improve the performance by 2% precision, 16% recall and 12% F1. It is consistent and confirming with the design purpose of boosting the performance by unlabeled requirements and historical DFs. Specifically, the bootstrapping based algorithm increases the recall but decreases precision. DF-oriented language model remedies the problem to improves the precision significantly without sacrificing recall. Totally, the recall has been significantly increased, and the precision keeps promising performance.

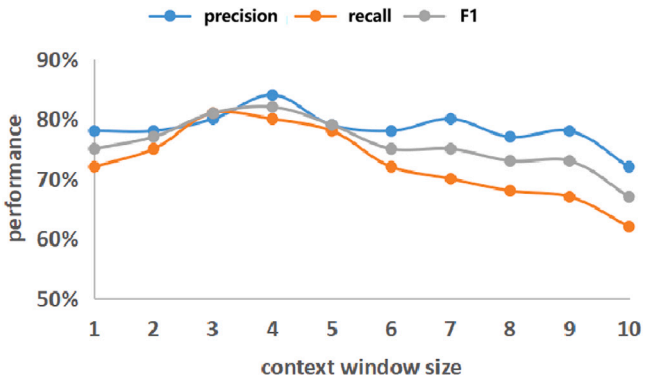


Fig. 6. The performances under different context window sizes.

6.3. Hyper-parameter sensitivity

Fig. 6 shows the performances under different context windows sizes. In general, DEX achieves the best performance when the window size is 4 (84% precision, 80% recall, and 82% F1). Moreover, the size of the context window does not bring obvious changes in precision changes, and the precision could be maintained between 78% and 84% in most cases. But for recall, we found it dropping rapidly after 4. When the context window is 10, recall drops to 62%. The results indicate capturing too much context information will lead to the decline of recall.

Using the greedy strategy introduced in Section 5.3, we set the context window size as 4 that achieves the best performance under different training configurations. Fig. 7 shows the performances under different bootstrapping parameters. Specifically, Fig. 7(a) shows the performances of c1-c5 which set low_limit_{crf} and δ_{crf} as constants and change T_{crf}. DEX achieves the best performance when T_{crf} is 0.9. However, compared to other configurations, the differences are not significant. Then, we set T_{crf} as 0.9, Fig. 7(b) shows the performances under different configurations of low_limit_{crf}. Among the first three configurations c6-c8 (low_limit_{crf} is larger than 0.6), there is no significant difference in performance. After c8, the performance significantly drops. The results indicate that too low a value low_limit_{crf} results in too many false positives added into the seed set, which leads to performance degradation. Then, we set T_{crf} as 0.9 and low_limit_{crf} as 0.7 when DEX achieves the best performance. Fig. 7(c) shows the performances under different configurations of δ_{crf}. We can see that the performance gradually increases as δ_{crf} decreases. When δ_{crf} is 0.01, DEX could achieve the best performance (80% precision, 84% recall and 82% F1). Large δ_{crf} means adding more samples at once to expand the seed set, which may introduce more bias to the seed set and results in performance degradation.

Summary: The hyper-parameters can affect the performance of DEX. Tuning these hyper-parameters is a key step to achieve promising model performance. The greedy strategy is an effective way to determine the values of these hyper-parameters in practice.

6.4. Instance size sensitivity

As reported in Section 6.1, DEX could achieve promising results using 3586 labeled requirements and 3500 unlabeled requirements. Considering that the domain-specific requirements are difficult to access and expensive to label, we further investigate whether DEX could still when reducing the number of available requirements.

Fig. 8 shows the average performance under different sizes of unlabeled requirements. The results show that the size of unlabeled requirements has no obvious effect on the precision, but reducing the size of unlabeled requirements will decrease the recall. Before 1500, the recall curve is relatively gentle. Recall obviously decrease after 1500. The results indicate that if the number of unlabeled requirements is reduced to less than 1500, the recall will be obviously reduced. Therefore, 1500 is identified to be an acceptable point to achieve nearly optimal model performance.

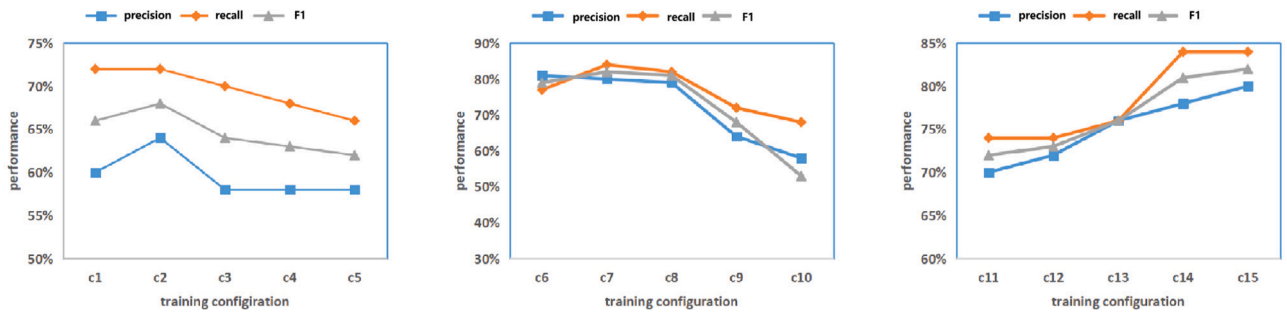
Then, we use all historical DFs and 1500 unlabeled requirements and change the sampling sizes of labeled requirements. Fig. 9 shows the average performances under different sizes of labeled requirements. The results show that with the reduction of training instances, both precision and recall will decrease. From the overall trend of accuracy and recall, it can be observed that the decline of precision is greater compared to recall. For precision, the results of the Mann–Whitney Test between each pair of adjacent sizes show that precision will not significantly decrease before 60% (2166) labeled requirements and significantly decrease after 60%. Thus, 2166 is a reasonable number of labeled requirements to train the CRF model in DEX. Moreover, compared with our previous study RENE, DEX could achieve acceptable performance with a small number of requirements. For example, DEX could reach 76% precision and 78% recall with 361 (10%) labeled requirement and 1500 unlabeled requirement, while RENE reaches 62% precision and 32% recall with same size (361) of labeled requirements and more (2500) unlabeled requirements [12].

Summary: In general, 1500 unlabeled requirements and 2166 labeled requirements are considered sufficient to train the CRF model in DEX. Besides, DEX could still achieve promising performance using 1500 unlabeled requirements and 361 labeled requirements, which indicates the DEX's advantage with a small number of requirements.

6.5. Expert review

Table 5 shows the results of usefulness evaluation of DEX using the 11 industry projects. The column "Developer" illustrates the manually extracted data from corresponding development teams. The column "DEX" shows the DFs automatically extracted by DEX, along with its performance metrics of "precision" and "recall". The figures in brackets indicate performance differences compared to the performance of development teams. The results show that in the real industrial application scenario, the quality of manual extraction is not promising (66% precision and 70% recall). In comparison, the results of the Mann–Whitney Test show that the differences between "Developer" and "DEX" are significant for both "precision" and "recall". The results indicate that DEX could produce more accurate and completed DFs for almost all the projects. By analyzing the DFs extracted by developers, we find that the major reason for the differences is that DEX tends to extract more complete phrases that describe the particular business concepts, while developers tend to extract partial phrases, esp. for rather long, business-specific DFs. For example, there is a sentence *"As a business manager, I would like to automatically load the merchant loan information template when the merchant fills in the loan information so that the merchant could fill in the relevant information in the specific format"*. The DF extracted by the developer is *"loan information template"*, while the ground truth is *"the merchant loan information template"*. It corresponds to a loan form specific for the merchants rather than personals in CMB. This indicates that manual labeling is prone to extracting incomplete expressions, leading to ambiguity and affect subsequent software development. By comparison, DEX could handle the problem better and extract the complete DFs in practices.

Summary: When applying DEX to 11 on-going software projects, it achieves the precision and recall both at 82% and 83% respectively for automated DF extraction, and significantly outperform the manual extraction by the developers. The results indicate that the more accurate and complete DF list for each project could be built with the help of DEX.



(a) T_{crf} (b) low_limit_{crf} (c) δ_{crf}

Fig. 7. The performances under different bootstrapping parameters.

Table 5
The data functions extracted by engineers and DEX.

Project	# Requirements	Developer			DEX		
		# Data function	Precision	Recall	# Data function	Precision	Recall
Project-1	46	45	62%	70%	41	78%	80%
Project-2	26	26	58%	68%	22	82%	82%
Project-3	40	41	71%	78%	36	89%	86%
Project-4	19	18	67%	71%	17	82%	82%
Project-5	29	29	79%	74%	38	76%	94%
Project-6	14	15	73%	79%	15	73%	78%
Project-7	30	32	69%	71%	25	92%	74%
Project-8	51	48	58%	64%	47	83%	84%
Project-9	24	24	75%	67%	30	77%	85%
Project-10	33	33	55%	62%	30	83%	86%
Project-11	44	44	57%	63%	41	83%	85%
Average	–	–	**66%**	**70%**	–	**82% (+16%)**	**83% (+13%)**

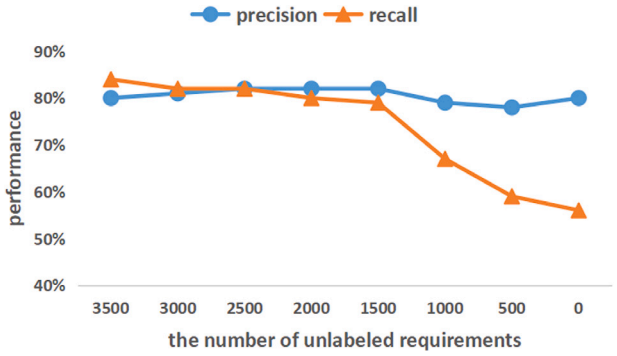


Fig. 8. The performance under different sizes of unlabeled requirements.

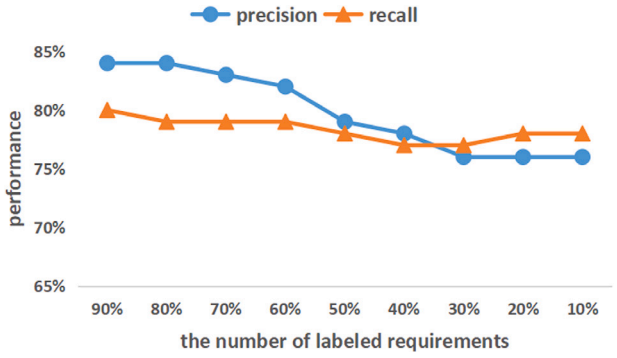


Fig. 9. The performance under different sizes of labeled requirements.

7. Discussion

In this section, we discuss the prospective aspects of our results, illustrate the expectations for researches on automated FPA, highlight lessons learned, as well as threats of the proposed approach.

7.1. How to infer the transactional functions with the data functions?

As we mentioned, the DFs are expressed as noun phrases, and the transaction functions are described in the form of "verb + DFs". Given the requirements and the extracted DFs, the key to inferring transaction functions is identifying the corresponding operations acting on the DFs. We randomly sample 100 requirements together with corresponding DFs and transaction functions in Table 1, and manually investigate how to find the operations given the DFs. There are mainly two situations.

The first one is that the DF and the corresponding operation are written in the same sentence, and the core verb of the sentence corresponds to the functional operation. This situation accounts for 89 out of 100 sampled requirements. For example, there is a requirement "I would like to modify billing reminder time so that I can pay on time".. The DF is "billing reminder time", and the corresponding operation "modify" is in the same sentence. Fig. 10 shows the part-of-speech and dependency parsing results of the sentence returned by Stanford CoreNLP. The part-of-speech tag "VB" indicates that "modify" is the core verb of the sentence, and the dependency symbol "dobj" indicates that the compound noun "billing reminder time" is the direct object of the verb. Accordingly, we can infer that the "modify" is the operation acting on the DF, and the transaction function in requirement is "modify billing reminder time". Generally, given a DF, we can infer the functional operation by finding the sentences containing the DF, and parsing the core verb and syntactic dependency of the sentences.

The second one is that the core verb of the sentence containing the DF corresponds to multiple operations due to the coarse-grained descriptions of the requirements. For example, there is a requirement

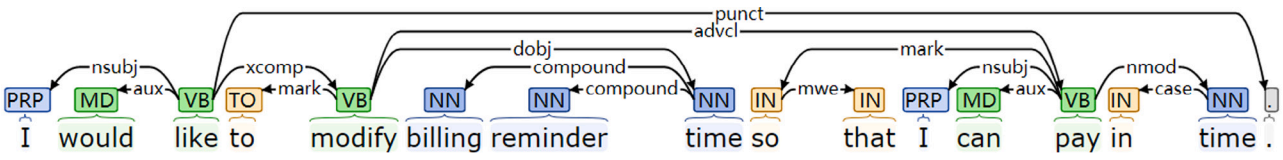


Fig. 10. The parsing results by Stanford CoreNLP.

"As the customer manager, I would like to maintain the basic customer information so that I could update customer information in time". From the results of part-of-speech and dependency parsing, we can find "maintain" is the core verb. However, there are four operations corresponding to the requirement, i.e., "add", "delete", "modify", and "search". In this situation, the functional operations could be inferred by building the inference rules. For the above example, the inference rules could be built as "If 'maintain' is the core verb of the sentences containing DFs, the functional operations are 'add, 'delete', 'modify', and 'search' ". Some association rule learning techniques [43,44] could also be applied to automatically mine association rules of operations. The solutions for the two situations are just our starting inspirations, we will focus on extracting the transaction functions and evaluate the extraction results in our future work.

7.2. Expectations for researches on automated function point analysis

After interviewing the FPA experts from our industrial partner, we summarize the following research questions on automated FPA.

How to automatically extract functions from traditional software system prototypes? Although we resolved the automated function extraction from requirements, there are still a large number of projects using system prototypes that are difficult to extract functions. How to automatically extract functions from those artifacts is still a big challenge.

How to predict the final size of function points based on product/project information and textual artifacts in the early phase? Our approach addresses the first step of function point counting that is the automated extraction of DFs based on requirements. Given the DFs, FPA practitioners need to analyze the complexity and system characteristics to estimate the final size or budget for the projects, which is a complicated and high-cost task. It will be a great help if researchers can work out a prediction model that can analyze product/project information and textual artifacts in the early phase, and predict the final results of the FPA.

How to automatically review functions identified by development teams based on FPA rules? For large organizations that use FPA standards to assess the contributions of IT teams, the functions identified by the development team members are likely to be overestimated or incorrect. There is a need for an automated review approach on functions that can prompt those inappropriate functions, such as duplicated functions, misclassified functions on types, and so on.

7.3. Lessons learned

Dealing tasks with few labeled data. Thanks to the advances of machine learning techniques, many related tasks have now reached impressive performance [32,45,46]. However, they are mostly supervised learning which requires massive labeled data, incurring significant labeling effort [47]. Compared to labeled data, it is much easier to obtain an amount of historical data without labeling. In our practice, we found that historical unlabeled requirements are also valuable. Besides, there are also lots of freely accessible data sources such as Wikipedia, StackOverflow, and Github, which have been applied to improve the performance of specific tasks such as software text retrieval [48], requirements term extraction [13], and text classification [49]. Therefore, when we only have limited labeled resources, it is a potential choice to consider using open corpora to improve performance.

Sequence modeling instead of bag-of-words modeling. There are many software artifacts written in natural languages, such as software requirements specifications, source code documents, issue reports, app descriptions, and so on. When dealing with tasks such as text classification, information retrieval, these natural language artifacts are usually modeled using bag-of-words (BOW). BOW puts all words in a bag, regardless of their morphology and word order, that is, each word is independent. Considering that natural language is a natural sequence of words, in which each word carries rich grammatical and contextual information, it is a better choice to model the natural language articles into sequences rather than BOW. The maximum entropy model, hidden Markov model, and CRF are all alternatives for sequence modeling. Besides, if there are a large number of labeled data to train the model, recurrent neural networks like LSTM [50] and GNU [51] could also be taken into consideration.

Handling out-of-vocabulary problem in requirements analysis. In our practice, we found that some textual requirements are incorrectly parsed by NLP toolkits. After analyzing the incorrect cases, we noted that most cases are related to business terms that are named by the company according to the specific business scenarios. It is known as the out-of-vocabulary (OOV) problem [52] in NLP. This is due to that the toolkits are all trained on the general corpus, such as WordNet[10] and OANC[11] which mainly contains a common vocabulary of terms. Especially, the OOV problem is particularly obvious in the languages which are not naturally segmented by spaces, such as Chinese, Korean, and Japanese. The errors of NLP toolkits can be introduced into the approaches built upon the general NLP techniques, which could also influence their performance. To solve the OOV problem, almost all the NLP toolkits have provided interfaces to import custom dictionaries to help parse the texts. However, this solution requires lots of human efforts to carefully review a large number of documents to ensure the high quality of dictionaries, which is cost-consuming. Another solution is to leverage the learning-based approaches, which are not totally built upon NLP toolkits and have been verified to handle the OOV problem in NLP tasks effectively [53–55].

7.4. Validity

External validity. The external threats are related to the generalization of the proposed approach. First, we experimented with the data taken from CMB. The results may be different from other scenarios. However, we train the model and evaluate the performance on the requirements from 20 systems, which could reduce this threat. Second, the studied subjects in this study are requirements only, which may not be appreciated by other artifacts like app reviews and code documentation. However, DEX extracts DFs from sentences and does not utilize unique characteristics except for natural language description, which could alleviate the threat. Third, our corpus is written in Chinese. Due to the differences in linguistic morphology, the proposed approach may not be suitable for other languages. While, when switching to other languages, the method only needs to be slightly adjusted in data preprocessing. For example, when dealing with English, the approach could work after performing some preprocessing steps such as stemming each word into its morphological root.

[10] https://wordnet.princeton.edu/.
[11] http://www.anc.org/data/oanc/.

Internal validity. The internal threats relate to experimental errors and biases. First, we evaluate the model with a few labeled requirements. The limited requirements for evaluation may introduce randomness of the performance. Second, when investigating the training cost, we collect the training time in minutes using Python embedded timer. The bias of collected minutes may be introduced to the training cost. However, we repeat each experiment 10 times and use the average of the 10 experiments as the final performance, which could alleviate the threats.

Construct validity. The construct threats relate to the suitability of evaluation metrics. We use *Precision*, *Recall* and *F1* as the measurement metrics to evaluate the performance of DEX. We consider DEX could be effectively applied to industrial practice based on the measurement metrics. It would be better if we investigated the cost savings or the effort savings to evaluate the performance of DEX. However, the *Precision*, *Recall* and *F1* are the commonly-used metrics in the machine learning and the information retrieval fields [40]. Moreover, we compare the DEX with state-of-the-art approaches and developers respectively using these metrics, which could indicate the advantages of the DEX.

Conclusion validity. The conclusion threats relate to the appropriateness of the conclusion. In the RQ4, we adopt the greedy strategy and empirically choose 15 combinations to tune the three bootstrapping parameters. The corresponding best result of the 15 combinations is considered as the final performance of DEX. However, the optimal one may not exist in the 15 parameter combinations. Accordingly, the results reported in RQ1 may not be the best performance that DEX could achieve. However, it is impossible to enumerate all the parameter combinations, and the adopted strategy could reach promising results. Thus, the threat could be alleviated.

8. Conclusion

In function point analysis, extracting data functions is an important and fundamental step. In industrial practices, the extraction is typically performed in the manual way, which is the effort-intensive activity with personal bias. This paper proposes a novel approach to extract data functions from textual requirements automatically. It takes the data function extraction as a sequence tagging problem and builds the CRF model that could consider the contextual information for accurate data function extraction. It adopts semi-supervised learning that can make use of the bootstrapping based algorithm to resolve the few labeled data issue. Besides, it builds a DF-oriented language model based on historical data functions to improve extraction accuracy. We evaluated the approach in the dataset from a real industrial environment. It can reach 80% precision, 84% recall, and 82% F1 respectively, and outperform the three state-of-the-art baselines. The results showed that our approach could retrieve most of the accurate data functions. Furthermore, we verified the value of unlabeled data and provided an effective practice to make use of these data, which could be an inspiration on how to build the model with few labeled data.

The data function extraction is the starting step for the automated function point analysis. In the future work, we will continue to design the approaches to automate other steps in function point analysis, such as transactional function extraction, function type classification and adjustment factor determination. We are closely collaborating with our industrial partner and has deployed the related tool online. Returned results will further validate the effectiveness, as well as guide us in improving our approach.

CRediT authorship contribution statement

Mingyang Li: Conceptualization, Methodology, Software, Validation. **Lin Shi:** Writing – review & editing. **Yawen Wang:** Data curation, Writing – original draft. **Junjie Wang:** Writing – review & editing. **Qing Wang:** Writing – review & editing. **Jun Hu:** Writing – review & editing. **Xinhua Peng:** Data curation, Writing – reviewing. **Weimin Liao:** Data curation, Writing – reviewing. **Guizhen Pi:** Data curation, Writing – reviewing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key Research and Development Program of China under grant No. 2018YFB1403400, China Merchants Bank, the National Science Foundation of China under grant No. 61802374, No. 61432001, No. 61602450.

References

[1] J.E. Matson, B.E. Barrett, J.M. Mellichamp, Software development cost estimation using function points, IEEE Trans. Softw. Eng. 20 (4) (1994) 275–287.
[2] M. Bundschuh, C. Dekkers, IFPUG Function Point Counting Rules, Springer Berlin Heidelberg, 2008, pp. 453–482.
[3] International Organization for Standardization, Information Technology, Software Measurement, Functional Size Measurement: Definition of Concepts, ISO/IEC, 2007.
[4] N.A.Z. Adem, Z.M. Kasirun, Automating function points analysis based on functional and non functional requirements text, in: 2010 the 2nd International Conference on Computer and Automation Engineering (ICCAE), Vol. 5, 2010, pp. 664–669.
[5] L. Shi, M. Li, M. Xing, Y. Wang, Q. Wang, X. Peng, W. Liao, G. Pi, H. Wang, Learning to extract transaction function from requirements: an industrial case on financial software, in: ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020, ACM, 2020, pp. 1444–1454.
[6] G.C. Low, D.R. Jeffery, Function points in the estimation and evaluation of the software process, IEEE Trans. Softw. Eng. 16 (1) (1990) 64–71.
[7] A. Hira, B.W. Boehm, Function point analysis for software maintenance, in: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016, Ciudad Real, Spain, September 8-9, 2016, 2016, pp. 48:1–48:6.
[8] M. Bundschuh, C. Dekkers, The IFPUG Function Point Counting Method, Springer Berlin Heidelberg, 2008, pp. 323–363.
[9] S.J. Baek, J.S. Han, Y.J. Song, Security threat modeling and requirement analysis method based on goal-scenario, in: Proceedings of the International Conference on IT Convergence and Security 2011, 2012.
[10] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer, 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, 2016, pp. 260–270.
[11] J.P.C. Chiu, E. Nichols, Named entity recognition with bidirectional LSTM-CNNs, Comput. Sci. (2016).
[12] M. Li, Y. Yang, L. Shi, Q. Wang, J. Hu, X. Peng, W. Liao, G. Pi, Automated extraction of requirement entities by leveraging LSTM-CRF and transfer learning, in: IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020, IEEE, 2020, pp. 208–219.
[13] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, Automated extraction and clustering of requirements glossary terms, IEEE Trans. Softw. Eng. 43 (10) (2016) 918–945.
[14] T. Gemkow, M. Conzelmann, K. Hartig, A. Vogelsang, [IEEE 2018 IEEE 26th International Requirements Engineering Conference (RE) - Banff, AB, Canada (2018.8.20-2018.8.24)] 2018 IEEE 26th International Requirements Engineering Conference (RE) - Automatic Glossary Term Extraction from Large-Scale Requirements, 2018, pp. 412–417.
[15] J. Lafferty, A. McCallum, F.C. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, 2001.
[16] B. Settles, M. Craven, An analysis of active learning strategies for sequence labeling tasks, in: Conference on Empirical Methods in Natural Language Processing, 2008.
[17] A. Ratnaparkhi, A maximum entropy model for part-of-speech tagging, in: E. Brill, K. Church (Eds.), Conference on Empirical Methods in Natural Language Processing, EMNLP 1996, Philadelphia, PA, USA, May 17-18, 1996, 1996.
[18] A.B. Poritz, Linear predictive hidden Markov models and the speech signal, in: IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '82, Paris, France, May 3-5, 1982, IEEE, 1982, pp. 1291–1294.
[19] T.D. Singh, A. Ekbal, S. Bandyopadhyay, Manipuri POS tagging using CRF and SVM: A language independent approach, in: Proceeding of 6th International Conference on Natural Language Processing (ICON-2008), 2008, pp. 240–245.

[20] Z. Huang, W. Xu, K. Yu, Bidirectional LSTM-CRF models for sequence tagging, 2015, arXiv preprint arXiv:1508.01991.
[21] N. Greenberg, T. Bansal, P. Verga, A. McCallum, Marginal likelihood training of BiLSTM-CRF for biomedical named entity recognition from disjoint label sets, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 2824–2829.
[22] X.J. Zhu, Semi-Supervised Learning Literature Survey, Technical Report, University of Wisconsin-Madison Department of Computer Sciences, 2005.
[23] U. Kanimozhi, D. Manjula, A bootstrapping approach for entity linking from biomedical literature, Polytech. Open Libr. Int. Bull. Inf. Technol. Sci. 56 (2017) 53–58.
[24] J.A. Pow-Sang, L. Gasco, A. Nakasone, A function point logic file identification technique using UML analysis class diagrams, in: Advances in Software Engineering - International Conference on Advanced Software Engineering and its Applications, ASEA 2009 Held as Part of the Future Generation Information Technology Conference, FGIT 2009, Jeju Island, Korea, December 10-12, 2009. Proceedings, 2009, pp. 160–167.
[25] A.R. Irawati, K. Mustofa, Measuring software functionality using function point method based on design documentation, Int. J. Comput. Sci. Issues 9 (3/1) (2012) 124–130.
[26] T. Edagawa, T. Akaike, Y. Higo, S. Kusumoto, S. Hanabusa, T. Shibamoto, Function point measurement from web application source code based on screen transitions and database accesses, J. Syst. Softw. 84 (6) (2011) 976–984.
[27] M.A. Sag, A. Tarhan, Measuring COSMIC software size from functional execution traces of java business applications, in: Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on, IEEE, 2014, pp. 272–281.
[28] D. Bourigault, Surface grammatical analysis for the extraction of terminological noun phrases, in: Proceedings of the 14th Conference on Computational Linguistics-Volume 3, Association for Computational Linguistics, 1992, pp. 977–981.
[29] A. Dwarakanath, R.R. Ramnani, S. Sengupta, Automatic extraction of glossary terms from natural language requirements, in: 2013 21st IEEE International Requirements Engineering Conference (RE), IEEE, 2013, pp. 314–319.
[30] P.A. Ménard, S. Ratté, Concept extraction from business documents for software engineering projects, Autom. Softw. Eng. 23 (4) (2016) 649–686.
[31] T. Johann, C. Stanik, M. Alireza, B. Alizadeh, W. Maalej, SAFE: A simple approach for feature extraction from app descriptions and app reviews, in: Requirements Engineering Conference, 2017.
[32] J.R. Finkel, T. Grenager, C. Manning, Incorporating non-local information into information extraction systems by gibbs sampling, in: Meeting on Association for Computational Linguistics, 2005.
[33] R. Huang, E. Riloff, Inducing domain-specific semantic class taggers from (almost) nothing, in: Meeting of the Association for Computational Linguistics, 2010.
[34] D.S. Sachan, P. Xie, E.P. Xing, Effective use of bidirectional language modeling for medical named entity recognition, 2017, CoRR abs/1711.07908.
[35] K. Jones S, A statistical interpretation of term specificity and its application in retrieval, J. Doc. (2004).
[36] H.J. Dai, P.T. Lai, Y.C. Chang, T.H. Tsai, Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization, J. Cheminform. 7 (S1) (2015) S14.
[37] R. Rosenfeld, Two decades of statistical language modeling: Where do we go from here? Proc. IEEE 88 (8) (2000) 1270–1278.
[38] P.F. Brown, V.J.D. Pietra, R.L. Mercer, S.A.D. Pietra, J.C. Lai, An estimate of an upper bound for the entropy of English, Comput. Linguist. 18 (1) (1992) 31–40.
[39] C. Tantithamthavorn, ScottKnottESD: The scott-knott effect size difference (ESD) test, 2017.
[40] D.M. Berry, J. Cleland-Huang, A. Ferrari, Maalej, Panel: Context-dependent evaluation of tools for NL RE tasks: Recall vs. precision, and beyond, in: Requirements Engineering Conference, 2017.
[41] T. Zhang, F. Damerau, D. Johnson, Text chunking based on a generalization of winnow, J. Mach. Learn. Res. 2 (4) (2002) 615–637.
[42] S.J. Pan, Y. Qiang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[43] A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, in: European Conference on Principles of Data Mining & Knowledge Discovery, 2000.
[44] H. Li, Y. Wang, D. Zhang, M. Zhang, E.Y. Chang, Pfp: parallel fp-growth for query recommendation, in: Acm Conference on Recommender Systems, 2008.
[45] A. Di Sorbo, S. Panichella, C.V. Alexandru, J. Shimagaki, C.A. Visaggio, G. Canfora, H.C. Gall, What would users change in my app? summarizing app reviews for recommending software changes, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2016, pp. 499–510.
[46] W. Maalej, Z. Kurtanović, H. Nabil, C. Stanik, On the automatic classification of app reviews, Requir. Eng. 21 (3) (2016) 311–331.
[47] V.T. Dhinakaran, R. Pulle, N. Ajmeri, P.K. Murukannaiah, App review analysis via active learning, in: International Requirements Engineering Conference, 2018.
[48] Z. Lin, Y. Zou, J. Zhao, B. Xie, Improving software text retrieval using conceptual knowledge in source code, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2017, pp. 123–134.
[49] J. Howard, S. Ruder, Universal language model fine-tuning for text classification, 2018, arXiv preprint arXiv:1801.06146.
[50] F.A. Gers, E. Schmidhuber, LSTM recurrent networks learn simple context-free and context-sensitive languages, IEEE Trans. Neural Netw. 12 (6) (2001) 1333–1340.
[51] R. Dey, F.M. Salemt, Gate-variants of gated recurrent unit (GRU) neural networks, in: IEEE International Midwest Symposium on Circuits and Systems, 2017.
[52] L. Qin, Learning Out-of-Vocabulary Words in Automatic Speech Recognition (Ph.D. thesis), Citeseer, 2013.
[53] W. Ling, I. Trancoso, C. Dyer, A.W. Black, Character-based neural machine translation, 2015, arXiv preprint arXiv:1511.04586.
[54] A. Maas, Z. Xie, D. Jurafsky, A. Ng, Lexicon-free conversational speech recognition with neural networks, in: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2015, pp. 345–354.
[55] W. Chan, N. Jaitly, Q. Le, O. Vinyals, Listen, attend and spell: A neural network for large vocabulary conversational speech recognition, in: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 4960–4964.