Learning to Extract Transaction Function from Requirements: An Industrial Case on Financial Software

Lin Shi
Mingyang Li
Mingzhe Xing
Yawen Wang
(shilin,mingyang,mingzhe,yawen)@itechs.iscas.ac.cn
Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

Qing Wang
wq@iscas.ac.cn
Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

Xinhua Peng
Weimin Liao
Guizhen Pi
(joeyxhpeng,liaowm,p98119)@cmbchina.com
China Merchants Bank, Shenzhen, China

Haiqing Wang
wanghq@bscea.org
Beijing Software Cost Evaluation Technology Innovation Alliance, Beijing, China

ABSTRACT

In practice, it is very important to determine the size of a proposed software system yet to be built based on its requirements, i.e., early in the development life cycle. The most widely used approach for size estimation is Function Point Analysis (FPA). However, since FPA involves human judgment, the estimation results are some degree of subjective, and the process is labor and cost intensive. In this paper, we propose a novel approach to automatically identify transaction functions from textual requirements by leveraging a set of natural language processing techniques and machine learning models. We evaluate our approach on 1,864 requirements and 104,691 transaction functions taken from 36 financial projects from one banking industry. The results show that the contents of the suggested transaction functions by our approach are high in quality, with low perplexity value of 8.5 and high BLEU score of 34 on average. The types of suggested transaction functions can also be accurately classified, with overall accuracy of 0.99 on average. Our approach can provide reasonable suggestions that assist industrial practitioners to identify transaction functions faster and easier.

CCS CONCEPTS

• **Software and its engineering** → **Requirements analysis**; **Software implementation planning**.

KEYWORDS

Requirements, Function Point, Machine Learning

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3417053

ACM Reference Format:
Lin Shi, Mingyang Li, Mingzhe Xing, Yawen Wang, Qing Wang, Xinhua Peng, Weimin Liao, Guizhen Pi, and Haiqing Wang. 2020. Learning to Extract Transaction Function from Requirements: An Industrial Case on Financial Software. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3368089.3417053

1 INTRODUCTION

In industrial practice, it is desirable to have a reliable cost estimation available before software systems are actually built. One of the most intensively used and mature approaches to estimate the software size in the early stage is Function Point Analysis (FPA), which has been well developed over 30 years, and also has been adopted as international standards, e.g. ISO 20926 [2, 19, 29]. FPA analyzes the "elementary process" of a software system from user requirements, which is the smallest unit of activity that is meaningful to the users. FPA provides a measure of size in Function Points (FP), and the definition is: "*A function point is 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*"[19]. Based on the counted FPs, reasonable size estimations can be calculated using FPA, and indirect cost estimations can be calculated.

Unfortunately, FPA has its own unsolved problems that prevent it from being a valid measure with widespread acceptance. One significant problem with FPA is that it needs to identify FPs from requirements documents first. These documents are often totaling hundreds of pages, and the process heavily relies on human measurers to read those documents. Another problem is that the estimation results are some degree of subjective. Human measurers need to follow a set of standard FP counting rules, which are open to interpretation on many occasions, thus they often produce inconsistent estimation results. Differences in the same product may occur even in the same organization. For example, Low and

Jeffery [30] reported that a 30-percent variance was caused within one organization and more than a 30-percent variance was caused across organizations.

Therefore, in order to obtain consistent estimations and reduce costly human involvement, it is important to propose approaches to promote the automation, effectiveness, and quality of function points analysis. However, one of the most challenging problems faced by such automated FPA approaches is extracting functions from natural-language requirements. We employ an example in Figure 1 to illustrate why it is challenging. The extracted functions are "receive Alipay payment confirmation", and the four words are scattering separately in the requirement text. It is difficult to efficiently extract functions due to three reasons. (1) The functions cannot be simply treated as the part-of-speech patterns with consecutive words from the requirement descriptions. There are several researches focusing on the automatic concept extraction with part-of-speech tagging and heuristic rules (i.e., *adjective-noun-noun*) [24, 26, 42, 45]. These techniques can not be directly utilized in this task because the functions are far from the pattern-based phrases. (2) Extracting functions from requirements need to be performed in compliance with rules defined in FPA methods. (3) The documents provided for function extraction are uneven in quality and readability since they are mostly written in natural language by different people, which results in the extraction activity even more difficult.

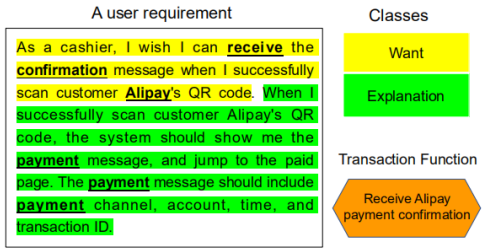

Figure 1: An example of requirement sentence classification and the corresponding transaction function.

In this paper, we propose a novel approach to automatically identify transaction functions by understanding the natural-language user requirements. First, we extract six categories of features for each word in requirements, based on text parser, heuristic rules, word embedding model, and lexical graph. Second, we train word predictors with requirement corpus, and select the optimal one to predict words in functions. Finally, we construct transaction functions by combing words, ranking by perplexity, and classifying function types. The output of our approach is a list of suggested transaction functions extracted from the textual requirement together with their function types, which are External Inquery (EQ), External Output (EO), or External Input (EI). The three types are different elementary processes that reflect how the data or control information interact with the application [19]. We evaluate our approach on 36 financial projects taken from real banking industry. The results show that the contents of the suggested transaction functions are high in quality, with low perplexity value of 8.5 and high BLEU score of 34 on average. The types of suggested transaction functions can also be accurately classified, with overall accuracy of 0.99 on average.

The major contributions of this paper are as follows.

- We propose an automated solution to identify transaction functions from textual requirements.
- We design six categories of features, as well as their calculation methods, to characterize words in requirements, acting as foundations for advanced requirement analysis.
- We evaluate our approach on 36 financial projects taken from real banking industry, and the results are promising.
- We share the learned lessons from deploying our approach in the real industrial settings and expectations for researches on automated function points analysis.

2 FPA IN FINANCIAL SOFTWARE

2.1 Financial Software in CMB

China Merchants Bank (CMB) is the largest joint-stock commercial bank wholly owned by corporate legal entities in China, as well as one of the top 500 companies in the world. 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 1,800 branches worldwide. To support its widespread business, the IT department of CMB develops and maintains over 500 financial software, which involves huge budgets and employees.

Most of the financial software are transaction-oriented applications with data persistency, which are well adapted to functional sizing. CMB has utilized FPA to estimate and measure system size over 10 years. In addition, many other financial organizations in China also utilize FPA methods for early cost estimation. Among those organizations, CMB is the best pioneer to lead the progressing of Chinese FPA industrial practice.

2.2 FP Counting Practice in CMB

Following introduces the process of applying FPA in CMB. First, the FPA process launches when user requirements are collected and documented. Guided by the official Function Point Counting Practice Manual [19], product managers identify data functions and transaction functions from user requirements. Here data function represents functionality to meet internal and external data storage requirements, and transaction function represents the elementary process that provides functionality to the user to process data, including external input, external output, and external query. Specifically, data functions are domain-specific data entities that are obviously mentioned in the requirements, while transaction functions are highly summarized activities that are meaningful to users, e.g. "Add a new customer", "Report customer purchases", and "Change customer details". Therefore, it will be more difficult to summarize transaction functions than mark out data functions from requirement texts. Moreover, transaction functions take up a majority of all functions. As reported by IFPUG [19], nearly 80% of the functions are the transaction ones. Thus, summarizing transaction functions from requirements is the most time-consuming and lengthy activity in FPA practices.

Second, due to current FPA practice heavily relying on human judgment to estimate project cost, it is essential to make sure that the manually identified results are correct. Therefore, when the product managers submit identified functions, the FP experts will review their submissions and correct mistakes. We summarize the common types of incorrect functions during the expert review. As shown in Table 1, the most common issue is using a part of individual elementary process as a transaction function, e.g. identifying one step "enter passwords" rather than the entire process "user login". Breaking the elementary process will result in budget overestimate. The second most common issue is duplicate functions. The third most common issue is identifying elementary process or data entity that is meaningless to users.

Third, with the functions that have been approved by experts, the CMB experts put the number of functions into a pre-defined formulation guided by official Function Point Counting Practice Manual [19] to calculate the adjusted function points, and finally the budget and schedule.

In the FP process, CMB suffers from subjective deviations caused by human judgment and expensive expert involvement. Therefore, an automated solution to identify FPs is badly desired in the Banking industry, such as CMB, that needs to develop and maintain a large scale of financial software.

Table 1: Common Issue Types in Function Review

Issue Type	Percentage
Breaking elementary process	38.38%
Duplicate	28.62%
Meaningless to users	20.54%
Misclassification of transaction functions	6.57%
Missing functions	4.55%
Incorrect data function	1.35%

3 RELATED WORK

Due to the significant challenges in understanding natural-language requirements, very little research has addressed the automated extraction of functions from textual requirements. Adem et al. [1] 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 to the software industry with 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.

Several studies have investigated the automation of FP measurement from design specifications and source code. For example, Pow-Sang et al. [38] identified function point logic files from UML class diagrams that made use of association, composition, generalization, and association-class relationships. Uemura et al. [44] proposed a series of function-point analysis measurement rules using design specifications based on the Unified Modelling Language (UML) and described a function-point measurement tool, whose inputs are design specifications developed on Rational Rose. Irawati et al. [22] 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. Lamma et al. [28] presented a tool for measuring FP from Entity Relationship (ER) diagrams and a Data Flow Diagrams (DFD). Besides design specifications, researchers also explored the automation of FP measurement from source code. For example, Edagawa and Akaike et al. [15] proposed a method to automatically identify data and transaction functions from Web applications using static code analysis. Ali SAG and Tarhan [41] derived UML sequence Diagrams from functional execution traces at runtime with the help of AspectJ technology, and utilized it to measure the functional size. Kusumoto et al. [27] proposed measurement rules to count data and transaction functions for object-oriented program based on IFPUG method and examining the method to practical Java programs. They reported that although they got similar number of data and transaction functions, disagreement on classification of functions still existed between the tool and the specialist.

Existing automated FPA studies mainly targeted towards design specifications and source code artifacts, while very little research has addressed the automated extraction of functions from textual requirements. Since learning-based approaches have gain significant success in the field of NLP, we utilize such techniques to automatically extract transaction functions from requirements text, which can facilitate the automatic estimation of the systems size in the early stage.

4 APPROACH

Our approach is inspired by the keyphrase extraction approaches in NLP, which generally consist of three main procedures: generating candidate phrases, building the model, and predicting keyphrases [34]. Extracting FP can be divided into two procedures. First, we extract words from requirements, and then we construct FP phrases. Specifically, the proposed approach consists of three main steps (Figure 2): (1) extract six categories of features for each word in requirements, based on text parser, heuristic rules, word embedding model, and lexical graph; (2) predict words that will appear in transaction functions by training a word predictor; and (3) construct transaction functions by combing words, ranking in perplexity, and classifying function types.

4.1 Extract Features

By analyzing the words in user requirements, we consider features indicating lexical characteristics, syntactic characteristics, semantic characteristics, and historical records could contribute to predict functions in requirements descriptions. Therefore, we define six categories of features for words: Term Features, Location Features, Frequency Features, History Features, Representative Features, and Importance Features as shown in Table 2. Here we introduce the six categories of features:

Term Features (TERM) describe the basic linguistic characteristics of each word in one requirement. In this study, we use Part-of-speech tags and universal dependencies as term features (4.1.1).

Location Features (LOC) focus on the positions of the words inside one requirement. First, we utilize a set of heuristic rules to classify requirement sentences into 'want' or 'explanation' category

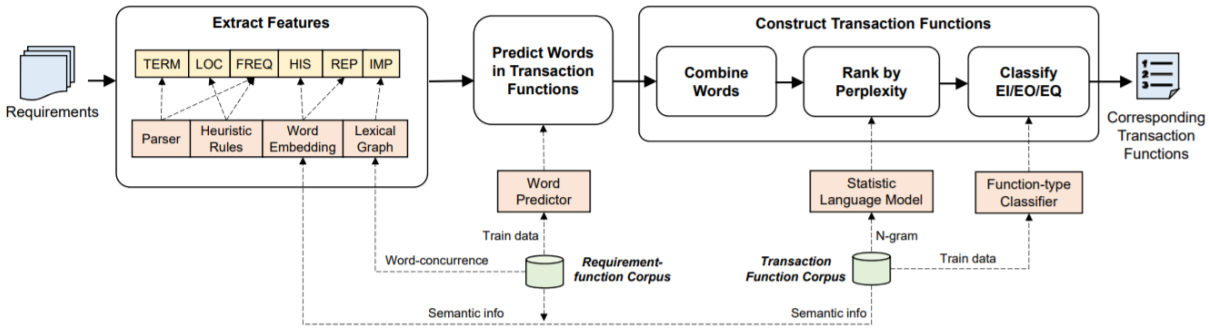

Figure 2: The approach overview.

(4.1.2). Then we define three features, WS, ES, and ON, based on the sentence that the given word belongs to. WS and ES denote whether the word appears in 'want' or 'explanation' sentences respectively. ON denotes the order number of the first sentence that the word appears in.

Frequency Features (FREQ) reflect the term frequency in sentence level and requirement level, including TF, IDF, WF, and EF. TF is term frequency in the requirements, and IDF is the inverse document frequency in the project scope. WF and EF denote the term frequency in 'want' or 'explanation' sentences.

History Features (HIS) indicate whether the word is frequently used in historical functions, including FH and SFH. FH denotes the word frequency in history functions. SFH enlarges the basic term frequency of the word with its synonyms by utilizing word embedding techniques [17] (4.1.3).

Representative Features (REP) illustrate the distance between the given word and the root nouns/verbs extracted from the 'want' sentences: the distance in the syntax tree (DHV), and the semantic distance (SV, SN, and SW).

Importance Features (IMP) are calculated from a lexical graph of requirement words by utilizing text summarization techniques and centrality measurements (4.1.4). We apply two text summarization techniques, PageRank and TextRank, to provide scores for words (PR and TRS), as well as four centrality measurements for the lexical graph: Degree Centrality (DC), Betweenness Centrality (BC), Closeness Centrality (CC), and Eigenvector Centrality (EC).

We utilize four techniques to automatically calculate the six categories of features: (1) text parser to calculate the *Term* features and *Frequency* features; (2) heuristic-rule based sentence classification to support *Location* and *Frequency* features; (3) word-embedding based similarity to support *History* and *Representative* features; and (4) word-concurrence lexical graph for *Importance* features.

4.1.1 Text Parser. We implement an automated text parsing tool that integrates with the Stanford NLP parser [8] to process the textual requirements. We directly obtain the POSTAG [43] and DEPREL [11] from Stanford parser. The POSTAG reflects parts of speech to each word, such as noun, verb, adjective, etc. The DEPREL denotes the grammatical relations between individual words, such as nominal subject, direct object, and auxiliary. The text parser also calculates TF and IDF when processing the textual requirements based on the fundamental definition of the two metrics [25].

4.1.2 Heuristic-Rule Based Sentence Classification. In CMB, user requirements are written in free-format natural language, and usually consist of user expectations and detail explanations. As user expectations briefly summarize what the user wants from the system, the sentences expressing expectations might imply transaction functions that summarize system functionalities. Besides the user expectations, other sentences are considered as detail explanations in this study.

In order to distinguish the two types of sentences, we utilize the heuristic linguistic rules provided by Di Sorbe et al. [14] for discovering requirements. Considering the expressing habits in the CMB settings, we select eight heuristic rules from Di Sorbe's research, and add a new rule(#9) for CMB as listed below.

(1) All [someone] needs are [something]
(2) [someone] would love/like to use [something]
(3) [someone] wants/likes to have/use [something]
(4) It would be [modal] to have [something]
(5) [someone] expects [something] to work
(6) [someone] thinks [something] would be [modal] to have [something]
(7) [someone] thinks [something] need [verb]
(8) [something] should/could support [something]
(9) As [role], I wish/want [something]

We classify sentences that match the nine rules as 'want', and other sentences as 'explanation'. Figure 1 shows an example of requirement sentence classification. We can see that, this requirement contains three sentences. The first one matches with rule #9, thus it is classified as 'want'. The other two are detailed explanations about how the system behave, and are classified as 'explanation'. We manually inspect 100 sentences, and 98% are correctly classified.

4.1.3 Word-Embedding Based Similarity. When calculating the semantic similarity for the above features, we use a word-embedding model [17] trained by all the text, including 1,864 textual requirements and 107,256 (104,691+2,565) history functions from the industrial repository. We conduct word segmentation and remove stop-words (i.e., "on", "the", etc.) to reduce noise. After that, we obtain a sequence of words for each sentence. We then use the publicly available software[1] to learn the word embedding model. We transform each word into a d-dimensional vector, where d is

[1]https://github.com/hankcs/HanLP/wiki/word2vec

Table 2: Features for words in requirement documents

Feature Type	Feature Name	Description	Value Range
Term	POSTAG	The result of Part-Of-Speech Tagger on the word	{Categories}
	DEPREL	The universal Stanford dependency relation to the word	{Categories}
Location	WS	Whether the word appear in the 'want' sentences	{True, False}
	ES	Whether the word appear in the 'explanation' sentences	{True, False}
	ON	The order number of the sentence that the word located in	{Positive Numbers}
Frequency	TF	The term frequency of the word in the requirement	(0, 1]
	IDF	the inverse document frequency of the word among all the requirements in one product	{Positive Numbers}
	WF	The term frequency of the word in the 'want' sentences	(0, 1]
	EF	The term frequency of the word in the 'explanation' sentences	(0, 1]
History	FH	The term frequency of the word among all the words in the historical transaction functions	(0, 1]
	SFH	The term frequency of the word and its similar words in the historical transaction functions	(0, 1]
Representative	DHV	The average of shortest path from the word to the root verbs in the 'want' sentences	{Positive Numbers}
	SV	The average similarity between the word and root verbs in the 'want' sentences	(0, 1]
	SN	The average similarity between the word and root nouns in the 'want' sentences	(0, 1]
	SW	The number of its similar words in the 'want' sentences	{Positive Numbers}
Importance	DC	Degree Centrality is the number of weighted edges incident upon a word vertex [4]	{Positive Numbers}
	BC	The Betweenness Centrality is the number of shortest paths traversing that word vertex [5]	[0, 1]
	CC	The Closeness Centrality is a measure of the proximity to the rest of the vertices in the network [9]	(0, 1]
	EC	The Eigenvector Centrality is a measure of the importance of a word vertex in the network [3]	{Positive Numbers}
	PR	The scores calculated with Page Rank Algorithm [33]	[0,10]
	TRS	The scores calculated with the TextRank Algorithm [32]	(0,1)

set to 100 as suggested in previous studies [46, 47]. The distance of two word vectors is their semantic similarity.

4.1.4 Lexical Graph for Word-Concurrence. Existing text summarization techniques utilize a lexical graph to extract important words from natural language texts, e.g. PageRank [33] and TextRank [32]. In this study, we build a word-concurrence graph for all non-stop words in requirements. Each node in the graph corresponds to a unique word in the requirements. To construct edges between nodes, we tokenize each sentence in the requirements into a sequence of words. Then, we build a window (w_{i-n}, ..., w_{i-1}, w_i, w_{i+1}, ..., w_{i+n}) for each word in the sequence, where w_n is the current word and w_{i-n}, w_{i+n} are the n^{th} words before and after w_n respectively. After that, we add edges between w_n and other words. With the lexical graph, the word importance is measured by calculating its centrality measurements.

4.2 Predict Words in Transaction Functions

Comparing with the corresponding actual functions, we can automatically obtain positive and negative labels to each word in the requirement according to whether they are included in the actual functions. With the features that characterizing words in requirements, we then select several commonly-used supervised machine learning methods to train corresponding models that can predict whether a given word will appear in the corresponding transaction functions or not. After comparing the performances of the selected methods, our approach chooses the optimal model with the highest performance as our word predictor.

In some cases, one word may appear multiple times in one requirement. Although these words are identical in textual format, their values of the six categories of features may be different, for example, the same words may have different locations in requirements or different results in the syntax tree. Thus the word predictor might give different results for the same words. In order to obtain a unified classification result, we further reconcile the word predictor by conducting a combination strategy that all the same words will be predicted to be positive on condition that there exists one positive predicting result. Given a word w, let $P = \{p_1, p_2, ..., p_n\}$ be the set of prediction results of all the n appearances of word w. Then the reconciled prediction result $f(w)$ is defined as below:

$$f(w) = \begin{cases} True, \exists\, x \subset P, x = True \\ False \end{cases} \quad (1)$$

4.3 Construct Transaction Functions

After we obtain the list of words that are predicted to appear in the corresponding transaction functions, we then construct these words into suggested transaction functions by three steps: combine words, rank in perplexity, and classify function types. Alg. 1 demonstrates the pseudocode of constructing transaction functions.

4.3.1 Combine Words. We define the format of function as "[Verb] + [Compound Noun]", where the compound nouns describe the detailed subjects, and the verb describes the action of the functions, e.g. "Display employee information". As shown in the COMBINE procedure in Alg. 1, for each verb and noun in the predicting words, we select its related noun or adjective in the corresponding sentences according to dobj DEPREL[2] (line 2-11). Then we obtain the compound noun, and combine the verb with the compound noun as one transaction function. Finally, we remove the duplicate combinations and output the functions (line 15).

4.3.2 Rank by Perplexity. To suggest functions with good readability in high priority for early review, we utilize a *N-gram* language model to rank the combined results. A N-gram language model can learn to predict the Perplexity of a sequence of words, thus can indicate the readability of a given phrase against the training corpus [7]. We train the language model based on 104,691 functions taken from CMB. We utilize a public library[3] to build the language model $model$. As shown in the RANK procedure in Alg. 1, for each combined result $function$, we calculate its perplexity score: $model(function)$. Then we rank the suggestions by the perplexity, and suggest functions with low perplexity in priority.

[2] dobj is the noun phrase which is the (accusative) object of the verb
[3] https://github.com/adampauls/berkeleylm

4.3.3 Classify Function Types. In the last step, we determine the three function types (EI, EO, or EQ) for each combined result. Four common text classification methods are considered for classifying the function types, i.e. Naive Bayes (NB), Logistic Regression (LR), J48, and Random Forest (RF). These four classifiers are also widely used in the previous studies [20, 21, 23]. We use the TF-IDF method to represent each word in functions, in which the frequencies of words are multiplied by their inverse document-frequency. We train these classifiers by the transaction function corpus. We select the optimal one with the highest performance as our function-type classifier (*model*). As shown in the CLASSIFY procedure in Alg. 1, we add the predicted function type to the combined transaction function $model(function)$.

Algorithm 1 Construct Transaction Functions

```
1: procedure COMBINE WORDS(List words)
2:     for each String word : words do
3:         s ← find sentence which contains the word
4:         if POS(word) is verb then
5:             coreNoun ← noun whose DEPREL is dobj in s
6:             cpNoun ← connect coreNoun with compound words in s
7:             functions.add(word + cpNoun)
8:         end if
9:         if POS(word) is noun then
10:            cpNoun ← connect word with words whose DEPREL is compound
    in s
11:            verb ← verb whose DEPREL is dobj in s
12:            functions.add(verb + cpNoun)
13:        end if
14:    end for
15:    functions ← deduplicates(functions)
16:    return functions
17: end procedure
1: procedure RANK(List functions, langModel model)
2:     for String function : functions do
3:         scoreList ← (model(function), function)
4:     end for
5:     rank scoreList by perplexity
6:     return scoreList.function
7: end procedure
1: procedure CLASSIFY(List newfunctions, classifyModel model)
2:     for String function : newfunctions do
3:         functionTypes.add(model(function))
4:     end for
5:     return functionTypes
6: end procedure
```

5 EXPERIMENTAL DESIGN

Our evaluation is guided by the following research questions:

RQ1: To what extent can the word predictor identify the words in the transaction functions? We compare and investigate the performances of the function-word predictors trained by different supervised machine learning approaches on the industrial requirement dataset.

RQ2: How effective is our approach in extracting transaction functions from requirements? To demonstrate the effectiveness of the proposed approach, we evaluate the correctness of the content of extracted functions in terms of BLEU, perplexity, and validity ratio, by comparing the text of extracted functions with the text of ground-truth functions that are manually extracted by human experts.

RQ3: How well can our approach correctly classify the types of transaction functions? We compare and investigate the performances of the function-type classifiers that are trained by different supervised machine learning approaches on the industrial function dataset.

5.1 Data Collection

Our experimental data is collected from the repositories of the IT department in CMB. We collaborate with the project management department of CMB, and help them to increase the level of automation in software effort estimation, which reduces expensive labor costs and difficulties in applying FPA methods. Our evaluation uses the following two datasets.

Table 3: Two Experiment Datasets

(a) Requirement-function Corpus

Time Span	2019.1-2019.12
#Req	1864
#Words	285543
#Projects	36
#Tranaction Functions	2565

(b) Transaction Function Corpus

Time Span	2010.1-2018.12
#EI	53237
#EQ	17131
#EO	34323
Total	104691

Requirement-function Corpus contains textual requirements and their corresponding functions, which are used to evaluate RQ1 and RQ2. Because the industry did not record the relationship between requirements and their corresponding functions until 2019, we select all the finished 36 financial systems from CMB repository from January 2019 to December 2019 as our studied projects. Requirements of these projects are all written in natural language, and we can retrieve the corresponding actual functions for each requirement in the corpus. First, we exclude requirements that are low in quality from three aspects: (1) The requirements that are incomplete. (2) The requirements that link to abnormal numbers of functions. In this study, we found that almost 90% requirements have less than 10 functions, and only 10% have more than 10 functions. Therefore, we consider requirements with more than 10 functions to be outliers. (3) The requirements with low readability that are hard to understand. Second, we retrieve the corresponding actual functions related with each requirement from the requirement corpus as ground-truth functions. To ensure the quality and validity of the ground-truth functions, we select the trustful ones according to three criteria: (1) manually extracted functions that have been double-validated by FPA experts to be correct; (2) the relationship between functions and corresponding requirements are explicitly designated; (3) 90% words of the related functions have appeared in the requirement texts. In total, we collect 1,864 requirements, involving 2,565 transaction functions and 285,543 words. The detailed information is shown in Table 3(a).

Transaction Function Corpus contains all the transaction functions in the industrial case from 2010.1 to 2018.12 (9 years). It is used to train and evaluate the function-type classifiers (RQ3). This corpus also acts as an important organizational asset that supports for new financial system requirement analysis and effort estimation. As shown in Table 3(b), we collect 104,691 transaction functions, including 53,237 EIs, 17,131 EOs, and 34,323 EQs.

5.2 Evaluation Measurements

We employ different measurements for evaluating suggested transaction functions.

Precision, **Recall**, **F1**, **Accuracy**, and **AUC** are commonly used measurements for performance assessment in classification tasks [39]. *Accuracy* is the number of correct predictions divided by the total number of predictions, which can reflect the overall performance of multi-classification prediction. *AUC* (Area Under roc Curve) is the area of the two-dimensional graph in which false positive rate is plotted on the X axis, and true positive rate is plotted on the Y axis [16]. AUC can avoid performance inflation when evaluating on imbalanced data. The AUC value varies between 0 and 1, and higher values indicate better performance.

Perplexity [6] measures how well a model predicts samples. Low (e.g. single digit) perplexity values indicate the model is good at predicting a given sequence, and the lower the better. The perplexity score of a sentence can be calculated by the following equation:

$$PPL(s) = 2^{-\frac{1}{N}\sum \log(p(w_i))} \quad (2)$$

Where N is the length of sentence, $p(w_i)$ is the score of i^{th} n-gram in sentence, and $\sum \log(p(w_i))$ is the log of n-gram language model score of a sentence.

BLEU is a well-known and popular metric for automatically evaluating the quality of machine-translated sentences [35]. It has been shown to correlate well with human judgments [10, 18]. BLEU calculates how well a given sequence is matched with an expected sequence in terms of the actual tokens and their ordering using an n-gram model. We apply BLEU to evaluate the suggested transaction functions. The output of the BLEU metric is a number between 1–100. The higher values denote the better matches between suggested functions and actual functions. For natural language translations, BLEU scores of 25–40 are considered high scores [35].

Validity Ratio. A suggested transaction function is considered valid if the BLEU value between the given function and its expected function is over 25. In this study, we define the percentage of valid suggested functions over the total number of suggested functions as *Validity Ratio*.

MAP@N (Mean Average Precision) and **Recall@N** are widely-used for evaluating the quality of ranked results in text retrieval [31]. *MAP@N* considers whether all of the relevant items tend to get ranked highly in the first N^{th} suggestions, where *Recall@N* considers to what degree the ground-truth items could be retrieved. In this study, we calculate the MAP@5 and Recall@5 of the ranked functions for each requirement, according to the valid functions which are over BLEU 25.

5.3 Evaluation Design

Experiment I (RQ1). When building the word predictor, we select four supervised machine learning (ML) approaches including Naive Bayes, Logistic Regression, J48, and Random Forest [40] to train the models since they are the popular and representative algorithms that are widely used in natural language processing tasks to solve a binary classification problem. We choose the parameter set of Random Forest which uses the adaptive depth of decision trees, and the number of features in a single decision tree is $\log_2(\#total\_features)+1$. We train the Random Forest model for 700 epochs until converging. We optimize the hyper-parameters of four ML approaches respectively and carry out a number of experiments. After comparing the performances of the four ML models, we select the best model with the highest performance as the final decision for the word predictor. To make sure that all the words in one requirement are only used either in train dataset or test dataset, we conduct the **10-fold cross validation** on the requirement level. Specifically, we first randomly partition all the requirements into 10 parts. One single part is retained as the testing data, and the rest 9 parts are used as the training data. Then we repeat 10 times.

Experiment II (RQ2). The goal of experiment II is to evaluate the correctness of the contents of the extracted functions, in terms of BLEU, perplexity, validity ratio, MAP@5, and Recall@5. To evaluate the qualities of suggested functions, we treat the ground truth functions as reference set and calculate the BLEU score of each suggestion. We apply Laplace smoothing to avoid zero-division when calculating BLEU score. Since the perplexity score of a sentence need to be derived by n-gram language model score, we train a n-gram language model on the transaction function corpus as illustrated in Table 3(b). Then we calculate the perplexity score based on the length and $N-gram$ score.

Experiment III (RQ3). The goal of experiment III is to evaluate the correctness of the classified types for the extracted functions. First, we conduct the 10-fold cross validation on the transaction function corpus, which contains 104,691 history transaction functions. We use the same four supervised ML approaches as stated in Experiment I (i.e., NB, LR, RF, and J48). Second, we train the four models on the nine-year transaction function corpus (2010.1-2018.12), and then test on requirement-function corpus (2019.1-2019.12). We evaluate the correctness of the classified function types in terms of precision, recall, F1-score, and overall accuracy.

6 RESULTS AND ANALYSIS

6.1 Effectiveness of Word Predictor (RQ1)

Figure 3 demonstrates the performances of the four word predictors with different ML methods over the 36 projects. The short lines are medians. We can see that the medians of the RF model are the highest in terms of Precision, Recall, F1, and AUC, which indicates that the RF model can work quite well on predicting whether words will be included in the functions than other methods. Therefore, we select the Random Forest model as our final word predictor. The average of precision, recall, and F1-score of the RF model are 94.9%, 88.6%, and 91.6% respectively. The overall accuracy is 0.94 on average among the 36 projects. We consider the performances of the word predictor are relatively good on account of utilizing multi-dimensional features and word embedding techniques. In addition, the model trains at a speed of about 500 requirements/40 minutes on a standard company server, which uses 8G RAM and 4 cores CPU. Thus, it is efficient and acceptable in an industry scenario.

Answers to RQ1: The word predictor can effectively predict whether the words in a requirement will be included in its corresponding transaction functions or not. The Random Forest approach significantly outperforms Naive Bayes, Logistic Regression, and J48. On average, the RF-based word predictor can reach 0.94 of the overall accuracy, which is a promising result.

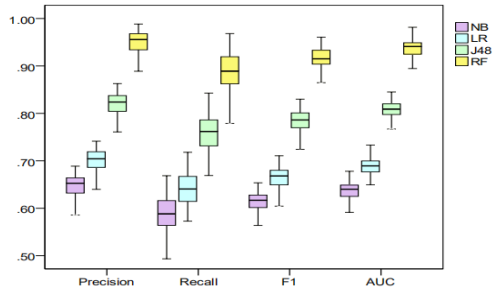

Figure 3: Performance comparison among ML approaches

6.2 Quality of Function Content (RQ2)

Figure 4(a) demonstrates the distribution of average BLEU values of the 36 projects. We can see that, most average BLEU values are between 30 and 40, with 34 of mean and 6 of standard deviation. Recall that high BLEU scores (i.e., 25-40) are desired for the task, the suggested transaction functions are highly similar to ground truth functions that are manually extracted by human experts. Figure 4(b) illustrates the distribution of average perplexity values. We can see that, most average perplexity values are between 7 and 10, with 8.48 of mean and 1.52 of standard deviation. Recall that low perplexity (i.e., single digits) is desired, thus the suggested transaction functions have good clarity, and are suitable for further reading and reviewing. Furthermore, we investigate the distribution of BLEU values on individual projects, as shown in Figure 5. We can see that all the medians are above 25 of BLEU, and most of suggested functions are also above 25, which indicates that the suggested functions have good matches with the actual functions among different projects.

Figure 6 shows the number of suggestions and validity ratios on each project. The average number of suggested functions is 121, and the average validity ratio is 66%. Most projects (26/36) have more than 60% valid suggestions. By further investigating the suggested functions for the 10 projects that are less than 60%, we find that, these projects are mainly related to data management systems, and heavily use high-appearance words in their functions,

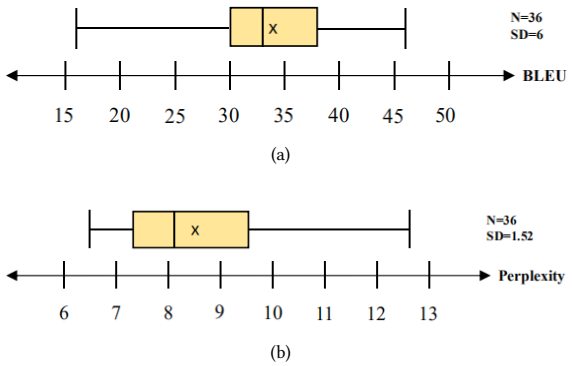

Figure 4: Distribution of BLEU and Perplexity over projects

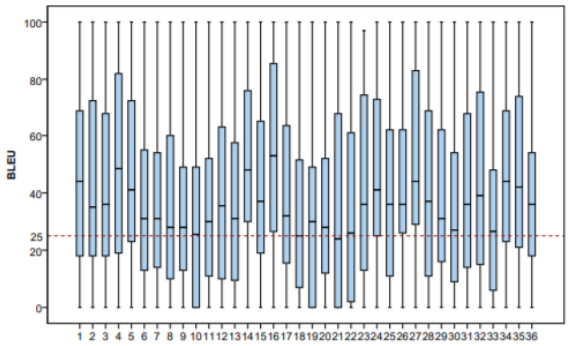

Figure 5: Distribution of BLEU on individual projects

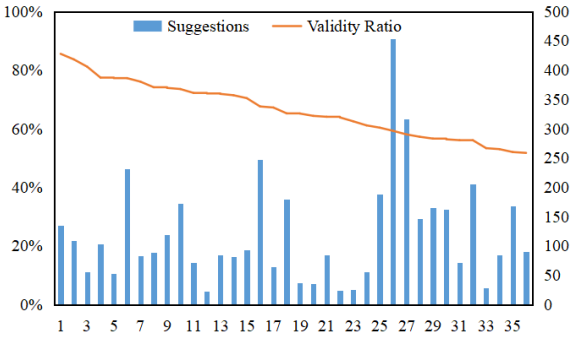

Figure 6: Distribution of validity ratios and total suggestions over 36 projects

such as "report" and "data". In such a case, although the word predictor can accurately predict most of the words in the ground-truth functions, the combine-word activity retrieves all the dobj DEPREL for those high-appearance words. Thus, many redundant suggestions are introduced for those high-appearance words in the 10 projects, which makes the validity ratios become lower. To further improve the percentage of valid suggestions, we plan to improve the combine-word activity by retrieving limited DEPREL instead of all DEPREL.

Figure 7 demonstrates the value of *MAP@5* and *Recall@5* of the suggested functions at project level. Nearly all the projects can reach a relatively good level of performance on *Recall@5*, and the average *Recall@5* is 0.94. Meanwhile, these projects also perform well in term of *MAP@5*. Most of the projects (34/36) can reach 0.5 on *MAP@5*, and the average *MAP@5* is 0.6 among the 36 projects.

Answers to RQ2: The contents of the transaction functions suggested by our approach are high in quality, in terms of accurate sequences, good clarity, and valid suggestions. Most of relevant functions could be recalled, and the recalled functions are suggested in priority. Specifically, our approach reaches a low perplexity value of 8.5 and high BLEU scores of 34 on average. Among all the suggested transaction functions, 66% are valid on average. The average MAP@5 and Recall@5 are 0.6 and 0.94 respectively.

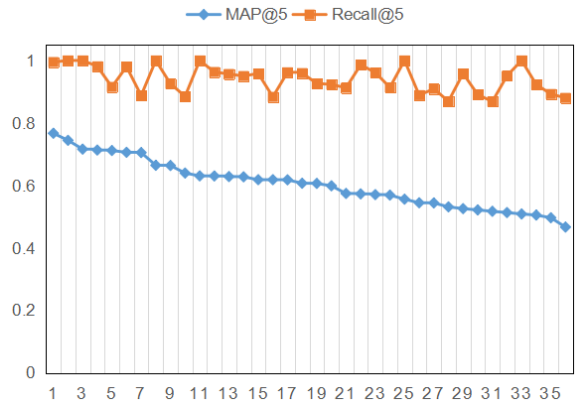

Figure 7: The *MAP@5* and *Recall@5* of all the projects

6.3 Correctness of Function Type (RQ3)

Table 4: Performances of function-type classifiers on transaction function corpus

Classifier	EI			EO			EQ			Acc.
	P	R	F1	P	R	F1	P	R	F1	
NB	79%	63%	70%	29%	**65%**	40%	74%	52%	61%	0.59
LR	81%	90%	86%	59%	41%	48%	84%	84%	84%	0.80
RF	84%	**94%**	**89%**	**78%**	55%	**64%**	**88%**	86%	**87%**	**0.85**
J48	**85%**	91%	88%	67%	54%	60%	87%	**87%**	**87%**	0.83

Table 5: Performances of function-type classifiers on requirement-function corpus

Classifier	EI			EO			EQ			Acc.
	P	R	F1	P	R	F1	P	R	F1	
NB	83%	70%	76%	32%	66%	43%	78%	60%	68%	0.66
LR	87%	94%	90%	74%	48%	58%	87%	89%	88%	0.85
RF	**98%**	**100%**	**99%**	**99%**	**95%**	**97%**	**99%**	**99%**	**99%**	**0.99**
J48	94%	97%	96%	91%	75%	82%	93%	95%	94%	0.93

Table 4 demonstrates the performances of different text classification approaches on classifying history transaction functions corpus (Table 3(b)). We can see that the random forest approach reaches the highest performances in most of the cases. It can achieve 0.85 of overall accuracy (Acc.), and F1 score of 89%, 64%, 87% respectively. Table 5 shows the performances of function-type classifier on requirement-function corpus (Table 3(a)). We can see that the optimal RF model also performs best on the requirement-function corpus. It could achieve quite good performances with the accuracy of 0.99, F1 score of 99%, 97%, and 99% for EI, EO, and EQ respectively. The promising results confirm that our approach could provide an accurate classification of the three transaction-function types.

Answers to RQ3: Our approach can automatically train four ML models, and select the optimal RF model to correctly classify the three types of transaction functions. On average, the RF model achieves overall accuracy of 0.99, F1 score of 99%, 97%, and 99% for EI, EO, and EQ respectively.

7 DISCUSSIONS

7.1 Usability

We develop a tool HyFinder to support organizations to apply the proposed approach in an end-to-end way based on the following functionalities. (1) Automated feature extraction. HyFinder can build the word embedding model and lexical graph automatically, as long as the organization sets up the corresponding text corpus. For each word, HyFinder can extract its features automatically. (2) Automated ML models training and selection. Some learning algorithms make particular assumptions about the structure of the data or the desired results. If the organization can find one that fits their needs, it can give more useful results, more accurate predictions, or faster training times [13]. Therefore, HyFinder embeds four ML models, i.e., NB, LR, J48, and RF, and can automatically conduct experiments to select the optimal one as the word predictor and function-type classifier. (3) Automated recommend and rank functions. HyFinder can automatically train the statistic language model, and recommend ranked results based on readability. In addition, with the accumulation of new training samples over time, HyFinder supports iteratively improvement by re-training the model with the final accepted functions from feedback.

7.2 Applicability

HyFinder has been integrated into the daily business of the FPA team in CMB. For each project, when a new list of requirements is established, our tool will parse each requirement and recommend a list of functions. By using HyFinder, FPA experts can directly select the functions from the recommended list instead of extracting them from the requirements manually, which largely reduces the effort and difficulty of applying FPA. If the FPA experts consider the recommended functions are incomplete or inappropriate, they can browse the corresponding requirements to retrieve the correct functions back. Meanwhile, HyFinder collects the above manual corrections as feedback for future improvements. To the best of our knowledge, this work is the first solution to automatically extract functions from requirements that are written in natural language. It is quite practically useful to improve automation when applying FPA method among software organizations. Moreover, this work provides opportunities to simplify and summarize complicated requirements into functions.

7.3 Benefits

There are two major benefits when using the proposed approach to automatically extract functions. First, the recommended functions can help preserve the extracted functions to be **objective** and **unbiased**. When extracting functions manually, the results might vary from individual to individual according to their understanding level towards the requirements and the FPA methods, e.g. a 30-percent variance was caused within one organization [30]. The subjectivity and personal bias can be alleviated if providing them the automatically recommended functions. Second, the automated function extraction approach can **reduce the amount of manual work**. Extracting functions from textual requirements is an expensive task, especially when the newly incoming requirements are large in size. In section 6, we can see that most suggested functions

are valid and readable, together with accurate function-type predictions. HyFinder can make them conduct FPA methods faster and reduce their amount of manual work. Especially for large organizations that use FPA methods to schedule project budges or appraise team contributions, this approach would be very useful for largely reducing their expensive manual work on functions extraction.

7.4 Lessons Learned

Importing domain-specific dictionaries. In the proposed approach, we leverage the Stanford NLP parser when analyzing the textual requirements. In the beginning, we use the Stanford parser trained from public news corpus. We note that a large number of sentences are incorrectly analyzed by the Stanford parser. After analyzing the mistakes, we found that most mistakes are related to domain-specific words that cannot be correctly identified by the parser. Therefore, we help CMB build their own domain-specific dictionaries based on new word detection technologies [37]. By integrating the domain-specific dictionaries into the parser, most words can be correctly analyzed. The established domain-specific dictionaries also benefit future NLP analysis tasks in CMB.

Performance consideration and support for big data. Typically, parsing sentences and outputting their semantic analysis results by using public NLP libraries cost high on CPU and memory, which results in low performance of the proposed approach. We encountered a performance issue when the size of requirements over 20,000. The tool will spend more than 24 hours to build models based on the dataset. We resolved the issue by refactoring the feature extraction methods and using multiple threads technology. In the future, we plan to use map-reduce [12] technology to improve the performance of our platform and support analyzing big data.

7.5 Expectations for Automated FPA

After interviewing the staff in CMB, we summarize the following research questions on automated requirement analysis that are desired by FPA practitioners.

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 functions based on product/project information and textual artifacts in the early phrase? Our approach addresses the first step of function counting which is automated recommendation functions based on requirements. Given the data and functions, practitioners need to analyze the complexity of functions and system characteristics to estimate the final size or budges for the projects, which is a complicated 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 functions counting.

How to automatically review functions extracted based on rules of FPA methods? For large organizations that use functions to assess the contributions of IT teams, the functions extracted by the IT team members themselves 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.

7.6 Threats to Validity

External Validity. The external threats relate to the generalizability of the proposed approach. First, we experimented on 36 projects taken from the real banking industry. The results may be different in other organizations. However, the variety of projects and the size of data relatively reduce this threat. Moreover, by training models with their own data, the proposed approach could also be applicable and effective to other organizations.

Internal Validity. The internal threats relate to experimental errors and biases. In this work, we use the functions extracted from requirements by product managers, who have been well-trained to apply FPA, as ground truth to evaluate our approach. The functions are subject to mistakes. However, we adopt ground truth functions that have been double-inspected by FPA experts. The threat of internal validity can be largely alleviated.

Construct Validity. The construct threats relate to suitability of evaluation metrics. The threat lies in the metrics of assessing whether the automated functions are semantically correct. To alleviate that threat, we adopt BLEU, which is one of the most popular measurements to claim a high correlation with human judgments of quality [36], to measure the similarity between two functions. We assume that the functions are valid if the value of BLEU is over 0.25. We randomly select 200 suggested functions and the corresponding actual functions. We employ two FPA experts in CMB, they both confirmed that 81% of them are correctly determined by using BLEU 0.25. As BLEU can determine the similarity appropriately, there are few threats to construct validity.

8 CONCLUSION AND FUTURE WORK

This paper proposed a novel solution to extract functions from textual requirements. Our approach is to train a classifier that can predict whether a given word in a requirement will be included in its transaction functions or not, and then constructing the predicted words into reasonable key phrases as the recommended transaction functions. We leverage a set of natural language processing techniques such as lexical graph, semantic analysis, word embedding, statistic language model, and text classification to provide the solution. We build the word-embedding, language model, and text classifier based on 104,691 historical functions taken from a real industry repository, and conduct case studies on 1,864 requirements from 36 projects. The results showed that the recommendation results of our approach can retrieve most of the actual functions, which can provide considerable savings on manual extraction work by FPA experts. Both our approach and supporting tool were accepted and applied by CMB IT department. Moreover, we discussed the prospective aspects of our results, as well as highlighted lessons learned and desiring researches on automated FPA approaches.

ACKNOWLEDGMENTS

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

REFERENCES

[1] N. A. Z. Adem and Z. M. Kasirun. 2010. 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. 664–669.

[2] Allan J. Albrecht and John E. Gaffney Jr. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans. Software Eng.* 9, 6 (1983), 639–648.

[3] Phillip Bonacich. 2007. Some Unique Properties of Eigenvector Centrality. *Social Networks* 29, 4 (2007), 555–564.

[4] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. 1976. *Graph Theory with Applications*. Vol. 290. Macmillan London.

[5] Ulrik Brandes. 2001. A Faster Algorithm for Betweenness centrality. *Journal of mathematical sociology* 25, 2 (2001), 163–177.

[6] Peter F. Brown, Stephen Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. 1992. An Estimate of an Upper Bound for the Entropy of English. *Comput. Linguistics* 18, 1 (1992), 31–40.

[7] Ciprian Chelba. 2010. *Statistical Language Modeling*. Wiley-Blackwell.

[8] Danqi Chen and Christopher D. Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 740–750.

[9] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. 2014. Computing Classic Closeness Centrality, at Scale. In *Proceedings of the second ACM conference on Online social networks*. 37–50.

[10] Deborah Coughlin. 2003. Correlating Automated and Human Assessments of Machine Translation Quality. In *In Proceedings of MT Summit IX*. 63–70.

[11] Marie-Catherine De Marneffe and Christopher D Manning. 2008. The Stanford Typed Dependencies Representation. In *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*. 1–8.

[12] Jeffrey S Dean and Sanjay Ghemawat. 2008. MapReduce. In *Conference on Symposium on Opearting Systems Design & Implementation*. 10–10.

[13] Janez Demiar and Dale Schuurmans. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7, 1 (2006), 1–30.

[14] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. 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*. 499–510.

[15] T Edagawa, T Akaike, Yoshiki Higo, Shinji Kusumoto, Shigeo Hanabusa, and T Shibamoto. 2011. Function Point Measurement from Web Application Source Code Based on Screen Transitions and Database Accesses. *Journal of Systems and Software* 84, 6 (2011), 976–984.

[16] Tom Fawcett. 2006. An Introduction to ROC Analysis. *Pattern Recognition Letters* 27, 8 (2006), 861–874.

[17] Yoav Goldberg and Omer Levy. 2014. Word2vec Explained: Deriving Mikolov et al.'s Negative-sampling Word-embedding Method. *Eprint Arxiv* (2014).

[18] Yvette Graham and Timothy Baldwin. 2014. Testing for Significance of Increased Correlation with Human Judgment. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 172–176.

[19] International Function Point Users Group. 2010. *Function Point Counting Practices Manual-Release4.3.1*. Technical Report. IFPUG.

[20] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann.

[21] Qiao Huang, Xin Xia, David Lo, and Gail C. Murphy. 2018. Automating Intention Mining. *IEEE Transactions on Software Engineering* PP, 99 (2018), 1–1.

[22] Anie Rose Irawati and Khabib Mustofa. 2012. Measuring Software Functionality Using Function Point Method Based on Design Documentation. *International Journal of Computer Science Issues* 9, 3/1 (2012), 124–130.

[23] Tian Jiang, Lin Tan, and Sunghun Kim. 2013. Personalized Defect Prediction. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*. 279–289.

[24] Timo Johann, Christoph Stanik, Alireza M. Alizadeh B., and Walid Maalej. 2017. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In *Requirements Engineering Conference*.

[25] Karen Sparck Jones. 2004. A Statistical Interpretation of Term Specificity And Its Application In Retrieval. *Journal of Documentation* 28, 1 (2004), 493–502.

[26] J. Kuchta and P. Padhiyar. 2018. Extracting Concepts from the Software Requirements Specification Using Natural Language Processing. In *2018 11th International Conference on Human System Interaction (HSI)*. 443–448.

[27] Shinji Kusumoto, Masahiro Imagawa, Katsuro Inoue, Shuuma Morimoto, Kouji Matsusita, and Michio Tsuda. 2002. Function Point Measurement from Java Programs. In *Proceedings of the 24th International Conference on Software Engineering*. ACM, 576–582.

[28] Evelina Lamma, Paola Mello, and Fabrizio Riguzzi. 2004. A System for Measuring Function Points from an ER-DFD Specification. *Comput. J.* 47, 3 (2004), 358–372.

[29] Luigi Lavazza, Sandro Morasca, and Gabriela Robiolo. 2013. Towards a Simplified Definition of Function Points. *Inf. Softw. Technol.* 55, 10 (2013), 1796–1809.

[30] Graham C. Low and D. Ross Jeffery. 1990. Function Points in the Estimation and Evaluation of the Software Process. *IEEE Trans. Software Eng.* 16, 1 (1990), 64–71.

[31] Christopher D. Manning and Prabhakar Raghavan. 2010. *Introduction to Information Retrieval*.

[32] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Texts. In *Proc. 2004 Conference on Empirical Methods in Natural Language Processing, Barcelona, Spain, July*. 404–411.

[33] L Page. 1998. The PageRank Citation Ranking : Bringing Order to the Web. *Stanford Digital Libraries Working Paper* 9, 1 (1998), 1–14.

[34] Eirini Papagiannopoulou and Grigorios Tsoumakas. 2020. A Review of Keyphrase Extraction. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 10, 2 (2020).

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. 311–318.

[36] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. 311–318.

[37] Fuchun Peng, Fangfang Feng, and Andrew Mccallum. 2004. Chinese Segmentation and New Word Detection using Conditional Random Fields. *Proceedings of Coling* (2004), 562–568.

[38] José Antonio Pow-Sang, Loretta Gasco, and Arturo Nakasone. 2009. 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*. 160–167.

[39] C. J. Van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann, Newton, MA, USA.

[40] S J Russell and P. N. Norvig. 1995. Artificial Intelligence: A Modern Approach. *Applied Mechanics and Materials* 263, 5 (1995).

[41] Muhammet Ali Sag and Ayça Tarhan. 2014. 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, 272–281.

[42] Shubhashis Sengupta. 2013. Automatic Extraction of Glossary Terms from Natural Language Requirements. In *Requirements Engineering Conference*.

[43] Kristina Toutanvoa and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP 2000, Hong Kong, October 7-8, 2000*.

[44] Takuya Uemura, Shinji Kusumoto, and Katsuro Inoue. 2001. Function-Point Analysis Using Design Specifications Based on the Unified Modelling Language. *Journal of software maintenance and evolution: Research and practice* 13, 4 (2001), 223–243.

[45] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. 2016. Phrase-based Extraction of User Opinions in Mobile App Reviews. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. 726–731.

[46] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting Semantically Linkable Knowledge in Developer Online Forums via Convolutional Neural Network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*. 51–62.

[47] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. 2016. Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 127–137.