Named Entity Recognition on Software Requirements Specification Documents

Garima Malik ^{†,*}, Mucahit Cevik[†], Yusef Khedr[†], Devang Parikh[‡], Ayşe Başar[†]

[†] Data Science Lab, Ryerson University, Toronto, Canada

[‡] IBM, Toronto, Canada

Abstract

Software Requirements Specifications (SRS) documents set the requirements and expectations of software development projects. This textual information is considered as a guideline for identifying various requirements for the developers. The content of an SRS document is concentrated around software requirement specific entities which can be used to improve software-based information retrieval systems. In this paper, we consider Named Entity Recognition (NER) over SRS documents and classify the software requirement specific entities using ML-based methods. Existing NER methods are typically restricted to four basic sets of entities that are not pertinent to NER in the software requirement domain. We define ten software requirement-based entities which cover the essence of the SRS documents. In order to predict the appropriate entity tags, we first create a limited feature set and experiment with various NER models including ML-based probabilistic models and DL models. We conduct a detailed numerical study to evaluate the effectiveness of the NER models over the SRS datasets. NER models with a basic feature set demonstrate promising performance with an F1-macro score of 81%. Our analysis shows that the entities can be extracted with high accuracy from the SRS documents, which then can be used for various purposes in practice such as extractive summarization of large SRS documents and effectively grouping the requirements.

Keywords: Software Requirements Specifications, NER, Conditional Random Fields.

1. Introduction

The development of any software product comprises of different phases which require formal documentation such as product requirements document or Software Requirements Specifications (SRS) document, user experience design document, maintenance and help guide, and API documentation. The information present in these documents has a defined semantic structure which is coherent for humans, however machines typically are not able to comprehend the same. To organize, curate and retrieve the relevant information from the SRS documents, there is usually a need for an automated information extraction system.

SRS documents build the foundation of product development by stating clear requirements of the system. It also provides an in-depth description of functional and non-functional requirements related to a particular software. We can consider SRS documents as a collection of unstructured data, i.e., natural language text, and extract the information which can reflect the essence of the document. This process is called information extraction or entity recognition. Extracting the main entities in an SRS document helps in sorting the data and identifying necessary information, which can be important for the development process, quality assurance, operations, and maintenance in software industry. Named Entity Recognition (NER) is the first step of the entity-centric search systems over text documents. Performing NER task over SRS documents can act as a catalyst for text classification (classifying functional and non-functional requirements), categorization of software documents, and topic modelling on SRS documents.

Several studies highlighted the importance of NER in the field of information extraction [1, 2]. In earlier studies, the NER task was primarily dedicated to extract a limited set of entities from natural text such as name, organization, location, and miscellaneous [3]. Later,

* garima.malik@ryerson.ca

researchers studied the effect of fine grained entity recognition i.e. adding a deeper level of hierarchy in entity tags to improve the pre-processing of NLP based applications [4, 5].

In general, building domain specific NER models are comparatively difficult than general NER models as it requires the domain specific corpora and external resources or dictionary to extract features [2, 5, 6]. Previous studies on NER mostly use ML-based discriminative models such as Conditional Random Fields (CRF), Conditional Maximum Entropy Model (C-MEM) and Hidden Markov Models [2, 6, 7]. These models perform well in the presence of handcrafted features and gazetteers, however, in the case of domain specific NER tasks, the overall performance typically drops due to models inability to understand the semantics of a given domain. In this paper, we investigate the performance of CRF and C-MEM with a minimal or basic feature set that is present in the SRS documents.

Recently, Deep Learning (DL) based methods provided promising results for NER tasks [8–10]. DL offers automatic feature extraction capabilities and can represent natural language text as raw vectors without compromising the semantic structure of the words. DL-based NER systems require limited feature engineering and provide state-of-the-art performance when a large training corpora is available. Every DL model can be visualized in the form of taxonomy of layers which starts with the word embeddings layered with context encoders (e.g., BiLSTM) and an output layer (e.g., Softmax or CRF) as a tag decoder [2]. We also consider a BiLSTM-CRF architecture with and without pre-trained embeddings to analyze the performance of the DL architectures over a small-sized SRS dataset.

Few other studies consider NER tasks in the software engineering domain [6, 11–13]. Both machine learning (ML) and DL models have been employed in these studies. Specifically, Ye, Xing, Foo, Ang, Li, and Kapre [6] explored the social context of software development using stack overflow posts and proposed a system called S-NER which can recognize a wider category of software entities in software related social media forums. Reddy, Prasad, Chikkamath, and Mandadi [13] implemented a software domain specific NER model on stack overflow posts with ML-CRF and BiLSTM-CRF. The authors identified 22 different software specific entities which includes object oriented languages, CPU instruction sets, and basic terminologies related to software engineering. Tabassum, Maddela, Xu, and Ritter [12] proposed an entity tagging system for the programming language code present in the stack overflow posts. In this study, we introduce a fine grained entity recognition task for SRS, which contributes to the software development process by enabling efficient extraction of the important information from the SRS documents.

Research Contributions. We investigate NER approaches that consider the contextual nature of natural language text to identify software requirement-specific entities from SRS documents. Existing research on NER in software engineering mostly focuses on social media context, e.g., Q/A forums, Github, and Stack Overflow posts. Few studies explore SRS domain for information extraction. As such, our study makes the following contributions:

- We formulate a NER task over SRS documents dataset to identify the software specific entities. Through this formulation, we also identify the design challenges and key decisions to be made in NER for software requirement domain.
- We collect and annotate a corpus of SRS documents, obtained through industry collaboration. We use this corpus to train and test our models while showcasing the effectiveness of the software requirement-based NER models.
- We perform a detailed numerical study with the existing NER models, namely ML-CRF, C-MEM, and BiLSTM-CRF, which provide empirical evidence on the effectiveness of such methods for software engineering domain-specific datasets.

2. Methodology

Understanding the challenges associated with domain specific NER systems helps creating a customized NER methodology for the software engineering domain. In general, domain

specific NER systems require standard corpus or dictionary lookup attributed to a particular domain and manual labelling of training data. Designing NER systems within a field or domain is difficult as existing NER corpora are labelled to identify real-world objects (e.g., person, location, organization, time and date). These entities as well as other open-source corpora entities are not meaningful in the software engineering domain, let alone the software requirements subdomain. Accordingly, we summarize the challenges faced when developing the NER methodology using SRS documents as follows.

- (1) SRS documents are expected to be formal with strict linguistic rules, however, the data might have occurrences of spelling and capitalization errors.
- (2) In SRS documents, switching between different softwares might prove difficult as the majority of the text includes subjective words that may differ from one software to another. This creates difficulty for determining appropriate tags as some of these are software specific internal processes which are not widely known.
- (3) Different software specific entities might have multiple meanings, for example “query” could be either a ‘Verb’ tag or a ‘Core’ tag depending on the context.
- (4) Manual tagging can be extremely tedious. We might expect discrepancies and inconsistencies with human annotators as they can overlook some of the details while tagging multiple requirements using ten separate entities.
- (5) Obtaining annotated data is typically difficult. Although we were able to obtain some SRS data, we were restricted to a few thousand lines of text, which cause entities to be relatively infrequent. This also makes it difficult to decide on the entity set which can cover all the important entities related to the SRS documents.

2.1. Data preparation

NER requires annotated data in order to perform supervised learning. We collected 5,100 lines of data from the SRS documents of Dynamic Object-Oriented Requirements System (DOORS) software [14]. It is a requirement management tool created by IBM to manage various projects and the business goals. For our annotations, we managed to extract the requirements stated in the SRS documents of the DOORS software. Out of 5,100 lines of text, we extracted 3,333 sentences after initial data cleaning and pre-processing steps (e.g., basic tokenization and stop words removal). We did not apply any lemmatization or stemming techniques as we opted to keep the semantic structure of words similar to the documents prepared in the software development processes. We determined a diverse set of entities covering different aspects of the software specifications (see Table 1). We defined ten entity categories which can help end users to extract information from the SRS documents.

Table 1. Software requirement specific entities used for the annotation of the SRS dataset.

Entity	Description	Examples
O	Non-entity tokens	Tokens not covered by the entity set
CORE	Token specific to software requirement domain	Artifacts, Module, Attribute
USER	Token denoting a specific user of the software	Admin, Developer, Rich client user
GUI	Components of the graphical user interface	Widget, Viewer, Wizard, Page
HARDWARE	Components of computer hardware	Keyboard, Mouse, CPU
LANGUAGE	Basic programming language terminology	Java, Binary, Inline, Parser
API	Software APIs	REST, JTS, VVC
STANDARD	Software/programming standards	JSON, PDF, Version, Format
PLATFORM	Applications and software platforms	Linux, Browsers
ADJECTIVE	Miscellaneous descriptive adjectives of software, hardware and design components	Existing, Available, Duplicate
VERB	Miscellaneous actions (verbs) related to software, hardware and design components	Upgrade, View, Store, Restructure

After defining the entity set, we perform the manual annotation of the data using a collaborative approach between multiple annotators with backgrounds in software development. Specifically, we leverage the capabilities of the IBM Watson knowledge studio tool

alongside the human annotators for the annotation task. The human annotators annotate a subset of the data. In order to isolate inconsistencies in the annotations, we create dictionaries for each software requirement specific entity. In the dictionaries, the keys consist of words or tokens that belong to a particular entity type. We apply a linear approach to annotations by running only the dictionaries on the dataset in the IBM Watson knowledge studio ensuring the highest consistency across the dataset. Final annotations are checked by multiple human annotators to ensure high-quality data. For creating the annotations, we apply the widely accepted BIO representation scheme. In software requirement context, BIO can be interpreted as Beginning, Inside or Outside of the entity tag. Figure 1 shows the sample annotated sentence in the BIO format from the SRS dataset.

As	a	user	I	should	be	able	to	filter	by	artifact	type
O	O	B-USER	O	O	O	B-Verb	O	B-Verb	O	B-Standard	I-Standard

Figure 1. Sample sentence from the SRS data set tagged in BIO format.

After preparing the uniform data annotations, we performed exploratory data analysis on the pre-processed SRS data set. Maximum token count in an instance is 60, which is helpful in padding all the sentences to a maximum length of 60 for the DL models. Examining the distribution of different software requirement specific entities, we observe that ‘Verb’, ‘Core’ and ‘User’ are the most frequent entities, whereas ‘API’ and ‘Hardware’ are the least frequent entities in the dataset. Additionally, entity distribution in the BIO format shows that ‘B-Verb’ is the most frequent annotated entity and ‘I-Hardware’ is the least frequent.

2.2. Named Entity Recognition models

We consider three different NER approaches, namely, ML-CRF, C-MEM, and BiLSTM-CRF. C-MEM relies on the conditional version of the maximum entropy classifier that is typically used for sequence labelling. Similarly, ML-CRF is a probabilistic approach for NER, where CRF models the conditional probability of assigning a tag sequence given the observation or token sequence. BiLSTM-CRF is a DL-based approach, which consists of word embedding layers followed by Bidirectional long short-term memory (BiLSTM) layers, and the CRF layer at the end. In our analysis with BiLSTM-CRF, we also consider GloVe embeddings, which are pretrained on the extracted data from Twitter and Wikipedia.

2.3. Experimental setup

Table 2 shows the values of hyper parameters used in training the NER models. We implemented the ML-CRF learning model using CRFsuite library [15] and performed extensive hyperparameter tuning. In Table 2, ‘C1’ and ‘C2’ are coefficients for L1 and L2 regularization, respectively. For C-MEM model, we found that ‘GIS’ (Generalised Iterative Scaling) with max_iteration as 100 provided the best performance. We also performed detailed hyperparameter tuning for DL models (BiLSTM-CRF), and identified the parameters in Table 2. For model training and testing, we used 10-fold cross validation.

Table 2. Hyper parameter values for the NER models.

Model	Settings
ML-CRF	algorithm= ‘lbfgs’, c1 = 0.1, c2 = 0.1, max_iterations = 100
MEM	algorithm= ‘GIS’, max_iterations = 100
BiLSTM-CRF	Units = 128, epochs = 30, batch_size = 16, Kernel Embedding = 60
BiLSTM-CRF(GloVe)	Units = 128, recurrent dropout = 0.1, batch_size = 32, epochs = 30, GloVe dimensions = 300

3. Results

Our experiments are designed to show the applications of NER in software requirement specification domain. We use standard NER performance metrics such as precision, recall, and F1-score (weighted and macro) to evaluate the performances of the employed models.

3.1. Performance of the NER methods

We first compare the performances of the NER models (see Table 3). In general, all the NER models manage to learn the patterns in the SRS dataset, however, ML-CRF outperforms the other models. We note that Reddy, Prasad, Chikkamath, and Mandadi [13] obtained F1-score of 85% with ML-CRF model and 95% with BiLSTM-CRF model on stack overflow data set consisting of 150,000 data instances. In comparison to existing methods, which require an abundance of handcrafted features and external knowledge to train the classifier [6, 13], we only used a basic feature set and managed to obtain an F1-macro of 81% and F1-weighted of 89%.

Table 3. Performance of different NER models (mean $\pm$ std).

Model	Precision	Recall	F1-weighted	F1-macro
ML-CRF	0.89 $\pm$ 0.01	0.90 $\pm$ 0.01	0.89 $\pm$ 0.01	0.81 $\pm$ 0.04
C-MEM	0.76 $\pm$ 0.01	0.86 $\pm$ 0.01	0.81 $\pm$ 0.01	0.73 $\pm$ 0.03
BiLSTM-CRF	0.86 $\pm$ 0.01	0.89 $\pm$ 0.01	0.88 $\pm$ 0.01	0.75 $\pm$ 0.03
BiLSTM-CRF(GloVe)	0.83 $\pm$ 0.05	0.88 $\pm$ 0.05	0.86 $\pm$ 0.05	0.75 $\pm$ 0.10

The DL models, BiLSTM-CRF with and without GloVe embeddings, show inferior performance to ML-CRF, with a mean F1-macro of 75%. This can be explained by the fact that DL models require a large set of training data to discover hidden patterns. The existing SRS dataset is insufficient to improve the F1-macro. Furthermore, BiLSTM-CRF without any pre-trained embeddings performs slightly better than GloVe architecture because of domain specificity. GloVe embeddings are trained on Twitter dataset which proves to be out of domain for the given NER task in software requirement domain.

We next assessed the performance of the best model (ML-CRF) in predicting the individual entity categories. We note that the model performs best for ‘Core’, ‘User’, and ‘Verb’ entities (F1-score ranging between 87%-97%), which are the most prevalent entities in the dataset. However, mostly due to lack of data, the model performs poorly for identifying the ‘Hardware’ (with F1-score of 73%) and ‘Language’ (with F1-score of 50%) entities.

3.2. Illustrative examples

Table 4 shows SRS dataset samples which are annotated by ML-CRF and compared against the standard annotations. For Sample-1, ML-CRF produced correct predictions except the word ‘viewer’ is classified as ‘User’ tag instead of ‘GUI’. This can be explained by the training feature set, that is, we trained the ML-CRF model with the characters extracted from the tokens and all the words attributed to ‘User’ entity that ends with character ‘er’ which causes wrong prediction. Similar observations can be made for Sample-2 as well.

4. Conclusion and Future Work

In this paper, we performed fine-grained NER tasks in the software requirement domain. We used four models namely ML-CRF, C-MEM, BiLSTM-CRF, and BiLSTM-CRF (GloVe). These models can be utilized to improve the performance of document querying systems (semantic similarity searches), document summarizing models and any other system that uses unstructured texts from this domain. When trained with the annotated SRS corpus,

Table 4. Samples from the SRS dataset annotated by the best performing model (ML-CRF).

sample	type	annotations
Sample-1	Predicted	As a Rich Client user USER I can view VERB an artifact CORE using VERB a viewer USER suitable for that artifact type STANDARD
	Standard	As a Rich Client user USER I can view VERB an artifact CORE using VERB a viewer GUI suitable for that artifact type STANDARD
Sample-2	Predicted	Participant ADJECTIVE would like to be able VERB to change VERB the name CORE of the columns GUI in a view VERB in the rich client USER
	Standard	Participant CORE would like to be able VERB to change VERB the name CORE of the columns GUI in a view VERB in the rich client USER

ML-CRF slightly outperforms the DL models. Dataset size and the distribution of software requirement specific entities impact the performance of the NER models. ML-CRF is able to learn the patterns for ‘User’ and ‘Core’ entities, however, unable to determine the distinct patterns for ‘Hardware’ and ‘Language’ entities. In future research, a large collection of annotated data can be developed to train the dense DL architectures. Furthermore, domain specific word embeddings such as BERT and ELMo (Embeddings from language modelling) can be built to train fine grained NER models. Lastly, a larger software engineering domain-specific entity set can be used to develop more informative NLP-based applications.

References

- [1] E. F. Sang and F. De Meulder. “Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition”. In: *arXiv preprint cs/0306050* (2003).
- [2] J. Li, A. Sun, J. Han, and C. Li. “A survey on deep learning for named entity recognition”. In: *IEEE TKDE* (2020).
- [3] D. Nadeau and S. Sekine. “A survey of named entity recognition and classification”. In: *Linguisticae Investigationes* 30.1 (2007), pp. 3–26.
- [4] C. M. Costa, G. Veiga, A. Sousa, and S. Nunes. “Evaluation of Stanford NER for extraction of assembly information from instruction manuals”. English. In: IEEE, 2017, pp. 302–309.
- [5] M. Habibi, L. Weber, M. Neves, D. L. Wiegandt, and U. Leser. “Deep learning with word embeddings improves biomedical named entity recognition”. In: *Bioinformatics* 33.14 (2017), pp. i37–i48.
- [6] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre. “Software-specific named entity recognition in software engineering social content”. In: *SANER 2016*. Vol. 1. IEEE. 2016, pp. 90–101.
- [7] T. Liu, J.-G. Yao, and C.-Y. Lin. “Towards improving neural named entity recognition with gazetteers”. In: *Proceedings of the 57th Annual Meeting of the ACL*. 2019, pp. 5301–5307.
- [8] Z. Huang, W. Xu, and K. Yu. “Bidirectional LSTM-CRF models for sequence tagging”. In: *arXiv preprint arXiv:1508.01991* (2015).
- [9] J. P. Chiu and E. Nichols. “Named entity recognition with bidirectional LSTM-CNNs”. In: *Transactions of the ACL* 4 (2016), pp. 357–370.
- [10] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. “Neural architectures for named entity recognition”. In: *arXiv preprint arXiv:1603.01360* (2016).
- [11] C. Zhou, B. Li, and X. Sun. “Improving software bug-specific named entity recognition with deep neural network”. In: *Journal of Systems and Software* (2020), p. 110572.
- [12] J. Tabassum, M. Maddela, W. Xu, and A. Ritter. “Code and named entity recognition in stackoverflow”. In: *arXiv preprint arXiv:2005.01634* (2020).
- [13] M. V. P. Reddy, P. Prasad, M. Chikkamath, and S. Mandadi. “NERSE: Named Entity Recognition in Software Engineering as a Service”. In: *Service Research and Innovation*. Springer, 2018, pp. 65–80.
- [14] E. Hull, K. Jackson, and J. Dick. “DOORS: a tool to manage requirements”. In: *Requirements engineering*. Springer, 2002, pp. 187–204.
- [15] M. Korobov. *sklearn_crfsuite.CRF*. <https://sklearn-crfsuite.readthedocs.io/>. 2015.