JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 295-316 (2022)
DOI: 10.6688/JISE.202203_38(2).0002

Investigating Software Domain Impact in Requirements Quality Attributes Prediction*

GREGORIUS AIRLANGGA[1,2] AND ALAN LIU[2]
[1]*Department of Information System*
Atma Jaya Catholic University of Indonesia
Jakarta, 12930 Indonesia
[2]*Department of Electrical Engineering*
National Chung Cheng University
Chiayi, 621301 Taiwan
E-mail: gregorius.airlangga@atmajaya.ac.id; aliu@ee.ccu.edu.tw

Several researchers have attempted to confront the problems in quality attributes prediction using AI approaches consisting of knowledge-driven and data-driven techniques. However, due to the lack of a shared training dataset and standardized definition of quality attributes, inaccurate feature extraction may lead to the inconsistency and poor performance of a prediction model. Different from prior works, we have investigated the impact of software domain in quality attributes prediction using deep learning methods with different datasets. From the results, we conclude with two recommendations: (i) the existing secondary dataset such as PROMISE and Concordia are not enough to be used as a ground truth; (ii) a deep learning approach should be supported from the aspects of broader domains in order to capture a variety of natural language requirements. The contribution of this paper is to raise the awareness of identifying the quality attributes in requirements writing and help requirements providers understand what issues to focus. A prototype of requirements annotator is introduced to show how the requirements are processed.

Keywords: software requirements, deep learning, quality attributes, software domain, natural language processing

1. INTRODUCTION

Software requirements are collections of description, knowledge and workflow between entities to describe how software should work and behave. The process to acquire and formulate software requirements is defined as requirements engineering. In a requirements engineering practice, functional requirements (FR) are easy to define, but the quality of the completed software system is affected by various factors such as lack of knowledge in quality attributes, requirements priority, technical resources, expertise, documentations, and development time. Those conditions often occur and lead to issues related to software quality attributes including privacy and security, performance,

Received September 30, 2020; revised December 9, 2020 & February 21, 2021; accepted March 18, 2021.
Communicated by Chih-Yi Chiu.
* This project is partially supported by Ministry of Science and Technology, Taiwan under Grant No. MOST 109-2221-E-194-022-MY3.

usability, *etc*. [1]. One of the solutions proposed is the implementation of AI techniques to predict quality attributes in the requirements stages.

The pioneering work in quality attributes prediction was first explored in Cleland-Huang's work [2] by producing the PROMISE dataset and predicting quality attributes from natural language. Based on the study, several investigations have been conducted until recently and most of them only focus on the increasing accuracy or F1-Score as prediction performance metric. There are two main categories of techniques: data-driven [3–7] and knowledge-driven [8–12]. The data-driven approach is the use of supervised learning techniques to predict quality attributes based on the statistical patterns of requirements text. The data collection is the crucial process in order to generate a robust prediction model. In contrast, a knowledge-driven approach is the use of combination between prior knowledge and inference rules to do prediction with a very limited or even no dataset available in the training process.

Even with these promising results, there are some limitations in the recent study. First, the current works only focus on the increasing accuracy or F1-Score metric, which means that the prediction techniques are dominantly proposed with limited or without major focus on the dataset semantics. Second, the performance analysis was limited by the availability of the open dataset and community tools. Third, the standard definition of quality attributes is lacking, negatively impacting the validity of data annotation.

In this study, the main discussion is about analyzing dataset semantics using data-driven perspective on various software domain and its impact in predicting quality attributes. In addition, we have also developed a community based tool for collecting and annotating software requirements texts used for research. We have named this tool *softrequest.co* or the Annotator.

Moreover, in order to build comprehensive analysis, we suggest two main research questions. First, how would the model's performance predict quality attributes from software requirements in various domains when the trained model is generated from a specific domain? Second, what is the impact of data domain augmentation to classify the PROMISE dataset? To answer these research questions, we have analyzed our experiments with a deep learning model in the Convolutional Neural Network (CNN) to predict quality attributes from software requirements texts in different software domains.

The result of our investigation gives a further understanding and recommendation for other researchers to be aware of the effect of data domain. In this paper, we start the introduction in Section 1. Section 2 defines the domain dependent problem. The related work will be presented in Section 3; in Section 4, we present the proposed method. Section 5 analyzes the experimental results, and the last section provides the conclusion and future work.

2. DOMAIN DEPENDENT PROBLEM

In a text classification problem, we are given description $b \in B$ of a document, where B is the document space, along with a fixed set of classes $C = \{c_1, c_2, ..., c_j\}$. Classes are also called categories or labels. Using a supervised learning method, we must approximate a classifier $\underline{\gamma}$ that maps documents to classes. Using this pattern,

several problems such as sentiment analysis, spam detector, language detection, *etc*. can be resolved.

In sentiment analysis, the document space B contains the written expression of emotion about an event or an object, and C is a collection of sentiment label such as positive, neutral and negative. In the security problem such as spam detector, B contains a collection of spam and non-spam text which are further labeled as spam or not spam. In language detection, B is a collection of languages and C is the language category. In this section, we explain about the quality attributes prediction problem as one of the text classification problem in software engineering.

Quality attributes prediction can be categorized as a text classification problem, in which we are given requirements text ($r \in X$) of a document, where X is a high-dimensional space representing a semantic feature of r. A fixed set of quality attributes can be described as $Q = \{q_1, q_2, ..., q_j\}$ where q_j can be encoded as one-hot encoding form, in $\mathbb{R}^+$, $q_j \rightarrow e_j$, meaning that $\mathbb{R}^+$ denotes the set of positive real numbers, that is $\mathbb{R}^+ = \{x \in R | x > 0\}$, and e_j is a vector of standard base and denotes the vector with 1 in the jth coordinate and 0 elsewhere.

The quality attributes are the labels defined by experts. Given a domain D dataset of labeled requirements text $\langle r, q \rangle$, where $\langle r, q \rangle \in X \times Q$, we can consider the other domain as a different dataset E where E and D have semantic relation $sim(D, E)$ in a human cognitive point of view. Therefore, $r \in E$ and $r \in D$ have shared domain context. However, the high dimensional space X representing semantic feature of r in E and D may not have similarity characteristics to each other.

Using a certain learning method, a classifier or classification function γ that maps high dimensional space X to quality attributes $\gamma : X \rightarrow Q$ will be used. This type of learning is called supervised learning because a supervisor (*e.g*. the software analyst who defines the quality attributes in training datasets) serves as a teacher directing the learning process. The supervised learning method is denoted by τ and can be written as $\tau(D) = \gamma$.

The learning method τ takes the training set D or E as input and return the prediction model or learned classification function $\underline{\gamma}$. The domain-dependent problem occurs when the learned classification function $\underline{\gamma}$ from D cannot give an approaching value to predict Q in E and vice versa.

3. RELATED WORK

Predicting quality attributes from software requirements could be classified as processing a cross-domain problem such as mentioned in Section 2. Most of the related work studied the sentiment analysis problem from the view point of text sourced data, and no quality attributes prediction has been conducted on cross-domain software requirements.

This work is inspired by a comprehensive systematic literature review [13] containing the methods and techniques employed in a cross-domain sentiment analysis. A prediction model that performs well in one domain might under perform in another. We consider this work as a baseline to our research for conducting the initial intuition of the challenge in predicting quality attributes from software requirements texts since both cases have similar characteristics referred in Section 2.

Different from the problems in sentiment analysis where mostly using three ground

truths: positive, neutral and negative, our problem is more complicated since the quality attributes from natural language express various technical analysis and software behavior expectations. In addition, the quality attributes prediction does not only contain two or three labels but it can be abundant since the software behavior itself is continuously evolving and related to specific technical domains such as hardware, network, interface, organizations, regulations, and user-perceived things.

Predicting quality attributes from natural language has been investigated by several researchers using data-driven or knowledge-driven approaches. Like [2] as an initial work, they built the PROMISE repository containing the software requirements from 15 different software domains. All requirements texts were further labeled into 12 quality attributes.

The dataset and the initial work were utilized as the benchmark by various researchers such as [3] and [7]. They proposed several preprocessing methods to extract useful features from software requirements texts; subsequently, the processed texts were fed into traditional classifiers such as Support Vector Machine (SVM), Naive Bayes and Decision Tree to decide the quality attributes. Besides traditional classifiers, several recent works have used deep learning models [6, 14, 15]. The benefit of using deep learning is that they reduce the preprocessing effort in terms of extracting word features from software requirements texts. Even though all prior works have performed well to classify the quality attributes from PROMISE dataset, further investigation to measure the quality attributes prediction performance in terms of domain dependent problem has not been conducted.

In this work we investigate the domain dependent problem without focusing into optimizing the classifier model. In addition, our goal is to know the effect of the cross-domain issues for the quality attributes performance and enrich software requirements dataset related to quality attributes. Besides using the PROMISE dataset, we have produced a dataset using our prototype system (*softrequest.co*) as an annotator to assign the software domain and quality attributes by experienced software developers. Furthermore, we have built a deep learning model to predict 12 quality attributes from different software domain and then analyzed the results to answer the research questions.

4. PROPOSED APPROACH

4.1 Overview

We propose a framework to analyze software requirements for quality attributes prediction as a domain-dependent problem. The proposed approach is presented in Fig. 1, in which three main processes are proposed to answer research questions. Requirements elicitation depends on the Requirements Gathering process, which is explained in Section 4.2, and the Annotator system (*softrequest.co*) utilizing the results from the modeling and prediction parts of the framework has been built and described in Section 4.3. Modeling the prediction model generated from three datasets is explained in Section 4.4. Finally, a machine learning pipeline is explained in Section 4.5. The pipeline contains three sub-phases: categorizing, predicting, and assessing. The assessment contains domain-dependent and data augmentation evaluation to answer research questions.

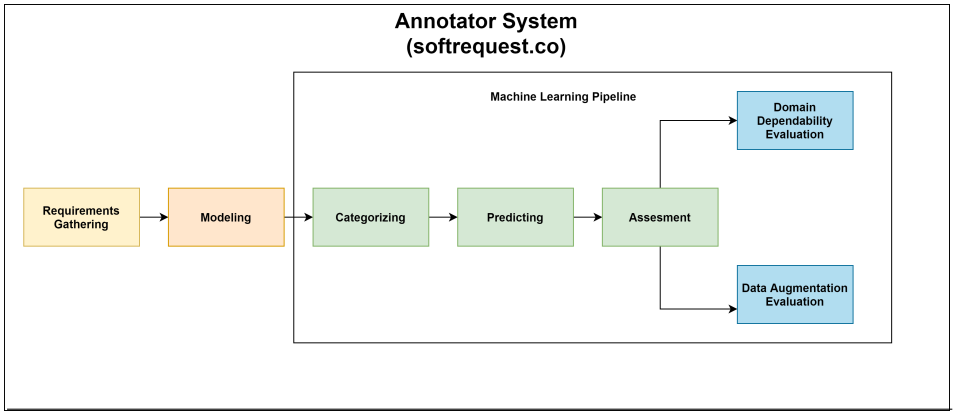


Fig. 1. Proposed framework.

4.2 Requirements Gathering

Requirements gathering is the crucial step in our experiment. There are two sub-phase to collect software requirements data: (i) Data Collection Process and (ii) Dataset Domain Labeling; The data collection process describes the process to define a requirements text using *softrequest.co* application and to label it with appropriate quality attributes. After the collection process has been conducted, the domain labeling process will be executed as a process to categorize, vote and argue the requirements text into specific domain.

4.2.1 Data collection process

The source requirements to analyze come from two datasets. The first is from the PROMISE repository [3] which comprised twelve software quality attributes categories, including eleven sets of nonfunctional requirements and one set of functional requirement (NFR and FR). Data was generated by Cleland-Huang from fifteen master students projects as mentioned previously. The dataset contains 625 rows, in which each row is a piece of independent requirements text.

The second dataset was gathered through our Annotator system (*http://softrequest.co/*); the data was gathered from 25^{th} May until 5^{th} June 2020 from 10 developers who were working on five software projects in Indonesia, including education information system, boutique e-commerce, smart home for elderly, smart farming system and police department information system. Each piece of requirements is then manually labeled by them into several quality attributes including 12 categories: functional, scalability, performance, usability, reliability, availability, security, operational, portability, language support, policy, and interoperability. The collected dataset consists of 3473 requirements texts. Some examples of those dataset are presented in Tables 1 and 2.

In order to guarantee the labeling quality, the first author of this paper gave the briefing about quality attributes in two hours of online meeting. The author gave quiz to make sure that the developers understood about the annotation process, and

then the author and developers did a cross-checking process in order to re-annotate the requirements text into 12 requirements categories. To ensure the validity of the an- notations, the interpreter agreement is computed by means of the Cohen's Kappa, resulting in $k = 0.91$, indicating an almost perfect agreement. The score definition is based on the following qualitative measures associated to the different ranges of the Cohen's Kappa: $k < 0$, no agreement; $0 \leq k \leq 0.20$, slight; $0.21 \leq k \leq 0.40$, fair; $0.41 \leq k \leq 0.60$, moderate; $0.61 \leq k \leq 0.80$ substantial; and $0.81 \leq k \leq 1$ almost perfect agreement.

4.2.2 Dataset domain labeling

Both data from [2] and *softrequest* have same number of categories and quality attributes labels, and the diversity is about the data quantity and the natural language style pattern. In the [3] dataset, the natural language pattern follows the IEEE standard 830-1998 while *softrequest* data is provided by the developers in any style. The requirements texts were further doubled-checked by the system analysts in order to label and validate the quality attributes.

In order to label the requirements domain, both developers and software analyst executed labeling and domain verification process to a developers community using the Annotator tool. First, we posted all labeled requirements text in the system, in which the labels contained the domain label and quality attributes label. We then invited experienced developers in various domains such as e-commerce and enterprise system, health information system, IoT system, mobile based applications, media and information system, education information system and security application as domain experts. The invitation was sent to 300 developers, but the total of registered developers ended up only 189.

We asked the developers to verify the quality attributes labels and domain labels for the posted requirements. If they agreed, they had to make a positive vote; otherwise comments regarding the reasons for a negative vote were collected. Interestingly, in this process, the developers were very enthusiastic to verify and comment, and most developers agreed with initial labels defined by the requirements providers.

Some constructive discussion sessions took place in processing ambiguous labels such as the smart home system requirements for the elderly presented in the item 5 of Table 2, "Smart Home should have an automatic medicine injection machine." Fifty-nine developers argued that the requirements text belonged to a health information system domain since they assumed health information including medical history, medicine, and hospital was to be processed while the technology concerning safety and hospital standard was to be observed.

However, the arguments were refuted by other developers, since the concern of the application itself was an automatic system, so the automatic medical injection system itself was only viewed as one of IoT devices. Moreover, the data associated with automatic injection was just used by the system to decide if the injection was needed or not. If the time was close, then the IoT device would send the notification to the elderly through a smart watch, and when the elderly got ready for a proper body position, then the machine would inject the medicine automatically. After majority voting, the requirements text was labeled as a smart home system for elderly.

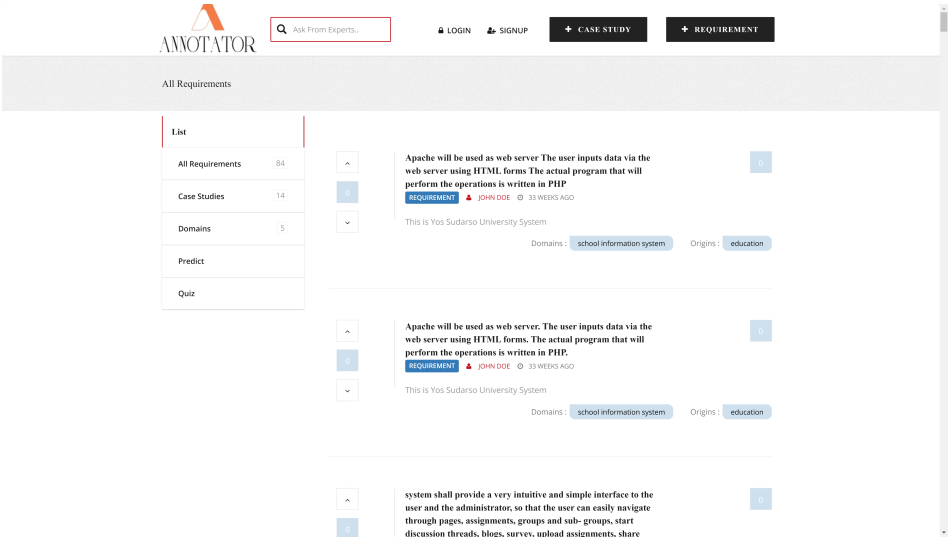


Fig. 2. Annotator system *https://softrequest.co*.

Table 1. Jane Cleland Huang *et al*. [2] dataset example.

No.	Requirements	Label	Domain
1	The system shall be intuitive to the Program Administrators/ Nursing Staff Members.	Usability	Health Information System
2	The vehicle data shall include vehicle year make and model	Functional	Vehicle System
3	The product shall conform to the Americans with Disabilities Act.	Usability	Disabilities Feature in Conference System
4	Website must be fully operational with MSN TV2	Operational	Media Based Application
5	The website should appeal to all Africans not only Nigerians	Look & Feel	News Website

4.3 Annotator System (*Softrequest.co*)

Softrequest.co or the Annotator system is a web-based application system to gather, annotate, communicate and predict requirements text for software engineering

Table 2. Softrequest.co dataset example.

No.	Requirements	Label	Domain
1	Guest should be able to browse Product (Catalogues) and Other Users (Boutiques) but NOT be able to add product directly. That will require herself to have her own boutique (Cavern) first.	Functional	Boutique Web E-Commerce
2	The air temperature in Smart Home also needs to be guarded, and then before we take a bath, we can first raise the temperature high enough. This is especially important for old people and patients	Security	Smart Home System
3	The Navigation module of the Police Information System (PIS) provides role based landing pages which help in navigating through the (PIS) application. It shows information such as cases assigned, alerts, pending tasks etc hence helping police personnel to plan better and execute with greater efficiency.	Functional	Police Information System
4	The software will interoperate with other software applications which are being developed under National e-Governance Program, in particular Central Agriculture Portal and Android/IOS Applications in Playstore	Interoperability	Smart Farming
5	Smart Home should have an automatic medicine injection machine. Some of the medicines need to be injected to occupant's body. And if elder live alone, there are no people who can help the user to do it. Hence, the system needs the feature.	Usability	Smart Home System For Elderly

stakeholders. To use the Annotator, a stakeholder first needs to register and sign in to share software requirements with the labels through the *all_requirements_menu* form, for input like requirements texts, domain, and quality attributes. The stakeholders also could add multiple requirements in a CSV form to be used by the system.

After the requirements are posted, other stakeholders could give comments and agreement scores and also propose the corrected quality attributes in conjunction with the reason. If the other stakeholders give higher voting scores than the original quality attributes score, then the corrected quality attributes will be selected. The user interface and the use case of the *all_requirements_menu* form could be seen in Figs. 2 and 3.

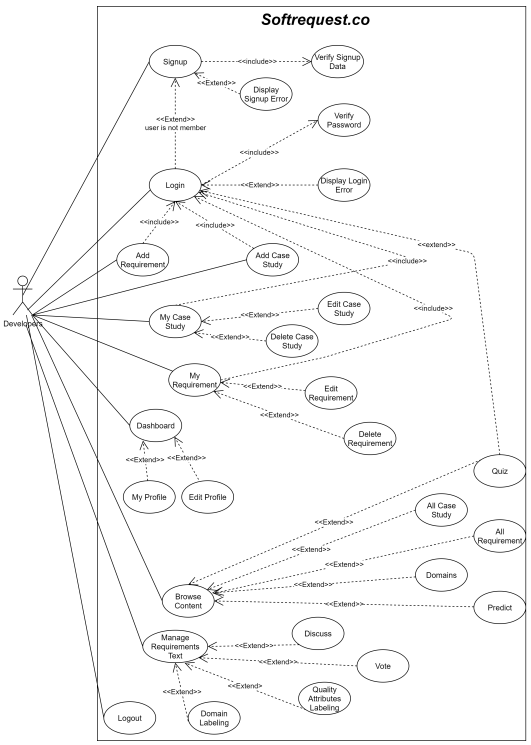


Fig. 3. *Softrequest.co* use case diagram.

For just gathering the opinions from the community, a stakeholder could also add only a requirements text without quality attributes. The scenario is compatible with the situation in which the developer seeks a suggestion from the community for a requirements text with vague characteristics like design constraints. The feature is similar with the *all-requirements-menu* form except for quality attributes labeling.

Another important feature is the capability to predict quality attributes from requirements texts using a pre-trained model. We have built the online deep learning model using TensorFlow that could train the model from all inputs periodically, and the best model will be used as a prediction function in the next application release. The prediction function can be seen in Fig. 4.

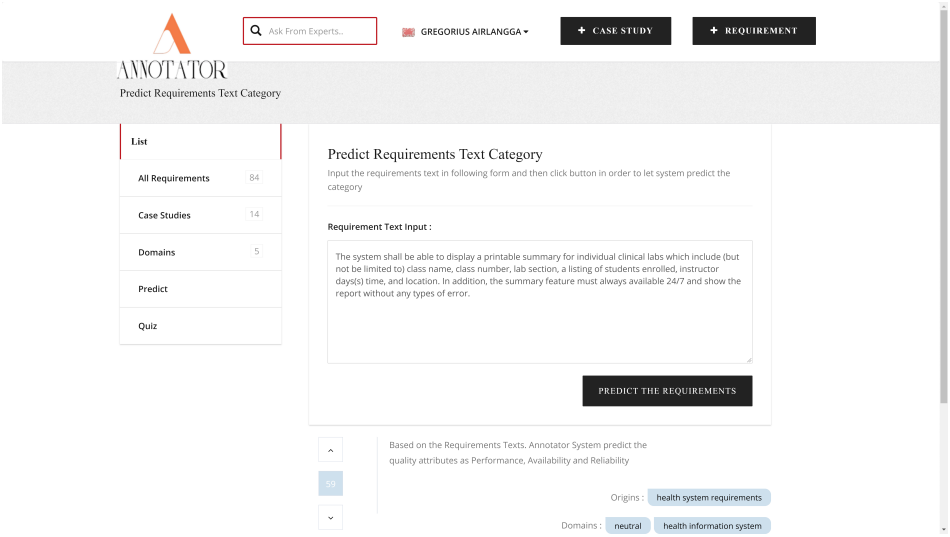


Fig. 4. Predicting the quality attributes.

4.4 Modeling

In this section we explain the deep learning based model called CNN and the overview predicting process from a raw requirements text into quality attributes. Also, the definition of training and evaluation metrics as indicators to answer the research questions are also revealed.

4.4.1 CNN model

We use notations from Section 2 in order to explain more precisely the model used in this experiment. As previously mentioned, the CNN method (τ) is used to acquire the classification function (γ). The requirements text (q) is tokenized and preprocessed into 300 dimension vector size. Using the vector embedding mechanism, we acquire the collection of high-dimensional space X representing a collection feature of the requirement text. Then, each of the outputs of these spaces will be reduced using the max pooling function before concatenating into a single tensor. In the last layer, the nonlinear softmax activation function will be used to decide the label, in which (q) is encoded by one-hot encoder, and each label space contains the prediction value probability between 0 to 1. A detail parameter setting used in this experiment can be seen in Table 3.

4.4.2 Training and evaluation

We have built three prediction models. The first model is the A model generated from the [3] dataset; the second model is the B model produced from the *softrequest.co* dataset; and the last model is the C model generated from the merged dataset from [3] and *softrequest.co*. Each model was generated using the CNN model and the word2vec approach that gave promising results in [6, 15, 16]. The information about the architecture

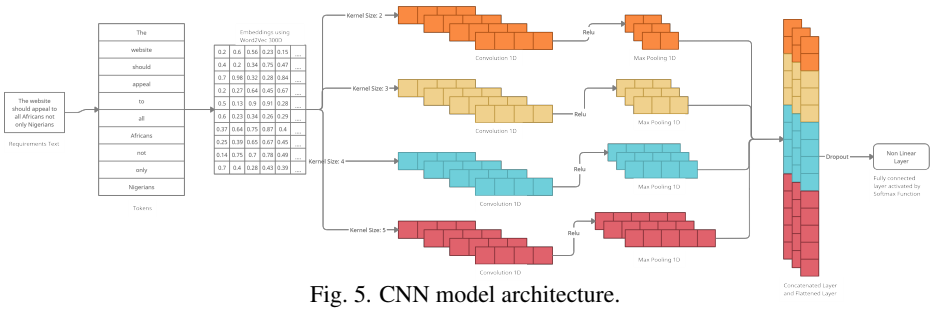


Fig. 5. CNN model architecture.

used is defined in Table 3. Each model has been generated by using 80% training data and 20% testing data. The evaluation metrics comprise four measurements: accuracy, macro average precision, macro average recall and the macro average F1-score.

$$Accuracy = \frac{True\ Positive}{True\ Positive + True\ Negative} \tag{1}$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

In order to calculate the evaluation scores, we use *scikit-learn* code in Python environment. The generated prediction model was then further used to investigate the impact of software domain in requirements quality prediction. First, we investigate the software domain effect by comparing the prediction performance between Model B and Model C to predict the [2] dataset and Model A to predict the *softrequest* dataset.

The domain of Model B is different from the domain of Model A, and the domain of Model C is partially different from Model A since the dataset [2] is also partially fed into Model C, constructing a model from mixed domains. Secondly, we also investigate the effect of data augmentation to show that prediction performance will increase if the trained data containing various domains by analyzing the results through Experiments I to V which are explained next.

4.5 Machine Learning Pipeline

The purpose of experiments is to investigate the quality attributes prediction performance in cross-domain condition. We predict unseen data from a different domain dataset. The classification performance was evaluated on multi-classification problem

containing 12 quality attributes as target classes. Performance measurement was conducted by analyzing the model's F1-score to predict quality attributes in the new domain. There were five experiments conducted in this work such as described in Table 4, the experiments contain five variants.

In Experiment I, we used Model A to predict the [2] test dataset. In Experiment II, we utilized Model A to predict the *softrequest.co* dataset, and Experiment III uses Model B to predict the [2] test dataset. Those experiments were very crucial to answer research questions about the performance of the trained model in terms of classifying the quality attributes from various software domains when the trained model itself was limited into a specific domain.

In Experiment IV, we generated Model C from mixed dataset to predict the [2] test dataset, and the experiment was also used as a supplementary result to investigate the impact of the partial domain in the prediction model. Furthermore, the F1-score and the testing accuracy results from Experiments I and V are compared to answer the second research question about the impact of data augmentation from various domains. Then a conclusion is drawn from analyzing the accuracy and F1-score results.

Table 3. Deep learning model architecture and parameter setting.

Layer Name	Parameter Value
Input Layer	Shape=(Number of Data x 300)
Embedding Layer	Shape=(Number of Data x 300 x 300)
Convolution 1D (a)	Kernel Size=3
Convolution 1D (b)	Padding=Same
Convolution 1D (c)	Activation=Relu
Max Pooling 1D (a)	Padding=same Activation=Relu strides=1
Max Pooling 1D (b)	strides=1
Max Pooling 1D (c)	
Concatenate Layer	
Flatten Layer	
Dropout Layer	Rate=0.4
Dense Layer	Neuron=12

Table 4. Experiments summary.

Experiment	Model used	To Predict/To Evaluate
I	A	[2] dataset (Baseline)
II	A	Softrequest dataset
III	B	[2] dataset
IV	C	[2] dataset
V	C	Merging dataset between [2] and softrequest

5. RESULT AND ANALYSIS

5.1 Answering Research Questions

Tables 5 and 6 report the experiment results. In these experiments, deep learning approaches were conducted on the three different software requirements datasets. The tables show the number of correct predictions made by the model with the evaluation metrics such as F1-score, precision, recall and the testing accuracy. For each dataset/metric, the best results are highlighted in bold except the baseline model labeled as "[ ]". In the following, we discuss the results aiming to answer our research question (RQ) about the trained model's performance to classify the quality attributes from different software domain requirements text.

Table 5. Prediction results to show the biased result.

Experiment	F1-Score	Precision	Recall	Accuracy
I	0.723	0.781	0.73	0.808
V	**0.934**	**0.961**	**0.945**	**0.963**

Table 6. Prediction result to prove domain-dependent.

Experiment	F1-Score	Precision	Recall	Accuracy
[I]	[0.723]	[0.781]	[0.73]	[0.808]
II	0.061	0.103	0.073	0.201
III	0.100	0.091	0.090	0.424
IV	**0.408**	**0.413**	**0.382**	**0.624**

Both of Models B and C were used to predict the [3] dataset as presented in Table 6 for Experiments III and IV. From these results, Model C achieves a better precision, recall and F1-score compared to Model B. In addition, Experiments II and III present low prediction results to predict quality attributes from the datasets of different domains. By analyzing those evaluation metrics, the main findings concerning the RQ include (i) From the comparison result between Experiments II and III, we conclude that the prediction accuracy of Model A in Experiment I is biased towards the quality attributes labels even if the accuracy and F1-score are 80% ; (ii) Since Model C gives better results compared to Model B, we assume that adding more dataset related to the software domain can improve data-driven prediction performance significantly as presented by Experiment IV in Table 6.

Although there is a significant improvement result from Experiment III compared to Experiment IV, the result is generally still inferior especially when compared to Experiment I. The detail explanation is given in Section 5.2. The inferiority is caused by the variation of requirements texts, the prediction model's incapability to catch semantic meaning, and the appropriateness of features from requirements texts. The detailed discussion is given below with the figures based on the results from the experiments. Fig. 6 (a) presents Model A's accuracy in predicting the test dataset in [2] requirements. The

accuracy is 80% to classify 12 quality attributes. Fig. 7 (a) shows Model A's accuracy to predict the test dataset in *softrequest.co* and Fig. 7 (b) presents Model B's accuracy to predict the test dataset in [2] requirements. The accuracy is inferior at 20.12% and 42.4% respectively. Fig. 6 (b) presents Model C's accuracy to predict the dataset in [2] requirements. The accuracy is higher compared to Model B but lower compared to Model A; in Model C, we get an accuracy of 62.4%.

Therefore, from the experiment IV, we can answer the additional RQ about the impact of data augmentation from different domains to classify the PROMISE dataset. It indicates that using data augmentation can generalize a better classifier to handle the variation of requirements highly, especially in a broad domain when the data training itself contains a similar domain with the test dataset. The biased conclusion about the result of Experiment I is in line with the results from Experiment V. According to the procedure in Section 4.4.2, we randomly split 20% of dataset containing domain from [2] and *softrequest.co*. As presented in Fig. 9, the accuracy of Model C to predict the mixed dataset is higher compared to Model C when only predicting the [2] dataset. The higher accuracy occurred since the validation dataset of *softrequest.co* is larger than the [2] dataset. In addition, there is an imbalance problem in the dataset that makes the word embedding vector mostly generated by the largest portion of dominant quality attributes such as functionality, security, usability and performance. Section 5.2 will provide detail explanation by observing the confusion matrix.

5.2 Confusion Matrix Analysis

In order to improve the analysis, we use macro average F1, precision, recall instead of weighted or micro average. The main reason is because each quality attribute has equal priority to detect. For example, the developers who stress the importance of "performance" will not expect "usability" more frequently. We expect the classifier function to detect any test dataset even the training and testing dataset is imbalanced. We do not use micro average since we cannot guarantee whether training and testing dataset are separated and the frequency distribution between each quality attributes will be equal or not. If it is not equal, we cannot use the total number of dataset labels in each evaluation metric. We also use a confusion matrix to find the reason about model performance. A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm. In this work, each row of the matrix represents the instances in an actual class, while each column represents the instances in a predicted class.

Model C (0.624 accuracy) yields worse performance comparing to Model A (0.808 accuracy) when both models are used to predict the [3] test dataset as shown in Experiments I and IV. From the confusion matrix in Figs. 6 (a) and (b), Model A only misclassified 1 data with true label "availability" into "functionality" compared to Model C that misclassified 3 data with true label "availability" into "functionality". The "functionality" quality attributes data in Model A are misclassified into operator (1), security (3) and usability (1) and about 51 data can be correctly classified as functional. However, Model C can classify 55 data correctly into functional requirements and only 1 data that misplaced as "operational". The attribute, "fault tolerance", in Model A was perfectly classified into ground truth. However, in Model C, only one data is correctly classified as "performance" and one other misclassified as "scalability". The quality

attribute, “Legal” is classified correctly in Model C; however in Model A only 2 data can be correctly classified, and the rest is classified as functional requirements. Interestingly, the quality attribute, “look and feel”, can be correctly classified in Model A with almost 78% accuracy (about 7 out of the total of 9 test data), in which one data is misclassified as operational requirements and one other is misclassified as functional requirements. On the other hand, Model C only can correctly classify 2 data as ground truth, the rest are classified as functional, operational, portability, scalability and security with values like 1,1,1,2,2 respectively.

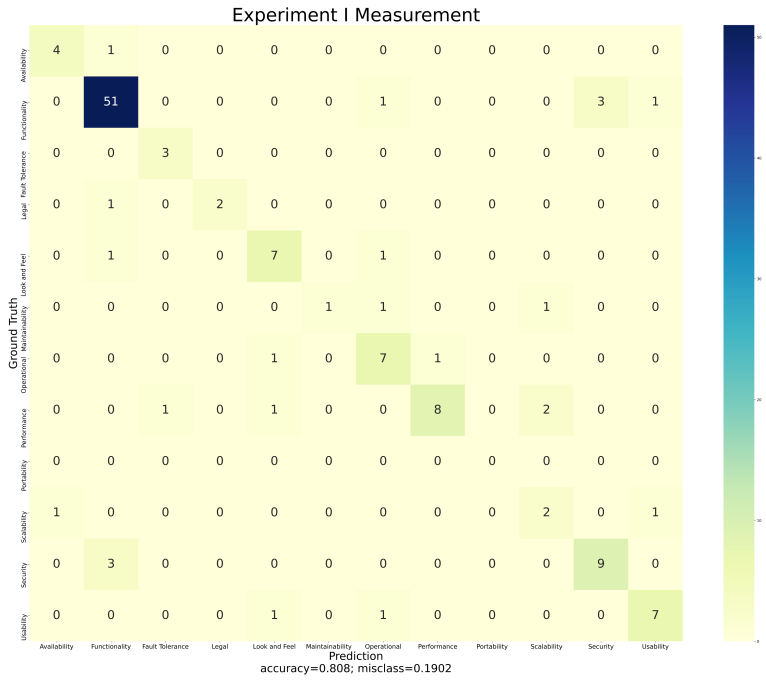
The performance concerning “maintainability” is worse in both models, in which none in Model C and only one in Model A produced a correct label. Model A achieves good performance in the operational quality attributes with almost 78% accuracy (7 out of 9) and only one data is misclassified as “performance” and the rest is misclassified as “look and feel”. For Model C, only one data can be correctly classified as operational requirements, while misclassifying as availability, functional and legal with values of 2. Also, one data is misclassified as “look and feel” and one as “performance”.

Model A can achieve a better result in the “performance” quality attributes. Eight are correctly classified, 2 as “scalability”, one as “look and feel” and one as “fault tolerance”. For Model C, only five were classified correctly, one is classified as “availability”, one as “fault tolerance”, one as “legal”, one as “look and feel”, one as “maintainability”, and 2 as functional requirements. For the “portability” quality attribute, both model did not provide reasonable results due to lack of data. For “scalability”, both models can only classify 2 each.

Model A misclassified one data as “usability” and one as “availability” while Model C misclassified one data as “maintainability” and one as “look and feel”. The “security” test dataset can be correctly classified in Model A with a 75% accuracy (9 out of 12). There are 3 misclassified as functional. In Model C, only one is correctly classified as “security”, 9 are misclassified as functional and 2 as “fault tolerance”. The quality attribute, “usability”, has same performance on two models: 7 correctly classified and one misclassified as ”operational” and “look and feel” in Model A, but as functional and operational in Model C.

Since we use a deep learning approach which is black-box, it is hard to understand the classifier behaviour exactly. However, based on our results, we can see that the misclassification occurred the most belongs to the functionality quality attribute which contained the largest proportion of dataset (about 40.4% of [3] dataset and 83% of *softrequest* dataset). A decision of false negative to “functional” happened 19 times with Model C and 6 times with Model A. The dominant number of false negatives in this finding indicates that the classifier in this experiment is trapped into high volume of dominant label. Therefore, it is hard for a classifier to predict data that has a small portion of training data. Instead of capturing the semantics of data, the classifier tends to capture the probability of word occurrence in each label.

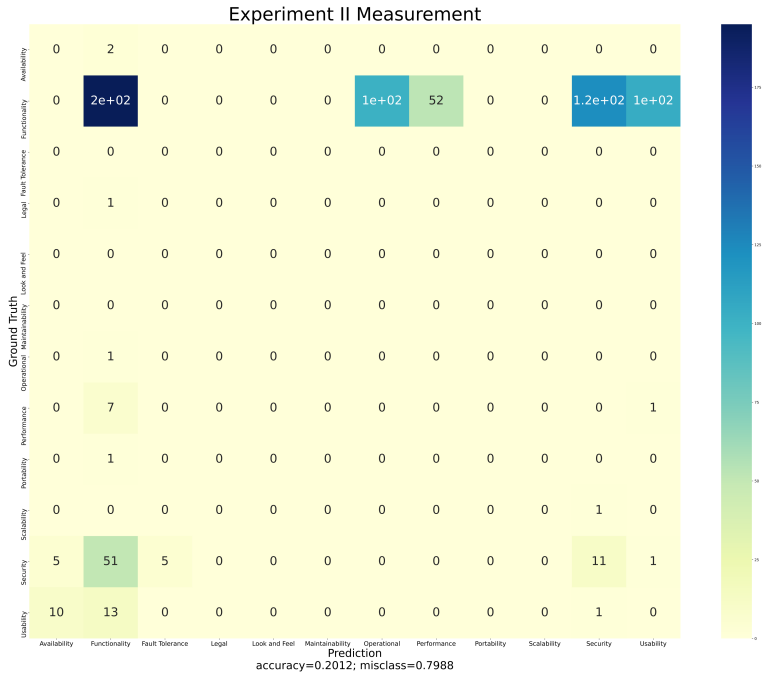
The main reason why Model C is worse than Model A is because when the C dataset is expanded, the probability density function of Model C became large and also the word vector embedding was expanded. The condition made some important features to affect the prediction. Moreover, since the increment of dataset is tend to gravitate on functional quality attributes, the smaller proportions of training dataset are mostly mixed into functional quality attributes. Therefore, we can see the phenomenon that most of the



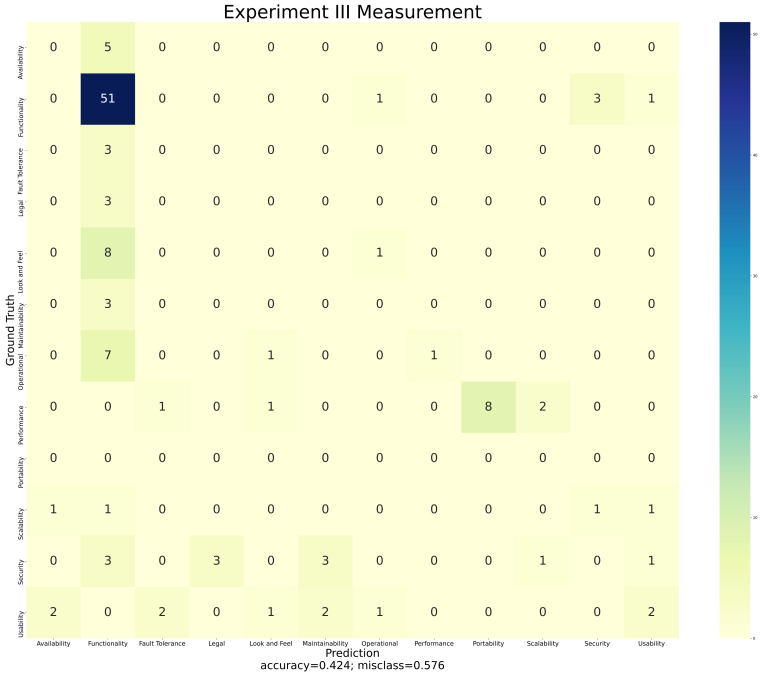
(a) Model A to predict [3] testing dataset.

(b) Model C to predict [3] testing dataset.

Fig. 6. Confusion matrix result to compare Experiments I and IV.

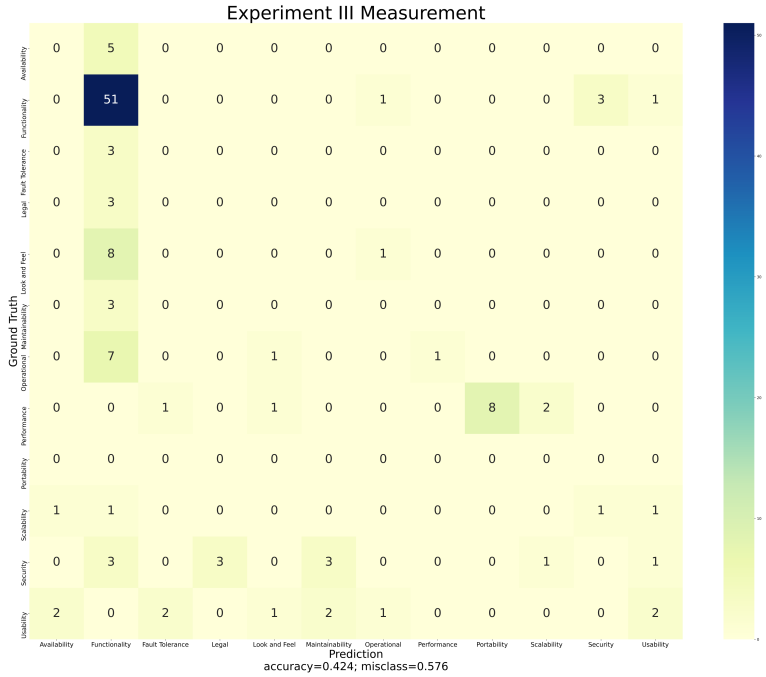


(a) Model A to predict softrequest testing dataset.



(b) Model B to predict [3] testing dataset.

Fig. 7. Confusion matrix result for answering research Question 1.



(a) Model B to predict [3] testing dataset.

(b) Model C to predict [3] testing dataset.

Fig. 8. Confusion matrix result for answering research Question 2.

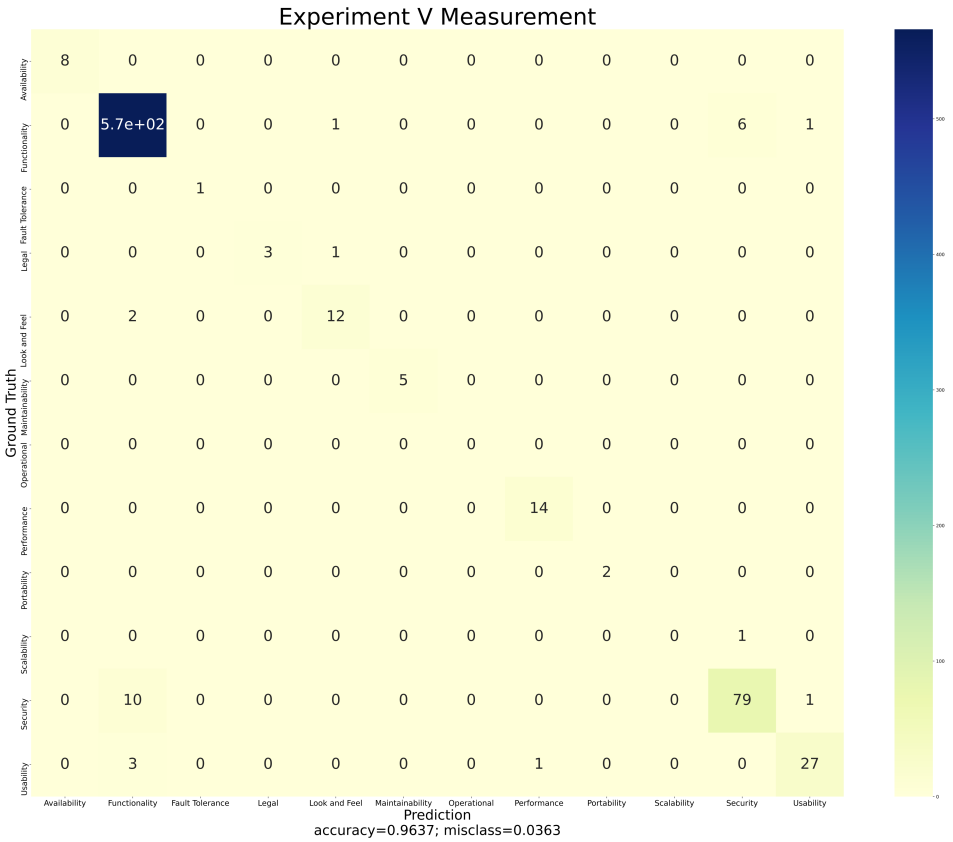


Fig. 9. Model C to predict mixed testing dataset.

[3] dataset would be classified as functionality comparing to others using Model C classifier.

From Experiments II and III, both test validation dataset has totally different word embedding. The confusion matrix in Figs. 7 (a) and (b) achieve the result that Model A mostly classifies the *softrequest.co* testing dataset as functional, operational, performance, security and usability. Interestingly, it is related to the number of dataset in each quality attributes which are functional (40.4%), security (10.5%), usability (10,2%), operational (9.9%) and performance (8.8%), in which the functional quality attribute is covered in the largest proportion of data. This condition is also similar in Model B when predicting with the [2] dataset. The text is mostly classified as the "functional" quality attributes, and this condition is similar to the phenomenon in Experiment IV which are already explained previously.

By analyzing the confusion matrix of Experiments I and V, obviously the results are biased. Both models cannot really capture semantics since they depend on the label distribution of dataset. Therefore, each word domain in training dataset must be considered carefully since it affects the word embedding that contributes for classification performance.

6. CONCLUSION

Based on our observation, the data-driven approach depends on the text semantic behavior between training and testing from the machine point of view. The experiments have shown that a prediction model cannot give precise classification to unforeseen variation such as disparate domain. Based on our findings, we recommend two awareness issues to other researchers in the related areas. First, adding more datasets may benefit from building a practical model, and the second, collecting a broader domain may improve the performance since the number of unique tokens and semantics might capture more robust text features than only one dataset. Accumulating experiences in requirements analysis concerning quality attributes remains challenging, and a useful annotator like the proposed prototype may benefit the requirements providers to focus more on quality attributes issues while collecting data for future usage.

We will investigate the fusion of data-driven and knowledge-driven approaches in online learning behavior for further research. The knowledge-driven approach may be able to resolve the prediction problem with the limited dataset and software domain in training data. The combined techniques can make the model adopt the specific rules and ontology to coordinate with data preprocessing and weight optimization in the deep learning methods. To support more reliable prediction systems in real-time, online learning should be implemented robustly to handle enormous text requirements dataset with unforeseen domain and natural language styles.

REFERENCES

1. F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "Requirements classification with interpretable machine learning and dependency parsing," in *Proceedings of IEEE 27th International Requirements Engineering Conference*, 2019, pp. 142-152.
2. J. C. Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, Vol. 12, 2007, pp. 103-120.
3. Z. Kurtanovic and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *Proceedings of IEEE 25th International Requirements Engineering Conference*, Vol. 25, 2017, pp. 490-495.
4. R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *Proceedings of IEEE International Conference on Advances in Computing, Communications and Informatics*, 2016, pp. 2027-2033.
5. L. Toth and L. Vidacs, "Comparative study of the performance of various classifiers in labeling non-functional requirements," *Information Technology and Control*, Vol. 48, 2019, pp. 433-445.
6. V. Fong, "Software requirements classification using word embeddings and convolutional neural networks," Master's thesis, Faculty of California Polytechnic State University, San Luis Obispo, 2018.

7. Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? A study of classifying requirements," in *Proceedings of IEEE 25th International Requirements Engineering Conference*, Vol. 25, 2017, pp. 497-501.
8. P. Singh, D. Singh, and A. Sharma, "Rule-based system for automated classification of non-functional requirements from requirement specifications," in *Proceedings of IEEE International Conference on Advances in Computing, Communications and Informatics*, 2016, pp. 620-626.
9. H. Alrumaih, A. Mirza, and H. Alsalamah, "Domain ontology for requirements classification in requirements engineering context," *IEEE Access*, Vol. 8, 2020, pp. 89899-89907.
10. T. H. Nguyen, J. Grundy, and M. Almorsy, "Rule-based extraction of goal-use case models from text," in *Proceedings of the 10th ACM Joint Meeting on Foundations of Software Engineering*, Vol. 10, 2015, pp. 591-601.
11. A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-based classification of non-functional requirements in software specifications: A new corpus and SVM-based classifier," in *Proceedings of IEEE 37th Annual Computer Software and Applications Conference*, Vol. 37, 2013, pp. 381-386.
12. T. Li and Z. Chen, "An ontology-based learning approach for automatically classifying security requirements," *The Journal of Systems and Software*, Vol. 165, 2020, pp. 1-12.
13. T. Al-moslmi, N. Omar, S. Abdullah, and M. Albared, "Approaches to cross-domain sentiment analysis: A systematic literature review," *IEEE Access*, Vol. 5, 2017, pp. 16173-16192.
14. J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *Proceedings of IEEE 24th International Requirements Engineering Conference Workshops*, Vol. 24, 2017, pp. 39-45.
15. M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using RNN variants for quality software development," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, Vol. 3, 2019, pp. 25-30.
16. F. Petcuşin, L. Stănescu, and C. Bădică, "An experiment on automated requirements mapping using deep learning methods," in *Proceedings of International Symposium on Intelligent and Distributed Computing*, 2019, pp. 86-96.

Gregorius Airlangga is pursuing Ph.D. degree in Department of Electrical Engineering of National Chung Cheng University in Taiwan. He is currently an Assistant Professor with the Department of Information System, Atma Jaya Catholic University of Indonesia. His research interest in artificial intelligence and software engineering include machine learning, natural language processing, deep learning, software requirements and architecture.

Alan Liu received his Ph.D. degree from the Department of Electrical Engineering and Computer Science of the University of Illinois, Chicago, in 1994. He is currently a Professor with the Department of Electrical Engineering, National Chung Cheng University, Taiwan. His research interests in artificial intelligence and software engineering include knowledge acquisition, requirements analysis, intelligent agents, case-based reasoning, service computing, and applications in smart home systems and robotic systems. He is a member of IEEE, ACM, Taiwanese Association for Artificial Intelligence, Software Engineerig Association of Taiwan, and Robotics Society of Taiwan.