5-2022

Identifying Functional and Non-functional Software Requirements from User App Reviews and Requirements Artifacts

Dev Jayant Dave

Abstract

This thesis proposes and evaluates Machine Learning (ML) based data models to identify and isolate software requirements from datasets containing user app review statements. The ML models classify user app review statements into Functional Requirements (FRs), Non-Functional Requirements (NFRs), and Non-Requirements (NRs). This proposed approach consisted of creating a novel hybrid dataset that contains software requirements from Software Requirements Specification (SRS) documents and user app reviews. The Support Vector Machine (SVM), Stochastic Gradient Descent (SGD), and Random Forest (RF) ML algorithms combined with the term frequency-inverse document frequency (TF-IDF) natural language processing (NLP) technique were implemented on the hybrid dataset. The performance of each data model was evaluated by metrics such as accuracy, precision, recall, and F1 scores, and the models were validated using 10 k-fold cross-validation. The proposed approach can successfully identify and isolate software requirements with SGD performing the best with an accuracy of 83%. Overall, this thesis presents a comprehensive methodology for implementing machine learning algorithms combined with NLP techniques to identify requirements from user app reviews with a high degree of accuracy.

Keywords: requirements, mining, classification, machine learning, natural language processing

MONTCLAIR STATE UNIVERSITY

Identifying Functional and Non-functional Software Requirements from User App Reviews and Requirements Artifacts

by

Dev Dave

A Master's Thesis Submitted to the Faculty of

Montclair State University

In Partial Fulfillment of the Requirements

For the Degree of

Master of Science

May 2022

College of Science and Mathematics

Department of Computer Science

Thesis Committee:

_____

Dr. Vaibhav Anu

Thesis Sponsor

[redacted]

Dr. Aparna Varde

Committee Member

[redacted]

Dr. Jiacheng Shang

Committee Member

[redacted]

IDENTIFYING FUNCTIONAL AND NON-FUNCTIONAL SOFTWARE REQUIREMENTS FROM USER APP REVIEWS AND REQUIREMENTS ARTIFACTS

A THESIS

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

by

Dev Jayant Dave

Montclair State University

Montclair, NJ

2022

Acknowledgements

I would like to thank my thesis sponsor, Dr. Vaibhav Anu, and my committee members, Dr. Aparna Varde and Dr. Jiacheng Shang for their continued support and guidance throughout my graduate degree.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

The Software Development Life Cycle (SDLC) is a process consisting of multiple stages used to define the design, development, and testing of software. SDLC consists of five stages that are requirements analysis, design, development, testing, and maintenance. Requirements analysis is the elicitation of requirements that define what the software will do. The design stage involves creating prototypes of the software to illustrate basic functionality. The development stage involves writing code for the software whereas testing involves testing the code to ensure the piece of software functions as intended. Lastly, maintenance consists of monitoring the software to resolve any issues that may arise or implementing enhancements. The requirements phase in the SDLC is one of the most important stages of software development. Incorrect or missing requirements can lead to an incomplete product that does not satisfy customer demand. The quality of the software requirements specification (SRS) document will reflect the quality of the final software product. The SRS document outlines the functional and non-functional capabilities of an application. Thus, the development team and the client must share the same understanding [1]. Software requirements are gathered from clients who outline the functionalities of the system. Requirements can also be gathered from end-users during the testing of the software and by domain experts that can include any missed requirements. Lastly, software requirements can also be gathered from previous software development projects or case studies.

In recent years, the growth of mobile devices has led to an increase in mobile software. App distribution platforms such as the google play store and apple app store have had 4 million apps as of June 2016 with the number of monthly app downloads hovering around 1 billion per month [2]. Users that download these applications can rate the application and provide textual

feedback relating to the application. User reviews contain important information that can assist a developer in better understanding their user needs and wants. Making use of user reviews for app upgrades can lead to an increase in new users and retain existing users. Studies indicated that one-third of users have modified their app ratings after a developer's response. Mobile applications can receive more than 20 reviews per day and popular mobile applications such as Facebook can receive more than 4000 application user reviews a day [2]. Therefore, user reviews are an important source of feedback for developers to elicit requirements to provide software fixes or updates. However, it is difficult to sift through vast amounts of user reviews and filter out reviews that do not indicate a requirement. Automatic extraction of software requirements through various approaches and frameworks can enable developers to respond quickly to customer wants and needs and reduce the time and money spent on eliciting requirements. This thesis seeks to introduce a novel hybrid dataset consisting of Functional Requirements (F), Non-Functional Requirements (NFR), and Non-Requirements (NR) from Software Requirements Specifications (SRS) document and user app reviews. The hybrid dataset is used to create data models that use machine learning (ML) algorithms such as Support Vector Machine (SVM), Stochastic Gradient Descent (SGD), and Random Forest (RF) combined with natural language processing (NLP) techniques such as term frequency-inverse document frequency (TF-IDF). The data models are then thoroughly evaluated by using 10 k-fold cross-validation and calculating

TABLE 1: RESEARCH QUESTIONS

RQ No.	Research Question
RQ1	What is the type and size of data required to automate the identification of requirements from user app reviews?
RQ2	To what extent can the machine learning algorithms combined with Natural Language Processing (NLP) techniques accurately identify and classify Functional and Non-Functional Requirements?
RQ3	How effective are the data models in identifying Non-Requirements (NRs) from Functional Requirements (FRs) and Non-Functional Requirements (NFRs)?

accuracy metrics such as recall, precision, and the F1 score. Table 1 provides a list of research questions that guided the work described in this thesis.

Overall, the main contributions of this thesis are as follows:

1. *Identification and classification of Functional Requirements (FRs) and Non-Functional Requirements (NFRs) from formal requirements artifacts*: The automatic identification and classification of FRs and NFRs from SRS documents makes the requirements engineering (RE) phase more efficient as software engineers and project managers can capture requirements that may have been missed. Capturing requirements at a later stage of the SDLC increases the overall cost and time to deliver the project.
2. *Automatic identification of FRs, NFRs, and NRs from user app reviews*: Automatically identifying requirements from a host of user app reviews can save developers time by not having to manually sift through thousands of reviews. This also enables developers to better understand the wants and needs of their users.

Therefore, developers can iteratively improve their applications by being able to automatically extract requirements from user reviews.

3. *Development of new hybrid dataset consisting of formal software requirements statements as well as user app review statements*: This dataset will be useful for machine learning researchers who are interested in developing and evaluating ML models to extract software requirements from user app reviews.

The remainder of the thesis is divided into five chapters. Chapter 2 provides an overview of the current state of research for identifying and classifying FRs and NFRs from formal software requirements and user app reviews. Chapter 3 proposes the first study where FRs and NFRs are classified by using SGD, SVM, and RF ML algorithms. Chapter 4 proposes the second study where a novel hybrid dataset is created consisting of requirements from SRS documents and user app review statements. The requirements and user app review statements are classified into FRs, NFRs and, NRs by using the SGD, SVM and, RF ML algorithms. Each data model is evaluated by its performance metrics and is validated. Chapter 5 consists of the implications of this research in the real world. Chapter 6 presents the conclusion of both the studies and explores future work that can be done to further expand the studies presented.

Chapter 2: Related Work

Explicit requirements in SRS documents are well-defined functionalities of the system whereas Implicit Requirements (IMRs) are functionalities of a system that are assumed and not elicited during requirements gathering [3]. Unhandled IMRs can be a major contributor towards software failure [3]. Identifying and classifying functional requirements (FRs) and non-functional requirements (NFRs) from SRS documents and user app reviews can lead to a better quality of software. Requirements gathered from the stakeholders may not be well documented and it is up to the developer to meet the requirements as per their understanding. Vague requirements are an important factor that often leads to poor quality software and results in the failure of software projects. Additionally, many requirements are not initially captured but do get captured at later stages of the software development life cycle by end-users or clients before the software is moved to production [1]. For reader's reference, definitions, and examples for FRs and NFRs are provided the paragraphs below.

A functional requirement outlines the required behavior in terms of required activities, such as reactions to inputs, and the state of each entity before and after an activity occurs. It states the following [4]:

1. What will the system do?
2. When will the system do it?
3. Are there several modes of operation?
4. What kinds of computations or data transformations must be performed?
5. What are the appropriate reactions to possible stimuli?

A Non-Functional requirement describes some quality characteristics that the software solution must possess. It should outline the following [4]:

1. Performance
2. Security
3. Reliability and Availability
4. Maintainability
5. Usability and Human Factors
6. Precision and Accuracy
7. Time to Delivery/Cost

Examples of FRs include:

- The website will allow customers to search for movies by title actor or director.
- The product shall be able to delete room equipment.

Examples of NFRs include:

- The product should be able to be used by 90% of novice users on the Internet.
- System shall let administrator de-activate a customer account in under 1 minute. Customer will no longer be able to access the website.

Therefore, it is important to identify and classify software requirements during the requirements gathering phase to make certain that the software development projects meet client requirements, are within the budget, and are completed within the decided timeline. Tools such as COTIR have been proposed that integrate Commonsense Knowledge, Ontology, and Text mining for early identification of Implicit Requirements to reduce the time and effort spent by software engineers in identifying IMRs from large SRS documents [5] [6]. Additionally, deep learning approaches utilizing Convolutional Neural Network (CNNs) have been integrated with COTIR to better detect IMRs from complex SRS

documents that contain images and tables [7] [8]. The remainder of the section is divided into exploring the state of research for classifying formal software requirements from SRS documents and classifying requirements from user app reviews.

The following sections provide a review of literature on previous work performed on classifying formal software requirements and work done on identifying requirements from user app reviews.

2.1: Existing research on classifying formal software requirements

This section will explore the current state of research in classifying formal software requirements from SRS documents. The section is divided into several subsections with each subsection providing an overview of the approach or tool used for classifying formal software requirements.

2.1.1 Machine Learning

Binkhonain and Zhao [9] provide an overview of various machine learning algorithms and their performance to classify NFRs. 16 distinct machine learning algorithms are evaluated. Out of the 16 ML algorithms, 4 are unsupervised, 5 are supervised, and 5 are semi-supervised ML algorithms [9]. Table 2 consists of all the ML algorithms grouped by the type of algorithm.

TABLE 2: LIST OF ML ALGORITHMS GROUPED BY THE TYPE OF ALGORITHM IN THIS STUDY

Supervised	**Semi-Supervised**	**Unsupervised**
Support Vector Machines (SVMs)	Expectation-Maximization (EM)	Latent Dirichlet Allocation (LDA)
Naïve Bayes (NB)	Self-training	K-means
Decision Tree (DT)	Active learning	Hierarchical Agglomerative
K-Nearest Neighbors (K-NN)	Random Subspace Method for Co-training(RAS-CO)	Biterm Topic Modelling (BTM)
Multinomial Naïve Bayes (MNB)	Relevant Random Subspace Method for Co-training (Rel-RASCO)	

The ML approaches consisted of similar data preprocessing steps. Data preprocessing consisted of selecting appropriate features and preprocessing the text input. The text was preprocessed by using methods such as stop word removal, stemming, and tokenization. Appropriate features were selected by converting the text into a numeric matrix using Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). Important features were then evaluated by using information gain and the Chi-squared test. Each ML approach was evaluated by metrics such as accuracy, precision, recall, and F1 scores [9]. Overall, the ML algorithms achieve an accuracy score of 70% when classifying NFRs. The supervised ML algorithms performed better than the unsupervised and semi-supervised ML algorithms with SVM and NB achieving the best performance [9].

2.1.2 InfoVis

InfoVis is a tool that seeks to identify requirements based on how ambiguous or incomplete a requirement is [10]. Data visualization techniques and NLP methods are combined to identify requirements. A novel algorithm, Semantic Folding Theory (SFT), is proposed that takes in user input and calculates a similarity score between pairs of words. The ambiguity score is also calculated based on the term and context similarity. Based on the similarity and ambiguity score, Venn diagrams are generated to visualize and explore the requirements [10]. The performance of the tool is evaluated by using the WebCompany dataset that contains 98 user story requirements. Tern pairs in the dataset are classified as being low, medium, or high ambiguity. Students manually classify the ambiguity level of the user store requirements. The ambiguity classification score by the students and the InfoVis tool is highly correlated suggesting that InfoVis is effective in identifying requirements that are ambiguous [10].

2.1.3 Templates

Templates can be used to identify security requirements. Riaz et al. propose using templates that suggest security requirements to aid the process of eliciting requirements [11]. The template provides a list of important security requirements to be included to the developers as they are gathering the requirements. Requirements are given as input to the template and based on the requirements, a list of security requirements is created [11]. The template can serve as a helpful tool to be used with other tools that also focus on addressing security requirements and privacy concerns.

2.2: Existing research on identifying requirements from user app reviews

This section brings forth the current state of research for identifying requirements from user app reviews. The section is divided into several subsections. Each subsection provides an overview of the approach or tool used for identifying requirements from user app reviews.

2.2.1: Automatic classification of Non-functional and Functional requirements

Lu and Liang proposed four classification techniques for classifying FRs and NFRs [12]. NFRs are further classified into reliability usability, portability, and performance. NLP techniques such as techniques Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), Chi-Squared (Chi2) and Augmented User Reviews Bag-of-Words (AUR-BoW) are used in conjunction with Naïve Bayes, J48, and Bagging ML algorithms to classify user reviews [12]. The ML algorithms were evaluated by calculating the precision, recall, and F-measure scores. 10 k-fold cross-validation was performed to validate each approach. The AUR-BoW algorithm performed the best with an F-measure score of 71.7%, a precision score of 71.4%, and a recall score of 72.3%. Augmenting user reviews leads to better results when classifying user reviews [12].

2.2.2: Requirements Mining Framework

A requirement mining framework for mobile upgrades is proposed by Chen et al [13]. A ranking model is developed that can classify customer requirements and rank the importance of each requirement. The performance of the framework is evaluated by product quality improvements. User reviews are transformed into product upgrade requirements. The framework consists of the following key components: context-aware segmentation, opinion target extraction, opinion target grouping, and requirements summarization [13]. An empirical analysis

is conducted to evaluate the effect of requirements mined from customer reviews on app upgrades. The following metrics are used to evaluate the framework [13]:

1. AdScore (Adoption Score): Measures how well actual app updates matched the recommendation results from the framework
2. DifRating: Average rating difference between version k and version k+1 of the mobile app
3. DifDownload: Average difference in downloads and duration of version k and version k+1 of the mobile app

Four experiments were conducted on four case sets (E1 – E4). C1- C5 represent different app categories which are Shopping, Social, Contact, Camera, News, and Game. R1 represents count-based ranking and R2 represents count and rating-based ranking. The four case sets are defined as follows:

1. E1: Contains all cases
2. E2: Cases with DifRating > 0
3. E3: Cases with DifDownload > 0
4. E4: Cases with DifRating > 0 and DifDownload > 0

Case set E4 has the highest AdScore rating throughout the different app categories which indicates that cases that adopted app upgrade suggestions from the proposed mining framework led to better quality app upgrades as measured by the increase in rating, download count, and app usage duration after an app was upgraded [13].

2.2.3: Information Retrieval and NLP approach

Information retrieval with NLP techniques is proposed by Yang and Liang to identify and classify FRs and NFRs [14]. This approach consists of a User Reviews Extractor that uses an

API to collect user reviews of iBooks from the app store and the Requirements Identifier and Classifier is used to automatically identify and classify software requirements. The data is preprocessed by techniques such as stop word removal. After preprocessing, the TF-IDF NLP technique is used to determine the importance of a word. Words classified as important are then added to a keyword set that is used to identify and classify user reviews [14]. The effectiveness of this approach is evaluated by comparing the results of the manual classification of requirements by experts. The precision, recall, and F-measure scores are calculated for the automated classification of requirements using various sample sizes. The F-measure score increases when the sample size is increased from 1 to 20 for FRs and from 1 to 7 for NFRs indicating that a certain sample size is required for better performance [14]. Classifying FRs require a larger sample size when compared to NFRs as FRs tend to be domain-dependent. With an adequate sample size, this approach can achieve a good F-measure, recall, and precision score for classifying FRs and NFRs [14].

2.2.4: Text Classifiers

Williams and Mahmoud proposed an approach to classify text using ML algorithms such as Support Vector Machines (SVM) and Naïve Bayes (NB) combined with NLP techniques such as TF and hybrid TF-IDF to mine Twitter feeds to automatically gather software user requirements [15]. 4,000 tweets regarding 10 software systems are collected and manually classified into informative and uninformative messages. The tweets are further classified into Bug reports, User Requirements, and Other. After the data is collected, textual features are extracted using Textual Content (BOW), Text Processing, and Sentiment Analysis to improve the results of the SVM and NB classifiers. Sentiment analysis is conducted using Sentistrength which rates each word as positive or negative. Both classifiers, NB and SVM, achieved the best

recall, precision, and F-measure scores. NB achieved a precision score of 0.74, a recall score of 0.77, and an F-measure score of 0.76 whereas SVM achieved a precision score of 0.79, a recall score of 0.75, and an F-measure score of 0.77 [15]. However, sentiment analysis did not improve the scores of the classifiers due to software-related tweets being neutral and not polarizing [15]. Sentiment scores are largely neutral for bugs, user requirements, and other miscellaneous tweets.

Chapter 3: Study 1: Creating ML Models for Classifying Formal Software Requirements

The primary goal of this initial study was to develop ML-based models to classify FRs and NFRs and the subclasses of NFRs from formal requirements artifacts (known as SRS documents). This study used the publicly available PROMISE Repository of Software Engineering dataset to train and test the data models [16]. The TF-IDF NLP technique combined with SVM, SGD, and RF ML algorithms were used to create the data models. The performance of each data model was evaluated by performance metrics such as the accuracy score, precision, recall, and F1 scores. Additionally, each data model was validated by using 10 k-fold cross-validation to reduce model overfitting and any bias that may result from the random splitting of the train and test dataset.

3.1: Study 1: Research Methodology

Study 1 proposed a solution based on a comparative analysis of ML models combined with basic natural language processing (NLP). The PROMISE Software Engineering dataset consisted of 371 FRs and 255 NFRs. The NFRs were further classified into Availability, Legal, Look and Feel, Maintainability, Operational, Performance, Scalability, Security, and Usability classes. The distribution of FRs and NFRs and the subclasses of NFRs are shown in Tables 3 and 4 respectively. Study 1 consisted of four primary phases as follows:

1. **Data Preprocessing:** The dataset was first preprocessed before it is used as input for the ML algorithms. The NFR classes Fault Tolerance and Portability were removed due to inadequate observations. The stemming, tokenization, and stop word removal NLP techniques are applied to preprocess the data. Finally, the TF-IDF NLP technique is applied to convert the textual requirements into a numeric matrix. TF-IDF is a statistical measure that evaluates the frequency of a term in a document and offsets it by the frequency of the term across a set of documents. The numeric matrix can then be used to train the ML models. TF-IDF is defined as follows:
    a. tf-idf(t, d) = tf(t, d) * idf(t), where
        i. t: term
        ii. d: document set
        iii. idf is computed as idf(t) = log [n / df(t)] + 1 where n is the total number of documents

TABLE 3: FR AND NFR REQUIREMENTS IN THE PROMISE DATASET

Category	**Count**
Non-Functional Requirements (NFRs)	371
Functional Requirements (FRs)	255

TABLE 4: SUBCLASSES OF NFR IN THE PROMISE DATASET

Category	**Count**
Availability (A)	21
Legal (L)	13
Look and Feel (LF)	38
Maintainability (MN)	17
Operational (O)	62
Performance (PE)	54
Scalability (SC)	21
Security (SE)	66
Usability (US)	67

2. **Model Training**: The preprocessed dataset is randomly split with stratification into a training and testing set. The training set consists of 80% of the requirements and the testing set consists of 20% of the requirements. Random splitting with stratification is used to ensure that there is no unseen bias while ensuring the distribution of classes in the training and testing sets are similar to prevent model overfitting and underfitting. The training set is then used as input to the SVM, SGD, and RF ML algorithms to train the data models.
3. **Model Evaluation**: After the three data models are trained, their performance is evaluated by calculating their precision, recall, and F1 scores. The metrics are defined as follows:
    - Recall: Calculates the true positives in a class out of all the observations in the class. It is defined as True Positive (TP) / TP + False Positive (FP)
    - Precision: Calculates the number of true positives out of all the input classes. It is defined as TP / TP + False Negative (FN)
    - F1: Calculated based on the precision and recall scores. It is defined as 2 * Precision (P) * Recall (R) / P+R
4. **Model Validation:** Each model is validated using 10 k-fold cross-validation. 10 k-fold cross-validation is used to ensure the consistent performance of the data models. The data is split into 10 equal-sized groups. 10 iterations of model training and performance so that each iteration will consist of a distinct training and testing set. The average scores of all 10 groups are then calculated to compare the performance of each model.

Algorithm 1: Creating data models to identify formal software requirements from SRS documents

Input: Formal software requirements from SRS documents with a label indicating the type of requirement

Output: Predicted label for each software requirement in the testing set

Initialization:

1: Preprocess text data with stemming, tokenization, and stop word removal

2: Convert preprocessed data into a numeric matrix using TF-IDF

3: Split the dataset into training and testing sets with 80% and 20% of data respectively

4: Train SVM, SGD, and RF data modes on the training dataset

5: Evaluate the performance of trained data models on the testing set by calculating precision, recall, and F1 scores

6: Validate model with 10 k-fold cross-validation

3.2: Study 1: Experiment Results and Discussion

The set of experiments is split into two categories. The first set is to classify FRs and NFRs whereas the second set is to further classify NFRs into different subclasses. The precision, recall, and F1 scores of each respective data model are then calculated after 10 k-fold cross-validation. The results of the first set of experiments are shown in Figure 1. As shown in Fig. 1(a), SVM with TF-IDF produces the best results for identifying FRs. Fig. 1(b) shows that SGD produces the best scores for identifying NFRs. However, all three of the models have similar performance metrics when classifying FRs and NFRs.

In the second set of experiments, the NFRs were classified into 9 subclasses: Availability, Legal, Look and Feel, Maintainability, Operational, Performance, Scalability, Security, and Usability classes. As shown in Fig. 2(a), 2(b), 2(e), 2(f), and 2(h), SVM combined with TF-IDF achieved the best results for the following NFR subclasses: Scalability, Operational, Maintainability, Look and Feel and Availability respectively. For the Security, Legal, and Usability NFR subclasses. SGD combined with TF-IDF achieves the best results as shown in Fig. 2(c), 2(d), and 2(i). For

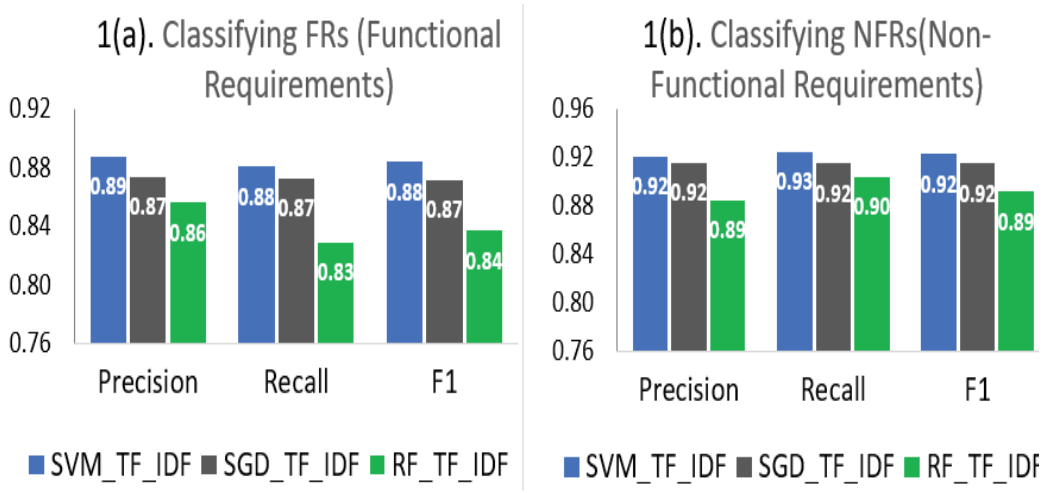

Figure 1: Classifying FRs and NFRs

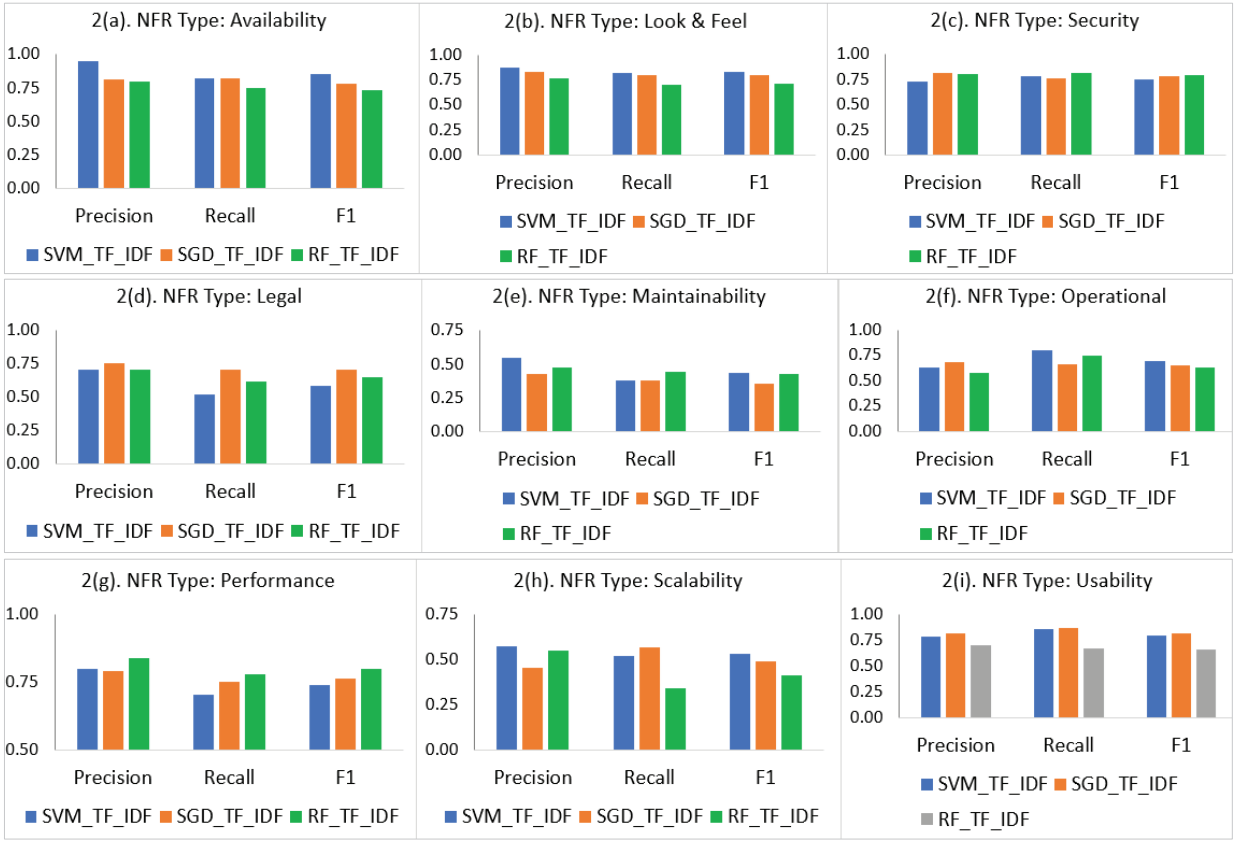

Figure 2: Classifying NFR-subcategories 1

the performance NFR subclass, RF combined with TF-IDF achieves the best results as shown in Fig. 2(g).

3.3: Study 1: Inferences

The issue of identifying and classifying FRs and NFRs has been addressed by previous research [10] [11]. Study 1 proposes a new approach that goes further than the current state of literature by presenting a thorough comparison and evaluation of multiple ML models combined with the TF-IDF NLP technique. Study 1 also presents a concrete methodology being defined by ML classifiers and NLP techniques to automate the identification of NFRs and NFRs as well as

identifying the subclasses of NFRs. The results from Study 1 were published in the IEEE Big Data conference [6]

Besides the introduction of a formalized method and improvement upon the current state of research, the results achieved by the two sets of experiments motivated Study 2 which seeks to identify FRs and NFRs from user app reviews to crowdsource requirements. Manual classification of thousands of user app reviews can be very time-consuming for app developers. The automatic identification of requirements would result in developers being able to quickly elicit requirements from user app reviews to better meet the wants and needs of their userbase. Chapter 4 provides a detailed discussion on Study 2.

Chapter 4: Study 2: Identifying FRs and NFRs from user app review data

Study 1 showcased that the ML models proposed performed well as per their precision, recall, and F1 scores. SVM combined with TF-IDF performed the best when identifying FRs with a recall score of 0.88, a precision score of 0.89, and an F1 score of 0.88, and SGD performed the best when identifying NFRs with a recall, precision, and F1 score of 0.92. For classifying the subclasses of NFRs, SVM performed the best when identifying Scalability, Operational, Maintainability, Look and Feel, and Availability requirements. SGD with TF-IDF performed the best when identifying Security, Legal, and Usability requirements. RF with TF-IDF performed the best when identifying Performance requirements. The performance of all three models prompted Study 2 which seeks to identify and classify requirements from user app reviews to crowdsource requirements. The automatic identification and classification of requirements can enable app developers to save time by quickly extracting requirements from user app reviews and further enhance their app as per user wants and needs.

4.1 Study 2: Development of a New Hybrid Dataset

A novel hybrid dataset was developed for Study 2. This hybrid dataset consisted of formal software requirements from the PROMISE Repository of Software Engineering [16] and user app reviews from the dataset prepared by Maleej et al. is proposed [17]. The PROMISE dataset contains 626 software requirements that are classified into FRs and NFRs. The NFRs are further classified into Availability, Legal, Look & Feel, Maintainability, Operational, Performance, Scalability, Security, and Usability. Fault Tolerance and Portability requirements are removed due to having a very small sample size. The Maleej dataset contains 3691 reviews from Apple's app store and Google's play store. The dataset is classified into Feature Requests,

TABLE 5: DISTRIBUTION OF FRs, NFRs AND NRs IN THE NOVEL HYBRID DATASET

Category	Count
Functional Requirement (F)	507
Non-Functional Requirement (NFR)	371
Non-Requirement (NR)	371

Bug Reports, Rating, User Experience, and Problem Discovery. Requirements in both datasets were relabeled to form a common set of labels. The subclasses of NFRs were relabeled as NFR and the FR label was unchanged in the PROMISE dataset. In the Maleej dataset, Feature Requests were relabeled as FR and the rest of the requirements were relabeled as Non-Requirements (NRs). To tackle the issue of class imbalance, the majority class, NR, was downsampled to match the count of the minority class, NFRs, with 371 observations. The final dataset distribution with relabeling and down sampling is shown in Table 5.

4.2: Research Methodology

Previous studies have focused on identifying and classifying FRs and NFRs from requirements artifacts such as SRS documents or user app reviews [1] [13] [15]. Study 2 proposes a hybrid approach to identify and classify requirements from SRS documents and user app reviews by implementing ML algorithms and NLP techniques. Figure 3 provides an overview of the stages of study 2. Study 2 consists of the following stages:

1. **Data Collection:** A novel hybrid dataset consisting of formal software requirements and user app reviews is proposed. The novel hybrid dataset will be used to train and test the ML models.
2. **Data Preprocessing:** NLP techniques such as stop word removal, stemming, tokenizing, and lemmatizing are applied to preprocess the data. The text is then converted into a numeric matrix using TF-IDF.

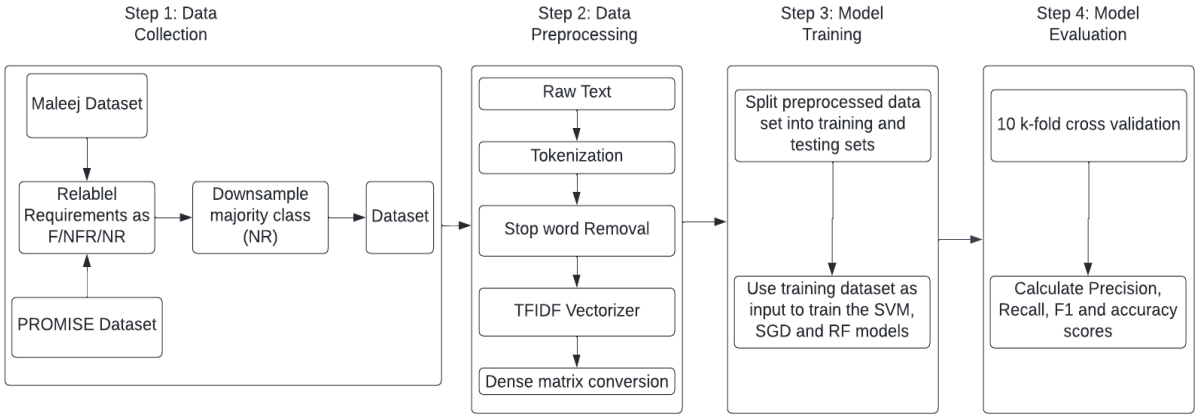

Figure 3: Stages in Study 2

3. **Model Training and Evaluation:** After the dataset is converted into a numeric matrix, it is split into training and testing sets. The training set consists of 80% of the data and the testing set consists of 20% of the data. The dataset is randomly split with data stratification to reduce any bias that may arise due to the random splitting of data. Stratification ensures that the training and testing sets have a similar distribution of FRs and NFRs. The training set is then used to train the SVM, SGD, and RF ML data models. The performance of each data model is evaluated by calculating their respective accuracy, precision, recall, and F1 scores.
4. **Model Validation:** Each model is then validated using 10 k-fold cross-validation. 10 iterations of model training and testing are conducted with each iteration having a distinct training and testing set. The average accuracy, precision, recall, and F1 scores of the 10 iterations are used to evaluate and compare the performance of each model.

Algorithm 2:Creating data models to identify software requirements from SRS documents and user app reviews

Input: Software requirements from SRS documents and user app reviews with a label indicating the type of requirement (FR, NFR, NR)

Output: Predicted label for each software requirement in the testing set

Initialization:

1: Form the novel hybrid dataset by combining the PROMISE and Maleej datasets

2: Relabel requirements in the combined dataset to FR, NFR, and NR labels

3: Down sample majority class, NR, to the count of the minority class, NFR, to form the final dataset

4: Preprocess text data with stemming, tokenization, and stop word removal

5: Convert preprocessed data into a numeric matrix using TF-IDF

6: Split the dataset into training and testing sets with 80% and 20% of data respectively

7: Train SVM, SGD, and RF data modes on the training dataset

8: Evaluate the performance of trained data models on the testing set by calculating precision, recall, and F1 scores

9: Validate model with 10 k-fold cross-validation

4.3: Experiment Results and Discussion

This section describes the results of each data model. The average accuracy, precision, recall, and F1 scores were calculated after 10 k-fold cross-validation of each data model. SGD combined with TF-IDF performed the best in identifying FRs with an accuracy score of 0.833, an F1 score of 0.788, a recall score of 0.774, and a precision score of 0.81. SGD with TF-IDF also performed the best when identifying NFRS with an F1 score of 0.913, a recall score of 0.916, and a precision score of 0.91. SVM achieved similar performance metrics as SGD when identifying NFRs. SGD with TF-IDF achieved the best results when isolating NRs from FRs and NFRs with an F1 score of 0.802, recall score of 0.83, and a precision score of 0.783. SVM and RF had similar performance metrics but were not as good as SGD. Figure 4 presents the performance metrics for each data model. The results cast a new light on the effectiveness of the data models proposed in classifying user app reviews and formal software requirements. From the results, it is clear that ML-based models combined with NLP techniques are an effective method to identify and classify FRs and NFRs and isolate NRs.

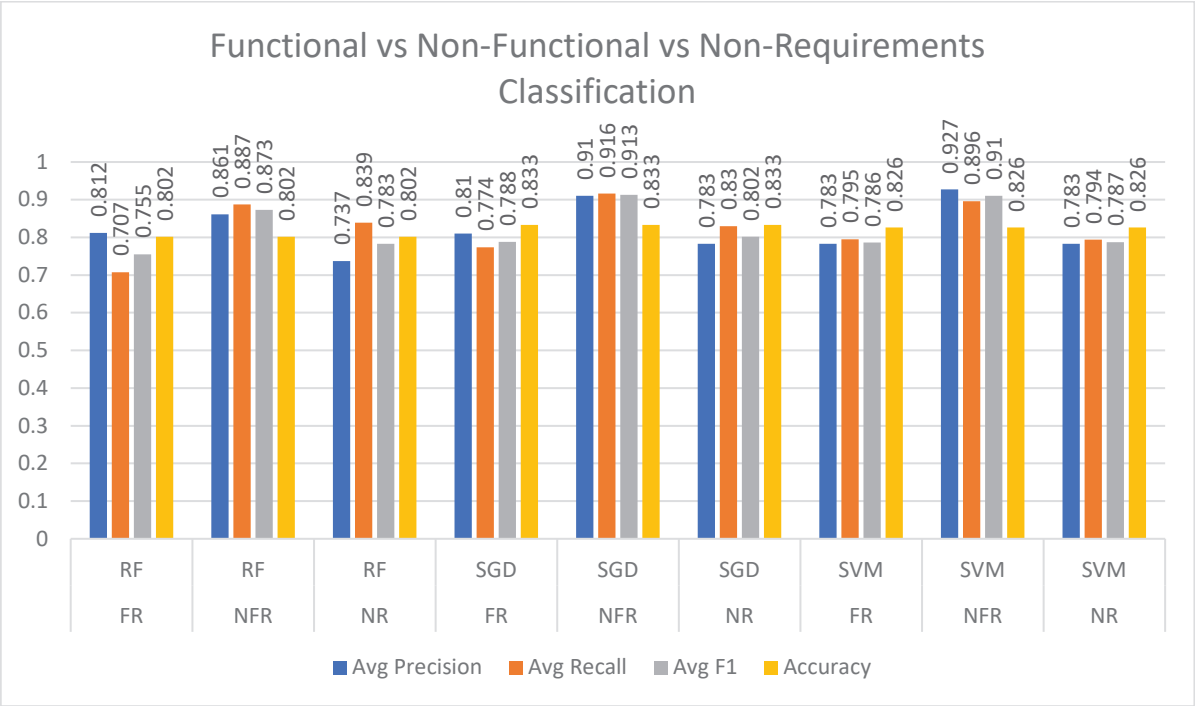

Figure 4: Performance metrics of SVM, SGD, and RF data models

Chapter 5: Discussion

This section discusses the results and implications of the research performed as part of this thesis (i.e., Study 1 and Study 2). This section also answers the research questions proposed in this thesis (for research questions, please see Table 1 in Chapter 1). Study 1 proposed three data models using the SVM, SGD, and RF machine learning algorithms combined with the TF-IDF NLP technique to classify FRs, NFRs, and the subclasses of NFRs from SRS documents that contained formal software requirements. The data was preprocessed to ensure that all the observations in the dataset were properly labeled and NFR subclasses with a very low count were disregarded. The TF-IDF NLP technique was then used to vectorize the data. The vectorized data was split into a training and testing set. The SVM, SGD, and RF ML algorithms were then trained on the training set. Each data model was then evaluated by calculating the accuracy, precision, recall, and F1 scores. After model evaluation, each model was validated with 10 k-fold cross-validation to ensure consistent model performance and to avoid model overfitting and underfitting. Based on the positive results of Study 1, a second study, Study 2, was proposed to identify and classify FRs, NFRs, and NRs from a novel hybrid dataset consisting of formal software requirements and user app reviews. The research questions of this thesis are answered as follows:

RQ1*: What is the type and size of data required to automate the classification of requirements from SRS documents and user app reviews?*

We proposed a novel approach to create a hybrid dataset using the PROMISE and Maleej datasets. The PROMISE dataset consists of formal software requirements from SRS documents and the Maleej dataset consisted of user app review statements from Apple's app store and Google's play store. All requirements and user app reviews in the datasets were relabeled to FR,

NFR, and NR to create a common set of labels. Next, the issue of class imbalance was resolved by downsampling the count of the majority class, NR, to the count of the minority class. Relabeling the requirements in both datasets and downsampling ensured the proper labeling of all observations and reduced any bias that may have been introduced by the majority class. The final dataset consisted of 507 FRs, 371 NFRs, and 371 NRs. The hybrid dataset consisted of a relatively equal count of observations in each category.

RQ2*: To what extent can the Machine Learning Algorithms combined with NLP techniques accurately identify and classify Functional and Non-Functional Requirements?*

The accuracy, precision, recall, and F1 scores of each data model were evaluated in Study 1 and Study 2. Study 1 identified and classified FRs and NFRs. SVM with TF-IDF performed the best when identifying FRs with a precision score of 0.89, a recall score of 0.88, and an F1 score of 0.88. SGD performed the best when identifying NFRs with precision, recall, and F1 score of 0.92. Positive results from Study 1 prompted Study 2 which included the creation of a novel dataset consisting of formal software requirements and user app reviews. The SVM, SGD, and RF models were implemented on the novel hybrid dataset to classify requirements and user app reviews as FRs, NFRs, and NRs. SGD with TF-IDF performed the best when identifying FRs with an accuracy score of 0.833, an F1 score of 0.788, a recall score of 0.774, and a precision score of 0.81. SGD with TF-IDF performed the best when identifying NFRs with an F1 score of 0.913, a recall score of 0.916, and a precision score of 0.91.

RQ3*: How effective are the data models in identifying Non-Requirements (NRs) from Functional Requirements (FRs) and Non-Functional Requirements (NFRs)?*

Study 2 proposed isolating NRs in addition to identifying FRs and NFRs. NRs are general feedback for an application and do not specify any requirement. SGD with TF-IDF

performed the best when isolating NRs from the novel dataset. It achieved an F1 score of 0.802, a recall score of 0.83, and a precision score of 0.783. The high scores suggest that the SGD data model is effective in isolating NRs from FRs and NFRs.

5.1 Real-World Impact

Manually going through a large number of user app reviews to extract requirements can be a time-consuming process, especially with many new user app reviews every day. Time spent on manual identification of requirements can be spent on development-related activities instead. Therefore, the automatic identification of requirements from user app reviews is vital to app developers to elicit requirements from user app reviews to better address the wants and needs of their userbase in a time-efficient manner. Studies 1 and 2 prove that requirements can be accurately crowdsourced. The data models proposed can be used by app developers to quickly elicit requirements and enhance the next iterations of their applications. In conclusion, using the data models proposed in this thesis can save developers time and effort by not having to manually identify requirements and being able to quickly incorporate much-wanted functionality into their applications.

Chapter 6: Conclusion

This section presents the conclusions drawn from Studies 1 and 2. It summarizes the findings of both the studies, highlights the primary contributions and presents avenues for future work.

6.1: Summary

In this thesis, three ML data models were created using the SVM, SGD, and RF machine learning algorithms. Two studies were conducted to evaluate and validate the performance of each data model. The first study used the PROMISE dataset consisting of formal software requirements from SRS documents that were classified as FRs, NFRs, and the subclasses of NFRs. The dataset was preprocessed using techniques such as stop word removal and stemming. The dataset was then converted into a numeric matrix using the TF-IDF NLP technique. After data preprocessing, a training and testing set was created. The data models were trained on the training set and their performance was evaluated on the testing set. The models were validated using 10 k-fold cross-validation and the precision, recall, and F1 scores were calculated.

6.2: Contributions and Findings

SVM had the best performance metrics when identifying FRs and SGD had the best performance metrics when identifying NFRs. The performance metrics of the data models from Study 1 prompted Study 2 which identified FRs, NFRs, and NRs from a novel dataset containing formal software requirements and user app reviews. SGD performed the best when identifying FRs, NFRs, and NRs. However, SVM had similar precision, recall, and F1 scores. Studies 1 and 2 prove that ML-based data models are effective in identifying FRs, NFRs, and NRs. The studies propose a concrete methodology for the automatic classification and identification of requirements with ML classifiers and NLP techniques. The automatic identification and

classification of FRs and NFRs from SRS documents can enable project managers and software engineers to catch any requirements early during the RE phase which can result in reduced costs and a quicker time to delivery for a software project. Automatic Identifications of FRs, NFRs, and NRs from user app reviews can assist developers in crowdsourcing requirements from their userbase. Automatically crowdsourcing requirements enable app developers to quickly sift through a huge number of app reviews to extract requirements. They can use crowdsourced requirements to enhance the next iteration of their app by meeting the wants and needs of their users.

6.3: Future Work

Future work for this research includes finding new sources of data to create new datasets consisting of formal software requirements and user app reviews. NLP techniques such as Bag-of-Words (BoW) [12] and Word Embedding for preprocessing textual data will be explored and ML algorithms such as XGBoost [18] and deep learning methods such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) [19] will be implemented and evaluated.

References

[1] O. Daramola, T. Moser, G. Sindre, and S. Bi. Managing implicit requirements using semantic case-based reasoning. In REFSQ, Springer LNCS, pages 7915:172-178, 03 2012.

[2] D. M. Fernandez, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetro, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius, et al. Naming the pain in requirements engineering. Empirical software engineering, 22(5):2298-2338, 2017..

[3] E. Onyeka, A. S. Varde, V. Anu, N. Tandon and O. Daramola, "Using Commonsense Knowledge and Text Mining for Implicit Requirements Localization," 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), 2020, pp. 935-940, d.

[4] C. Page, Software engineering: Theory and practice. Willford Press, 2019..

[5] Emebo, O., & Varde, A. S. Early identification of implicit requirements with the COTIR approach using common sense, ontology and text mining, Tech Report, Montclair State University - Fulbright Scholarship Program, 2016..

[6] D. Dave, V. Anu, and A. S. Varde, “Automating the classification of requirements data,” 2021 IEEE International Conference on Big Data (Big Data), 2021..

[7] E. Onyeka, V. Anu and A. S. Varde, "Identifying Implicit Requirements in SRS Big Data," 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 6169-6171, doi: 10.1109/BigData47090.2019.9006086..

[8] Dev Dave, Angelica Celestino, Aparna Varde, Vaibhav Anu --- Management of Implicit Requirements Data in Large SRS Documents: Taxonomy and Techniques --- ACM SIGMOD Record, Mar 2022..

[9] Binkhonain, Manal, and Liping Zhao. “A Review of Machine Learning Algorithms for Identification and Classification of Non-Functional Requirements.” Expert Systems with Applications: X, vol. 1, 12 Mar. 2019, p. 100001., doi:10.1016/j.eswax.2019.10000..

[10] L. Dalpiaz. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP. In REFSQ, pages 119-135, 2018..

[11] M. Riaz, J. Slankas, J. T. King, and L. A. Williams. Using templates to elicit implied security requirements from functional requirements - a controlled experiment. In ACM-IEEE Intl. Symp. on Empirical Software Engineering & Measurement, ESEM, page..

[12] M. Lu and P. Liang, “Automatic Classification of Non-Functional Requirements from Augmented App User Reviews,” Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, 2017..

[13] R. Chen, Q. Wang, and W. Xu, “Mining user requirements to facilitate mobile app quality upgrades with big data,” Electronic Commerce Research and Applications, vol. 38, p. 100889, 2019..

[14] H. Yang and P. Liang, “Identification and Classification of Requirements from App User Reviews,” Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering, 2015..

[15] G. Williams and A. Mahmoud, “Mining Twitter Feeds for Software User Requirements,” 2017 IEEE 25th International Requirements Engineering Conference (RE), 2017..

[16] Sayyad Shirabad, J. and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada.

[17] hawari, assem (2019), “A dataset of Mobile application reviews for classifying reviews into software Engineering's maintenance tasks using data mining techniques”, Mendeley Data, V2, doi: 10.17632/5fk732vkwr.2..

[18] K. Srisopha, D. Link, and B. Boehm, “How should developers respond to App Reviews? features predicting the success of developer responses,” Evaluation and Assessment in Software Engineering, 2021..

[19] J. Winkler and A. Vogelsang, “Automatic classification of requirements based on Convolutional Neural Networks,” 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), 2016..