UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Solagbade Ayodele Enitilo

A Model for Detecting and Resolving Conflicts in Features Extracted from App User Reviews

Master's Thesis (30 ECTS)

Supervisor: Ishaya Peni Gambo, PhD

Tartu, 2022

A Model for Detecting and Resolving Conflicts in Features Extracted from App User Reviews

Abstract:

Many app developers use user feedback to improve their app's quality. User feedback is a type of a review that originates from different user types and covers various parts of the app. This thesis employs in-depth review analysis to guide app developers in the requirement elicitation process to identify areas of an app where an upgrade is essential. However, reviews from various users might conflict based on their perspectives and feature interest in the app. This thesis addresses the problem associated with detecting and resolving conflicting user reviews by formulating a robust taxonomy of conflict types, causes, and effects. This is because establishing conflicts in user reviews has its own set of semantic and lexical concerns, such as identifying the conceptual overlap that exists between user reviews and decoding the semantic inference of a review, i.e., the *"why, how*, and *what"* of an app feature. To find the solution to the previously described conflict, 64, 964 app reviews were tested and evaluated from three different mobile app sources. As a result a semantic rule-based tool was developed that integrates knowledge representation and linguistic rules to detect conflicts in reviews gathered from app users. The result emphasised the holistic awareness of domain knowledge for effective conflict identification and resolution in app reviews. In addition, the findings suggested a methodical approach for applying conflict analysis to strengthen software design requirements. As a conclusion, considering the comprehensive and interpretable disposition of the conflict identification rules, the thesis presents a solution to resolve the conflicts identified in app reviews.

Keywords: Conflict, review, application, natural language processing, features, requirement engineering.

CERCS: P170 Computer science, numerical analysis, systems, control

Tarkvararakenduste kasutajate arvustuste põhjal väljaselgitatud funktsioonides esinevate vastuolude tuvastamise ja lahendamise mudel

Lühikokkuvõte:

Rakenduste arendajad kasutavad tihti kasutajate tagasidet, et parandada oma rakenduste kvaliteeti. Kasutajate tagasideks on arvustused, mida on jaganud erinevate soovidega tarbijad kattes mitmeid erinevaid rakenduse osi. Käesolev lõputöö rakendab põhjaliku arvustuste analüüsi, et aidata arendajatel tuvastada rakenduse osasid kus täiustused on esmatähtsad. Tuleb ette olukordi kus tarbijate soovid on vastuolus teiste tarbijate soovidega, kuna neil on teised vaated ja otstarbed samale rakendusele. Uurimistöö käigus loodi rühmad konfliktide liikidest, põhjustest ja mõjudest aitamaks tuvastada ja lahendada vastuolulisi arvustusi. Seda oli oluline luua kuna vastuolude tuvastamine arvustustes on semantiliselt ja leksikaalselt komplitseeritud. Keeruline on tuvastada ühiseid ideid arvustustes ja mida arvustuses üritati väljendada, lisaks ei ole alati selge kuidas kasutada arvustust parenduste teostamiseks. Selleks hinnati ja testiti 64,964 arvustust kolmest erinevast mobiili rakendusest. Selle uuringu tulemusena loodi semantiliste reeglite alusel funktsioneeriv tööriist mis kasutab lingvistilisi reegleid, et tuvastada vastuolusid rakenduste kasutajate arvustustes. Resultaat keskendub holistilisele arusaamale domeeni teadmistest, et leida effektiivne vastuolu tuvastamise ja lahendamise meetod rakenduse arvustuses. Uurimise leiud näitavad, et on olemas metoodiline lahendus vastuolude analüüsiks parendamaks tarkvara disaini nõudeid. Arvestades, et vastuolude tuvastamise reeglid on põhjalikud ja tõlgendatavad, suutis antud uurimistöö lahendada rakenduste tagasisidedes tuvastatud konfliktid.

Võtmesõnad: konflikt, ülevaade, rakendus, loomuliku keele töötlemine, funktsioonid, nõue mentitehnika

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine(automaatjuhtimisteooria)

Contents

List of Figures

List of Tables

1 Introduction

The notable increase in internet usage and coverage in recent years have brought about the development of new components and services [2]. And quite recently, popular individual application platforms such as Google App store and iOS App store providing millions of mobile applications to users, have been generating huge text data in form of reviews [3, 4]. These stores permit their users to search, deploy and even buy software applications within minutes of connection. They also permit users to air their opinions on the usability and features of an application. These user opinions, also called reviews, contain analytical information that could help in product optimization [5].

These reviews assist new users decide whether or not they want to use or buy an application [6, 7], they help existing users to sort current challenges or request new features [2], they help analysts in producing reports, and as well as help the app designers to fix bugs and improve the quality of their next upgrade to meet user's expectations [8]. In most cases, making sense of these reviews can be difficult for some reasons. The first reason is the difficulty in analyzing the large volume of data available as reviews. Some popular apps have been found to garner as many as thousands of reviews per day. How much effort will an analyst, a new or existing app user, or the app designer need to make sense of such voluminous data?

The second reason is the quality of the reviews which may vary contextually from harsh criticisms, innovative feature suggestions, and dissatisfaction to commendations. These contextual sentimental mix increases the difficulty of deciding where or what the review is targeted at and whether it can be categorized as positive, negative, general, technical, critical, or trivial [8, 6]. However, similar problems have been encountered in the use of prior general internet data, and several techniques have been proposed and used for effective analysis and data usage. The recommendation system is one of such techniques that uses data to suggest and recommend related products to the e-commerce

app user, to ease users' navigation and increase their satisfaction [2]. Another technique is opinion mining used in analyzing people's sentimental opinions to identify patterns and emotions from internet media [9]. Remarkably, there is currently a growing interest in other techniques involving the extraction and use of mobile app reviews data using Natural Language Processing (NLP) and Machine Learning (ML) algorithms. Review data have been extracted (or mined) and used for sentiment analysis, app development, spam checks, and surveys [10].

But despite the advantages that the extraction and use of user reviews can do, directly analyzing these reviews for insights can cause misinformation because some of these reviews are spam, while some are pure deceptions [6, 11]. Therefore, there have been different calls for innovative solutions that can automate the mining of more specific information from app reviews [2]. Using the NLP and ML automated approach, studies have looked into just mining informative reviews for developers [12], identifying fraud apps from reviews [13, 8], identifying and mining key features from app reviews [2], mining actions and arguments from user reviews [6], extracting and analyzing the sentiment associated with relevant app features [14, 8], identifying, extracting and analyzing Non-Functional Requirements (NFRs) in reviews [15, 16, 17], and so on.

However, conflicts may exist within a review, as discussed earlier, and there is a dearth of innovative work on identifying and resolving these conflicts. Therefore, this work will attempt to develop a framework for extracting features from apps reviews, detecting the conceptual overlap to identify potential candidates of conflicts (or sentimental mix), and resolving them appropriately. Although, there is no universally accepted definition of conflict (Afzalur [18]) due to different interpretations relating to the specific domain of interest [19]. But in this study, we define conflict as follows:

> *"An interactive sequence of expression that is characterized by discontent, incompatibility, or disharmony relating to a common domain of interest."*

Detecting conflict in the corpus of sentences is a crucial issue for filtering sentence

pairs. For example, given two sentences, S_1 and S_2, the aim is to determine and establish that conflicts exist between the two sentences. In many cases, differences in numerical expressions or named entity expressions are a significant indicator of an S_1-S_2 conflict [20]. Our study is focused on establishing that conflict exists in app reviews based on knowledge representation and linguistic attributes. The resultant goal of our research is to reduce the effect of conflicting reviews on the requirements engineering (RE) curriculum of app development.

Identifying conflicts in app reviews is not an easy to-do task as it presents both lexical and semantic concerns [21, 22]. The lexical concern involves different lengths and tones of a review (which may be difficult to decode in some cases) given by the user. The semantic concern is a multi-stage process. First, we must capture a review text's action and associated effect(s). Secondly, we analyze the reviews to check for conceptual overlap (i.e., establishing the chronological connection between different components of an app). Lastly, we check if the conflicts detected are either conditional or temporal. A conflict is conditional if a certain phenomenon or activity is satisfied, while a temporal conflict deals with the coordination of the intra-activity within a user review.

Therefore, our study presents a rule-based semantic approach for detecting conflicts in app reviews. There are two key contributions of this study, which are:

- we blended language elements with extensive external knowledge sources (such as Wikipedia and WordNet) to establish interpretive semantic rules for discovering and managing conflicts in app reviews; and
- we formulate the conflict detection problem using reviews mined from different sources, which offers an extensive taxonomy of conflicts as well as the comprehensive semantics that assists the conflicts detection process.

1.1 Problem Statement

The competition in the mobile app development world requires developers to constantly interact with feedback, questions, and requests from users on how to develop better apps or improve the existing ones (Mueez *et al.*, 2018). This is so because users' perceptions, opinions, and feedback can significantly affect mobile app usage and the usage traffic, which in turn affects the app's economic value. There have been attempts to help developers make the best use of mobile apps reviews by mining and classifying the reviews for specific reasons [12, 8, 6]. However, some app reviews are still wrongly classified because they contain conflicts that most existing models do not seek to detect or resolve. As a result, the chances of a developer getting the right insights to improve the app from user reviews are limited. Therefore, this study investigates how conflicts exist in app reviews and provides a computational strategy for identifying and resolving such conflict structures.

1.2 Research Justification

As fun as developing mobile applications can be, the primary development aim fails to be 'just-for-fun.' Apps are developed to solve the users' social problems while providing financial gain or some other gain to either the administrative manager or app developer [23, 24, 25]. Therefore, to keep social problems away and continue rewarding innovators, studies like this are necessary to grow and improve mobile application development.

1.3 Research Scope

This study focuses on the analysis of reviews posted by app users. The analysis entails establishing that there is conflict in user reviews, identifying the conflicts, and resolving the conflicts to assist software engineers in their app development process. The app referred to in this study applies to all software application platforms, either mobile or

computer-based apps. This study is not concerned with evaluating the core functionality of an app, either social or anti-social. Rather, we are concerned about enhancing the quality of apps by analyzing app reviews and identifying and resolving conflicts in the reviews. Although there are a number of other individual mobile application platforms, this work will only examine mobile applications from only the Google App store and iOS App store for analysis.

1.4 Research Aim and Objectives

This research aims to develop a rule-based framework for extracting features from app reviews, detecting conceptual overlap to identify conflicts, and resolving them accordingly. The process will help software developers during the requirement elicitation and implementation analysis phase. The specific objectives of this study are fourfold: First, we extract and analyze users' reviews on some mobile applications. Secondly, we formulate a conflict detection and resolution framework from extracted user reviews. Thirdly, we simulate the proposed model based on the formulated framework. Finally, we evaluate the model simulated.

1.5 Research Questions

For us to fulfill the aim and specific objectives of this research, the response to the following research questions (RQs) will be taken as a guide:

- **RQ1:** What are the applicable tools for detecting conflicts in app reviews, and how can the tools be optimally applied for feature extraction?
- **RQ2:** How can the conflicts detected in RQ1 be resolved using computational means?
- **RQ3:** How do we validate the conflict detection and resolution instruments in RQ1?

1.6 Research Methodology Overview

The implementation processes explored in this study involve three key phases, which are (1) mining and pre-processing of data sets from three different sources, namely app store[1], google play store[2] and kaggle repository[3] (2) identifying and reviewing conflicts in the reviews; and (3) presenting the requirement engineering benefits of phase (2) to software developers and software stakeholders for improving the app features and app deliverable. These methodology processes are diagrammatically described by Figure 5 in Chapter 3 of this thesis.

1.7 Research Contribution

This study contributes to the growing domain of knowledge on applying natural language computational tools in enhancing the quality of apps that are developed. This research aims to be pragmatic in identifying and resolving conflicts in app reviews. Our study findings will help the administrative controllers of an app to maintain the quality of their product, as well as the app developer (or coders) in the requirement engineering elicitation activity.

1.8 Thesis Structure

This chapter provides an introductory overview of this study. We discussed the problem statement, purpose, objectives, scope, and research questions. The remainder of this thesis is structured as follows. Chapter 2 identified the observed shortcomings of the existing study in this research focus. In Chapter 3, we discussed our techniques, data sets, and implementation procedure. Chapter 4 presents our study findings using suitable inferences, while Chapter 5 discusses our result. In addition, we discuss how we have

[1]https://play.google.com/store/apps/
[2]https://play.google.com/store/apps
[3]https://www.kaggle.com/code/mmsant/ratings-and-reviews-and-android-app-success/

been able to answer our research questions based on our findings. The threats to validity of this work are presented in Chapter 6. Finally, Chapter 7 concludes this thesis by summarizing the entire research process, results, recommended improvements, and direction for future work.

1.9 Definition of Terms

Definition of Terms:

App: is used as an abbreviation to represent a software application.

Corpus: represents collection (or database) of text documents, which can be referenced.

Review: statements representing feedback from users after interacting with the software application.

Reviewer: the individual (or group of individuals) presenting their opinion about a software application in form of feedback in a textual form.

Stakeholder: refers to the parties involved in the administrative governance and implementation process of the software application.

2 Literature Review

This chapter reviews recent works and general knowledge about key concepts related to conflict detection and resolution in app reviews. Considering the growing number of mobile applications, the app development industry has piqued the interest of researchers both within and beyond the software engineering industry. Based on this development, requirements engineering (RE) for software development has received equal research attention. For software engineers, one of the primary channels to improve the quality of their software product is to allow the user-stakeholder to give review about their product [24]. The review can be related to different parts of the software product ranging from complaints, suggestions, pricing, and feature addition [25, 26]. Interestingly, careful analysis of these reviews can result into the development of a new app or improve on the existing one, depending on the analytic consideration of the app owner. The task of app developers and app owner is to identify those reviews that are usable and beneficiary to their software product. Based on this, there is possibility to identify conflicts in app reviews, which could be caused by several factors such as psychological or social reasons. Therefore, we are keen to explore the dynamics of app review analysis, particularly as it relates to the requirement elicitation phase of the app development.

Conflict-detection-based research on app reviews has received a growing interest in the past decade. A study conducted by Crouch *et al.* [27] was among the pioneering research in this field of interest to highlight the significance of identifying conflicts in a text manuscript. Their study relied on a rigorous logical description of concepts used in this field. They did not publish the empirical results of their study, which made it difficult to appraise the impact of their study and contribution to the body of knowledge.

Another research carried out by Finley *et al.* [28] published the first empirical results for conflict detection in textual manuscripts, however they only concentrated on two classes of conflicts, namely; conflict generated by paraphrasing text and conflict caused

by negation. Their study showed an improvement in the work of Crouch, thereby offering other researchers the opportunity to explore more concrete findings in this research field. Further, we will discuss existing studies under the three key headings, namely; Natural Language Processing, Machine Learning and App Review Classification.

2.1 Machine Learning

Machine learning is a branch of artificial intelligence (AI) that trains computers to mimic humans by analyzing and learning from data without direct instruction [29]. Computers learn to imitate intelligent human behaviors and continue improving on its accuracy with the use of data and algorithms [30, 31]. A machine learning system can be descriptive, predictive or prescriptive. Descriptive ML system uses data to provide explanation for an occurrence, predictive system uses data to predict the possible outcome of an occurrence, while prescriptive system uses data to suggest possible actions to take during an event [32, 33].

ML algorithm automates the knowledge process and extracts information on trained data to make projections of undetected data insight [29]. Through statistical methods, ML algorithms are trained to uncover insights from data by classification and predictions. They can also identify unusual occurrences in data and determine structures present. The uncovered insights are used to drive policy or business decisions [31]. ML Algorithms are classified either by their techniques (supervised, unsupervised, or reinforcement learning) or by their family (such as classification, regression, and clustering). According to Shathil *et al.* [29], the popular machine learning techniques algorithms are as follows:

1. Decision Tree: this is the algorithm designed to train data sets into subsets attributes, generate training models' and predict the classes or values of variable destinations.

2. Naïve-Bayes: this is used to characterize belongings of possible and certain class theorems through classification techniques to increase sophisticated class-based

methods.

3. Linear Regression: this measures the value of reliant variables using an independent adaptable technique. It involves relational mappings of variable lines both on independent and dependent regression lines that most times intercept at a slope.

4. Support Vector Machine: this is a binary classifier use to separate data sets that is drawn, and n-dimension point of row data to enhance training margin.

5. Logistical Regression: this algorithm is used to define dependent discretion of separable variable sets. This approach provides a coefficient and probabilities of an estimated transformation.

6. Linear Discriminant Analysis (LDA): this is a linear classifier that densifies data and apply decisional class boundaries of Naïve-Bayes rules. It is also known to generalize data sets into lower dimensions by reducing it space complexity modeled and result in a computational outlay.

Machine learning systems work by collecting and preparing data, training a model, validating the model and interpreting the results. The way the above processes work is determined by the subcategory of the ML system. The system can be supervised, semi-supervised, unsupervised, or reinforced. Supervised ML models are trained with labeled data sets to classify data or predict outcomes accurately. As part of cross-validation to avoid over-fitting or under-fitting, the model continuously adjusts its weights with every data input until the model fits. An example of supervised ML algorithm is one trained with pictures labeled by humans, the pictures would be of different things and a target (e.g., an ant), and the system would learn ways to identify the target within all the pictures. This ML system is the most commonly used, and some of its methods are neural networks, support vector machine (SVM), Naïve-Bayes, random forest, linear regression, logistic regression, etc.

Unsupervised ML does not use a label data, it looks for patterns or trends that people may not see in unlabeled data. This ML system is best for image and pattern recognition, exploratory data analysis, customer segmentation, and cross-selling strategies, because of its ability to uncover similarities or differences in data. Methods of unsupervised ML system include neural networks, probabilistic clustering, k-means clustering, etc. Semi-supervised ML systems is midway between supervised and unsupervised models. This model uses few labeled data classify and extract features from a larger, unlabeled data. The semi-supervised model is used in situations when there are no sufficient labeled data for training a supervised ML algorithm [30, 34, 35]. Lastly, reinforcement ML systems work by trial and error for a reward. The system is a behavioral model similar to supervised ML but not trained with sample data [36].

2.2 Natural Language Processing

Natural language processing is a field of machine learning that enables computers to understand text and spoken words as human do [32, 37]. According to Messaoud *et al.*, it combines computational linguistics, machine learning, and deep learning models with statistics. The synergy between these technologies gives computers the ability to understand, interpret or translate and communicate human language. Because of the complexity of human language, NLP has to do a number of different tasks to serve its purpose. Its tasks range from speech recognition, part-of-speech tagging, co-reference resolution, word sense disambiguation, sentiment analysis, named entity recognition and natural language generation [38]. Asides the general use cases of NLP shown below, it has sector-specific use in finance, e-commerce, healthcare, education, and cybersecurity [39, 40].

NLP methods are useful in extracting informative data sets from texts. NLP is relatively used to automate keywords of certain analyzed features by extraction, ranking and categorization of semantic queries [41]. This method has been used to find collected

words and automate sentiment analysis technique by tackling undersized and low-quality opinions, text and association of every reviewed feature [13, 42]. Data pre-processing is carried out in preparation for feature extractions, using word lemmatization, part-of-speech (POS) tagging, and stop word removal techniques on NLP tool-kits libraries. In addition, NLP tool-kits libraries extracts collective algorithm and WordNet synonym dictionary in forms of a bi-gram associations (common phrases of two words) with informative features. Data processing are staged to analyze the sentiments and pre-define rules of extraction tools [43, 44]. These tools enable parsing identification of analyzed sentiments as either positive or negative [45]. In app review extractions and analysis, most NLP techniques are use on Google App store apps than in iOS App store because it mostly focuses on text and ignores the possibility and availability of metadata which iOS App stores provide [13, 42].

2.3 Mining User Reviews

The popularity of mobile devices and smartphone applications has exponentially grown the app market into an industry worth billions of dollars. With the number of apps, users and downloads constantly increasing, developers and marketers constantly desire to know what users think of their products [12, 8]. The deployment of app reviews allows developers and marketers to earwig on users' opinions and demands [12]. However, the effort and time required to make sense of these ever-increasing reviews is not affordable. So, developers constantly lookout for reliable user review analytic tools to help reduce the time and effort required to gain insights from reviews [36, 46]. Unfortunately, not much of these types of analytic tools are available to help developers and marketers with their needs [12]. This has recently brought attention to the practice of user feedback mining using various machine learning techniques, and a number of academic and industry research have been done in this area, howbeit, mostly exploratory.

The first focus on mobile app reviews were on automatic extraction/mining, to

identify, group or classify useful reviews [12, 39]. Users' satisfaction has been measured by extracting and analyzing app reviews through words and phrases matching with a predefined dictionary [15]. More limited, fewer works have analyzed the connection between app user satisfaction and non-functional requirements like security, performance, privacy, usability, and dependability [15, 45]. A study by Iacob and Harrison [47] adopted the linguistics rule to extract feature requests from app reviews and group the requests using Latent Dirichlet Allocation (LDA) algorithm. In 2019, contrary to the more complex approaches being used, [45] used the traditional Bag-of-Words (BoW) approach to automatically classify reviews and found that BoW's performance is as effective as the more complex models that have been previously tested.

The advancement in review extractions soon evolved into analyzing sentiments attached to the content of the extracted review. There has been varying interest in what researches want to do with reviews, a study by Kunaefi *et al.* [6] mined and analyzed actions and their reasons from reviews. Winbladh *et al.* [48] applied LDA for summarizing app reviews, as well as their sentiment. However, the focus of the summarization was not on the sentiment analysis as opposed to Guzman *et al.* [8] who after extracting apps features from reviews and analyzing the attached sentiments, described user acceptance of features using different granularities. Similarly, a framework that automatically extracts applicable features from user reviews and analyze the associated sentiment was proposed by Luiz *et al.* [2]. Through the application of sentiment analysis, summarization interface, and topic modeling, the framework proves to developers that the accuracy of sentiment rating is better than that of star rating [49].

The AR-Miner computational framework by Chen *et al.* [12] operates an identical model to the work of Luiz *et al.* [2]. However, AR-Miner has four components, and it is not focused on sentiment rating of reviews but the overall usefulness of the reviews. Furthermore, as an improvement of AR-Miner, Hirave *et al.*] [36] created additional feature that uses the Multinomial Naïve-Bayes algorithm to automate the reviews prioritization

process. Earlier in 2013, Fu [14] and his colleagues proposed a system called WisCom. The system analyzes user ratings and text reviews in three distinct levels: micro, meso and macro. It does this by identifying inconsistencies in reviews, scouting for reasons why people dislike an app, and monitors if and how the reviews change over time.

The methodology for mining and summarizing user reviews created by WisCom, help users to make the best choices without reading reviews, and it also help developers to know why people dislike their app [14]. A number of other tools have been created to do specific things with app reviews; among them is *REVSUM* which identifies information spontaneously such as feature request or reported bugs which are relevant to developer [45]. Naïve-Bayes algorithm have also been used on extracted mobile app reviews to identify fraudulent apps in application distribution platforms. The study cleaned the reviews by data pre-processing and used the ML algorithm to determine the polarity of the review content [29].

Literature on mobile app reviews moved from identification, grouping, classification, sentiment analysis to prediction. Araujo *et al.* [9, 50] investigated classification techniques, sentiment analysis, and utility prediction of mobile app reviews using different techniques from traditional BoW to the relatively new Neural Language models (NLM). However, very few have acknowledged the possibilities of conflict arising in the reviews. Aralikatte *et al.* [51, 52] and Islam *et al.* [53] employed some ML approaches to prove that mismatch occurs between reviews and star rating of an app. This means that a user can be satisfied with an app but only give one-star rating because the app lacks a suggested feature. This justifies the work of Luiz *et al.* [2] which proved to developers that the accuracy of sentiment rating is better than that of star rating. This inconsistency in review and rating is a kind of misinformation for the developer. Similarly, conflicts and mismatch exist within text reviews and there is a need to identify and resolve them [19, 54].

2.4 Inferences Observed in Existing Studies

We made some significant discoveries after reviewing some existing works in this research line, and we gleaned some key inferences from their findings, which are presented below:

1. Majority of app-review-based research did not focus on identifying and resolving conflicts. Typically, they concentrated on classifying app reviews into sub-types using machine learning or other classification algorithms. Other research focuses on using regression models to predict app ratings based on historical app ratings.

2. As a tool to aid in the analysis of app reviews, machine learning classifiers such as Maximum entropy (MaxEnt), Naive Bayes, and decision tree were used to classify app reviews into four types based on the review content, which are (i) feature recommendation (or request), which describes the recommendation of functionalities or request to implement key functionalities that is currently missing in an app. (ii) bug reports which describes noticeable issues associated with app's functional capability, such as advertisement intrusion, crash, and poor performance. (iii) user experience which represent users' interactions with the app and its features in certain scenarios. Usually, they are regarded as documentation for the app's functionality and requirements; and (iv) user ratings which contains information that are insignificant to app developers because they usually contain commendation, criticism, distracting critique, etc. Researchers noted that an app review can be grouped into various categories of review.

3. Some studies' findings are not applicable to improving the quality of a software application in terms of requirement engineering. As a result, such a study is not helpful to app owners and developers when it comes to developing a better version of their software product.

4. Some research analyzed app reviews using a pre-developed software application.

By doing so, they only published their findings, meaning that technical procedures were not engaged in their study. Therefore, they are confined to a mono-like result, since their implementation system cannot be configured to their research analysis.

5. Some of the studies on app review focuses on using tools such as VADER (i.e., Valence Aware Dictionary and Sentiment Reasoner), a lexicon and rule-based text analyzer to label a review as positive, neutral, negative, or compound review. This is accomplished by parsing each user review through the VADER python algorithm.

3 Research Approach

This chapter examined the computational tools and implementation approaches undertaken in our study, ranging from mining the data set to achieving results to align with our study goals. In addition, we exposed the relationship between different components of our research study and how we have been able to apply the required instrument for answering our research questions. The sections below describe the data sets and the implementation frameworks that were explored in this study.

3.1 Data set

In order to have a broad research scope, we retrieved our data set from three different sources; (1) we scrapped app reviews from android's google play store[4], as shown in Figure 1. (2) we scrapped app reviews from apple's app store[5]; and (3) we downloaded a pre-mined app reviews from kaggle repository [55].

3.1.1 Data Mining Procedure from Google Play Store

[1] **Step1:** In our python environment (jupyter notebook), we installed and import the required python module to connect to the google API (googleplay scraper).

[2] **Step2:** We fetched the app's unique identification (ID) code from the uniform resource locator (URL) of the target mobile app (e.g. the ID for app named *shot bubble game* is "*com.bubbleshooter.popbubbles.shootbubblesgame*").

[3] **Step3:** Based on (2), the ID of the target app is fed into a lambda pre-built function with hyperparameters (such as language, date range) that can be tuned to suit

[4]https://play.google.com/store/games
[5]https://apps.apple.com/us/genre/ios/id36

our research need. The function was implemented and configured to allow us download the comma separated values (CSV) version of the target app's review.

[4] **Step4:** The download CSV is re-imported into the python environment and stored in a data frame before they are merged with reviews extracted from apple's app store and kaggle environment.

3.1.2 Data Mining Procedure from Apple's App Store

[1] **Step1:** In our python environment (jupyter notebook), we installed and import the required python module to connect to the apple API (apple store scraper).

[2] **Step2:** We fetched the app's name and unique identification (ID) code from the uniform resource locator (URL) of the target mobile app (e.g., the ID for app named *Transporter* is "*id1450874784?mt=12*").

[3] **Step3:** Based on (2), the ID of the target app is fed into a lambda pre-built Appstore function with hyperparameters (such as date range, review count) that can be tuned to suit our research needs. The function was implemented and configured to allow us to download the comma separated values (CSV) version of the target app's review.

[4] **Step4:** The download CSV is re-imported into the python environment and stored in a data frame before they are merged with reviews extracted from google play store and kaggle environment.

Based on our mining results in google play store and apple store, we can make the following illations:

Illations from Data set

(i) A typical app review contains between 6 to 170 words, average rating of 3 (reference to Figure 3), and an average of three different symbols. Reviews with less than (or equal to) three syllables were excluded during data analysis as they can not contribute to the research analysis of this study.

(ii) User expressed reviews based on different factors such as logical, psychological, etc, in relation to their point-of-concerns in the app. In addition, examining a random sample of the reviews showed that some reviews do not express user's satisfaction or dissatisfaction. Some review could be observed to be extracting (or giving information), requesting features, or exposing problem with the app. How to make this information usable depends on the app owners and app developers.

(iii) The mined review count from google's play store is 31,425, consisting of reviews from 18 different applications with date range between January 2012 and November, 2021.

(iv) The mined review count from Apple's app store is 9,629, consisting of reviews from 7 different applications with date range between March 2014 and August 2021.

(v) The downloaded review count from kaggle is 21, 937, with date range between July 2017 and September 2021.

(vi) We primarily considered the review text as analyzed data set, and based on (iii), (iv), and (v), the summation of all reviews examined in this study was computed to be 62,991.

(vii) By examining the vocabulary of random samples of the reviews, we determined that requirement engineering process for improving application deliverable can be optimized by careful analysis of the app reviews.

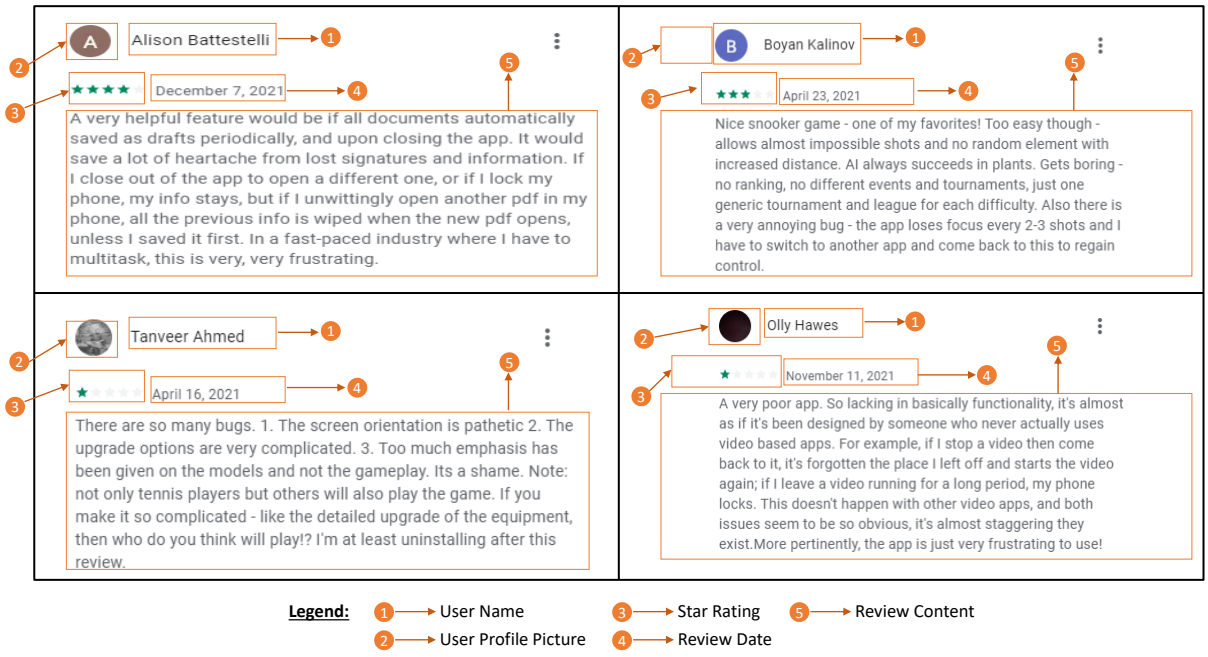

Figure 1. Pictorial Description of the Data sets

The data set downloaded from Kaggle data repository consists of 21,937 distinct reviews from which spans data range July 2017 to September 2021. We excluded columns such as review ID, review author, app version, app classification, and review date from the data set as they could not give a practical contribution to the research purpose. A random sample of three reviews from the data set as well as the app name and review rating are presented in Table 1, while the description of each review type is presented in Tables 2. The review type describes the topical heading where an app review can be categorized.

3.1.3 Categories of Review Attribute Peculiarities

In this study, we identified five key areas of lexical components that are present in our extracted app reviews, which are highlighted as follows.

- **Reasoning Expression:** this includes modifiers, apposition, meronymy, reason-

Table 1. Random Example of Reviews in the Data set

App Name	User Review	Rating
Dropbox	Dropbox is a great app. However, I can only make my files available offline if I upgrade my account. This is unfortunate and I have decided to use an alternative app which allows this in their free version...as I do not always have internet connection. Plzzz dropbox should reconsider including the offline functionality !!!	3
Applock	Hey guys, after update, i am unable to get fingerprint scanner option, directly asking 8 digit password mistakenly forgot al$o get locked this. please help getting soloution ....please	4
Baseball 9	This is $\hat{a}$n excellent game, good graphics and game play. I just find the game too long and gets boring. The needed game f$\tilde{e}$atures are expensive to buy as well. It will be nice if this can be fixed	3

Table 2. Description of each Review Type

Review Type	Description
Functionality	Unexpected app behavior
Network Issue	The app has connection problem with the network, e.g, communication with APIs
Update	App user attributes new problem to app updates
Feature Auditing	One or more feature is causing app misbehavior
Crashing	App crashes or stops working unexpectedly @a after a particular function is enabled
Response Time	The app takes long time to respond and process input
Compatibility	App does not function the same across different devices
Heavy Resources	App consumes high device resources, such as memory, processor, and battery
Ethical Concerns	App invades user privacy which causes for ethical concerns
Additional Cost	Relates to additional required features that must be purchased to access full app functionality
Other	A review that has no usefulness to app engineering

ing on quantities, spatial and temporal reasoning, relative clauses, and elliptic expressions.

- **Syntactic Features:** includes modifiers, negation, active or passive alternation, apposition, and lists.
- **Lexical Features:** includes synonymy, acronym, semantic opposition, and hyperonym.
- **Syntactic Features:** are features such as verbalization, nominalization, paraphrases, and causatives that are observed in the review text.
- **Discourse Expressions:** includes the use of anaphora, coreference, ellipsis, and apposition in the review text.

3.1.4 Review Ratings

By carefully examining the datasets used in this study, we found a disposition in how users rate an app. A rating is the overall user assessment of an app after interacting with its functionality [56]. However, it is interesting to note that users do not always appraise an app exclusively on its ratings, as a random sample of reviews examined revealed a disparity between the review content and the rating offered by such users. A rating is often a number between 1 and 5, inclusive. We retrieved the aggregate of ratings that users assign to their reviews from the three dataset sources investigated in this study. The rating's breakdown is presented in Table 3 and pictorially illustrated in Figure 2.

Table 3. Breakdown of Review Rating

App Source	Rating 1	Rating 2	Rating3	Rating 4	Rating 5	Total
Google Play	6,421	3,618	9,332	7,040	5,014	31,425
App Store	981	1,428	5,213	729	1,278	9,629
Kaggle	7,261	5,628	4,011	3,629	1,408	21,937
Total:	14,663	10,647	18,556	11,398	7,700	62,991

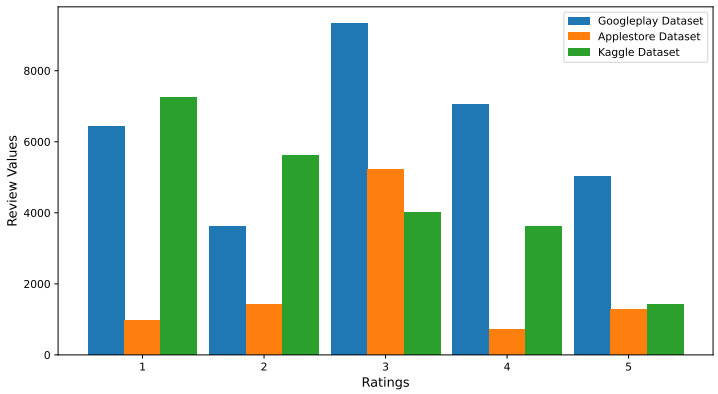

Figure 2. Ratings Versus User Review

3.2 Data Pre-processing and Text Analysis

This study primarily considered the analysis of the review comments, which are textual in presentation, and we assumed that all reviews were written in english language. Further, because user reviews are both formal and informal expression of their opinions regarding an app, the review content must be analyzed, and cleaned to remove unwanted characters. For example, the sample of the dataset considered in this study which is presented in Table 1 contains unwanted characters such as, "...", "*$*", "*!!*", and "@". Since these characters have a detrimental impact on the feature extraction process, we executed the following data cleaning and pre-processing actions, namely; removal of accented characters, word stemming and lemmatization, case conversion, removal of repeated characters and punctuation, expansion of contractions, removal of special characters, stop-word removal, and correction of misspelled words.

By examining these data pre-processing technique, the text is cleaned and transformed to a readable and usable form, allowing for a good feature engineering process, which is a key component of this study. The data processing procedures were all carried out

using the applicable Python modules[6]. The first data analysis that we performed was the removal of reviews with less than or equals to three tokens (syllables) from the datasets. 963, 361, and 692 reviews with less than (or equal to) three syllables were removed from the dataset mined from google play store, apple store, and kaggle respectively. The remaining dataset were cleaned and pre-processed, as described below.

3.2.1 Removal of Accented Characters

Accented characters are text characters that does not comply with the American Standard Code for Information Interchange (ASCII) standard used for text representation [57]. Example of accented characters are $\check{a}, \tilde{a}, \hat{a}$, and $\acute{a}$. We removed the accented characters by applying the *unicode()* function in python, which takes data texts as input and convert them to ASCII characters. For example, consider one of the review texts for the "Baseball 9" app given as *"This is $\hat{a}$n excellent game, good graphics and game play. I just find th$\tilde{e}$ game too long and g$\hat{e}$ts boring"*. After the *unicode()* function is applied to the above text, the accented characters are removed, and the resulting text is given as *"This is an excellent game, good graphics and game play. I just find the game too long and gets boring"*.

3.2.2 Word Stemming and Lemmatization

Our text processing objective is to extract plain text free of source-specific constructs that are irrelevant to our task. In natural language text processing, word stemming refers to the reduction and normalization of words to their root form by removing their morphological affixes [58, 59]. For instance, the word "branches", "branching", and "branched" can be reduced to the root word "branch", since they convey the same concept of separating in different path. Word stemming is meant to be a crude and fast procedure performed using very simple style rules of search and replace. Stemming can also be done by removing

[6]https://www.python.org/

suffixes such as "ing", "ies", and "ed," from words which then reduce them to their root words. We applied the *"SnowballStemmer()"* function, an NLTK word stemmer module in python.

Similarly, lemmatization is another technique which we applied to reduce a word to its normalized form [58]. In lemmatization, word transformation uses a dictionary to map different representation of word to its root form. For example, in lemmatization, "was", "is", and "were" is transformed to "be". We applied the *"WordNetLemmatizer()"* NLTK function for the lemmatization operation because of its capability to decode the part-of-speech (such as noun) for each word its lemmatizing. Both word stemming and lemmatization results in a meaningful word as output, but the key difference between stemming and lemmatization is that stemming technique does not require a word dictionary to function, whereas lemmatization does [59].

3.2.3 Case Conversion

We transformed all the text reviews to lowercase in order to avoid discrepancy and eliminate the processing error in feature extraction between uppercase and lowercase words. In machine operation, the lowercase and uppercase are interpreted as two different, hence, we converted all sentences to lowercase. For this purpose, we applied the *"lower()"* inbuilt python function for this procedure. An example of review phrase "the App drAins MY batteRy" is transformed to "the app drains my battery" after we applied the *"lower()"* python function.

3.2.4 Removal of Repeated Character and Punctuation

There are instances where symbols and punctuation in phrases are repeated more than required. This is regarded as an informal representation of words (or phrases) expressed by a user. The repeated characters and punctuation in a sentence reduce the linguistic meaning of a sentence. For example, words like "Cheeeeerrrrss !!!!!!", "grrrreat!",

and "Reallllllllyyyy" are expression of a user stressing their opinion about an app. We developed a regular expression function to remove the repeated character and punctuation. In addition, we applied the *Pattern alpha* function in our regular expression function to limit the repeated and consecutive characters to two, in order to preserve the meaning of the root word before spellcheck function is applied for word sanity.

3.2.5 Expansion of Word Contractions

In order to remove stop words from our reviews, it is expedient for us to deal with word contractions. Contractions are shorthand representation for words such as *"would not"* and *"they have"*, which is written in their contracted form as *"wouldn't"* and *"they've"* respectively. To expand on the contractions, we developed an expansion contraction function. First, we built a contraction word corpus extracted from Wikipedia [60]. Next, we tokenize each text into tokens and check if the provided token matches the key (in the word corpus), then, we replaced word with the key's value found in the corpus. As an illustration, a user review with the content *"I can't open the app simultaneously with other apps"* is transformed to *"I cannot open the app simultaneously with other apps"* after we applied the expansion contraction function.

3.2.6 Removal of Special Characters

Special characters are used to express and establish grammatical structures in sentence [58, 59]. In this study, we formulated a lambda function with regular expression that removes special characters such as "{", "(", "!", and "#" from user reviews. We removed the special character to prevent skewed result when tokenization procedure is performed later on the text during feature engineering.

3.2.7 Removal of Stopwords

In language structure, stopwords refers to the most frequent words used in sentences (such as conjunctions, prepositions, pronouns, etc) which offer little information and meaning to a sentence [59]. Examples of stop words includes are "a", “the”, “an”, “so”, and “what”. Stopwords are eliminated from the review text so that more emphasis may be placed on the words that define the review’s meaning. In this study, we explored the NLTK’s corpus spaCy[7] in python to remove stopwords from user review texts. SpaCy has a list of stopwords that can be imported from the spacy.lang.en.stopwords class. For example, a user review of an app with the content *"The red button located at the top right of the app by the home screen is distracting and should be removed"* translates to *"red button located top right app welcome screen distracting should removed"* after we applied the stopword removal function.

3.2.8 Correction of Misspelled Words

Our text analysis takes into consideration the typographical mistakes of users when submitting their reviews. Hence, we implemented a spelling correction feature capable of correcting wrong words that are spelled wrongly by users. There are several techniques that can be used for this auto-correction, but we explored python’s NLTK *"speller"* function which relates each word to the English language word dictionary and corrects them accordingly in case they are misspelled by a user.

It is noteworthy that the data processing techniques described above are consistent and applied to all user review texts considered in this study before feature extraction is stage.

[7] https://spacy.io/

3.3 Overview of App Reviews

A user's app review is analogous to a written representation of their opinions which is used to express the satisfaction level derived from using an app [61]. Based on this, the usefulness and ease-of-use of an app can be easily decoded. In general, user reviews are a type of value-driven process which is advantageous to the administration of app functionality by providing adequate feedback for criticism and improvement to app developer [62]. Critic feedback usually exposes where improvements should be made on an app in order to present a more usable and acceptable app version to the users [63].

In essence, the information value of reviews is determined by users' points of view [52, 40]. Positive reviews benefit app developers, while negative reviews benefit prospective users of an app as they will have a foreknowledge of the app functionality. Although, negative app reviews could be unpleasant to app developers, however, such reviews expose areas of an app that requires more functional attention. Over the years, the analysis of app reviews has been a major strength for industries in improving the user satisfaction experience of their apps [62, 61]. Research in these domains revealed the significant impact on review-based analysis on sales and that users prefer to examine existing reviews to guide their orientation about an app [61].

Furthermore, they claim that users submit reviews of varying lengths containing unconventional phrases, abbreviations, and incorrect spelling. They also noted that reviews usually cover a wide range of issues within the domain and the object under evaluation [62, 61]. In addition, studies have shown that most of the app reviews expressed by users are anecdotal and not holistic evaluation of the app. Hence, we took these factors into consideration in our implementation analysis.

3.4 Forms of Conflict Detection Features

This section outlines the major patterns of conflict features that is detected from our mined app reviews. They are outlined below;

3.4.1 Polarity Attributes

The existence of polarity features in user review is usually a positive indicator that conflicts exist as long as topic of interest strongly aligns [64, 65]. The polarity feature in app reviews describes the existence (or omission) of linguistic cues of negative polarity parameters. For instance, in English vocabulary, words are regarded as negative if they contain an explicit linguistic negation marker (e.g., negation word - not, restricted prepositions, and monotone determiner - few). A polarity gap in reviews can exist if a review R_i is negated while another review R_j is not negated. This contrast is validated by ensuring that the terms are not exact opposites and that there is no other connotation indicating that they do not refer to the same topic [57].

3.4.2 Structural Attributes

The structural features are used to evaluate whether the syntactic structure of a review and its hypothesis are sufficient to produce conflicting claims [66]. For instance, we examined the object and subjects of a review text, which assisted us to detect and ascertain a conflict whenever the object overlaps with the subject.

3.4.3 Factive Attributes

The scope in which a verb phrase is incorporated into a review could lead to conflict in interpretation and meaning. Factive patterns observed in reviews are primarily influenced by negation words [66, 67]. For example, *"the app failed to allow pdf file format"* conflicts with *"the app allowed pdf file format"*, while *"the app did not fail to allow pdf*

file format" does not conflict with *"the app allowed pdf file format"*. Therefore, based on the hypothesis analyzed for each review, we constructed features that are related to their factive class, which are considered in our implementation analysis presented in Chapter 4

3.4.4 Antonyms

Antonyms refer to word which offers opposite interpretation of another word. Antonyms are good indicator for detecting conflicting ideas. Our list of antonyms and contrasting terms examined in this study are extracted from the oppositional verbs and semantic knowledge base network in VerbOcean[8] and WordNeT[9]. We searched for aligned pairs of terms in the list, as well as frequent antonym prefixes (such as "*un*" and "*anti*"). We used the topic context's polarity to assess if the antonyms are strong enough to form conflicts in the app review.

3.4.5 Relational Attributes

Relational attribute describes the interaction between different element of a review [68]. Mining our datasets is an information retrieval task where the notion captured the extracting relationship between review components. Although relational elements give precise information about a review idea, we discovered they are challenging to expand for broad analysis. These characteristics allow us to simulate how opinions stated in review text interact in a phrase with respect to other reviews. Using relational attributes enabled us to evaluate numerous user's opinions simultaneously. Illustrations of the conflict detection features observed in the app reviews are presented in Table 4.

[8]https://demo.patrickpantel.com/demos/verbocean/
[9]https://wordnet.princeton.edu/

Table 4. Example of Conflicts in App Reviews

	Conflict Case	**Review**$_i$	**Review**$_j$
1	Action Polarity	Add a button that can allow us print a particular screen-view on the app	The addition of another button is not necessary for the current functionalities
2	Effect Polarity	Disable screenshot, but enable screen-view printing to allow for transparency	Ability to print every screen-view could increase response time of the app due to the required system's resources
3	Temporal Case	Implement a daily auto-backup feature to guarantee data reliability to user	Auto backup feature should not be added as more storage space will be occupied, which can be dissatisfying to user with a low storage device
4	Conditional	The three-stage login authentication can frustrate to users when logging-in into the app	Three-stage login authentication should be considered in order to boost app's security
5	Sub-regular	Add color palette to the chat functionality to allow for custom chat aesthetics	Use a monochromatic color as color palette causes app to freeze for about two minutes
6	Quantitative	Reduce the required scores to 20 instead of 50 before the next level will be unlocked	A high score above 50 for unlocking next level is required to test for user intelligence
7	Cumulative Effect	App log files should be cleared daily automatically	The app log files should be merged with previous version to help developers easily identify origin of issues in the app

3.5 Establishing that Conflicts Exists in App Reviews

App reviews contain different types of expression, originating from different app users. Empirically, the probability of review R_x conflicting with another (set of) review R_y within the same domain of interest (in our case, mobile applications) is estimated to be about 0.5 according to the proposition of Tejas *et al.* [36]. The different case and degrees of conflicts as presented in Table 4 is asserts that conflict exists in user app reviews.

3.5.1 Sources of Conflict Detection

Figure 3 demonstrates the critical importance of knowledge for the conflict detection task. Approximately half of the errors (48%) are caused by ambiguous reasons, which we discovered to be one of the most persistent restrictions to discovering real conflicts. A significant portion of conflicts is attributed to meronyms (about 33%), as well as use of synonyms (about 14%), implying that lexical resources with broad range than WordNet would significantly improve model performance. Interestingly, only three percent (3%) of conflicts resulted from extraction errors.

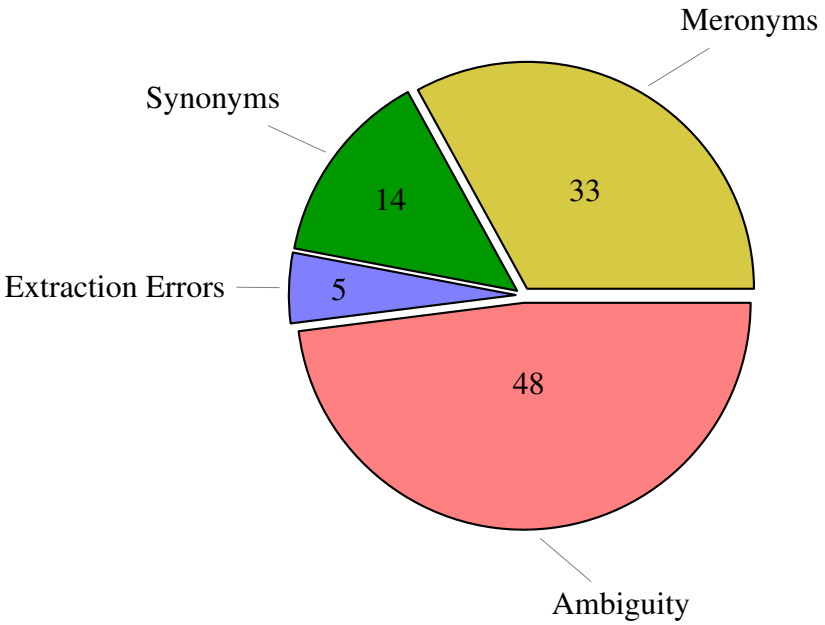

Figure 3. Distribution of Conflict Source [1]

3.6 Detecting and Resolving Conflicts in App Reviews

In order to achieve the goal of this study, and to establish that conflicts exist in app reviews, we implemented a rule-based computational framework for detecting conflicts in app reviews. The conflict detection algorithm is presented by Algorithm 1, and the

implementation details are based on problem formulation and are presented as follows;

3.7 Problem Formulation

In order for our solution to be well positioned in line of this research, we first elaborated on the dynamics and composition of a single app review as it relates to this study. Typically, we expatiate an app review to contain one (or more) *object* with an *action* element, where each *action* results into one (or more) *effects*. In this study, we agreed that the *object*, *action*, and *effect* are the three key components that is present in each of the app review in order to have a uniform frame of reference. Therefore, before we define conflicts computationally to aggregate the function of the aforementioned review components, we define the *object*, *action*, and *effect* component for any given review R_x present in app review corpus. We regard these components as the semantic segmentation of the app review as shown in Figure 4. It is worth mentioning that in this study, the features extracted from an app review refers to the primary connotation of a user's review.

We define the **Object** ($\bar{O}_i$) as the action controller that exists in a particular review. The object is regarded as the structural actor that manages different functions of an app. A review object can have a sub-regular $\breve{t}_i$ semantic representation. For example, an object could be a *button* on an app or the *view* showing the app's functionality.

Similarly, the **Action** ($\bar{A}_i$) refers to the implied effect of a review, which could be in a direct form signifying an imperative expression or indirectly form signifying a declarative expression. For instance, an imperative expression could be "*reduce the transition duration between the welcome screen and the introduction screen*", while a declarative expression could be "*the app features responds better in online mode than offline mode*". Furthermore, the action is used to identify the quantity $\breve{q}_i$ of an object present ($\bar{O}_i$) in a review (e.g., small button). The quantity $\breve{q}_i$ can be expressed as an adverbial quantifier $\breve{d}_i$ (e.g. small) or numerical quantifier $\breve{g}_i$ (e.g., 3 inches). Additionally, an action ($\bar{A}_i$) can be conditional (i.e., expressed as minimum of one conditional clause

$\breve{u}_i$) or temporal (i.e., expressed as a temporal clause $\breve{v}_i$).

The **Effect** ($\bar{E}_i$) represents the resultant impact of an action ($\bar{A}_i$) in a review. For example, in an app review, the effect ($\bar{E}_i$) could be "*swiping up causes the app to freeze for few seconds*". In addition, an effect ($\bar{e}_i$) has the ability to produce a chain-like ripples of resultant effect, which could be categorized as the primary or secondary effect.

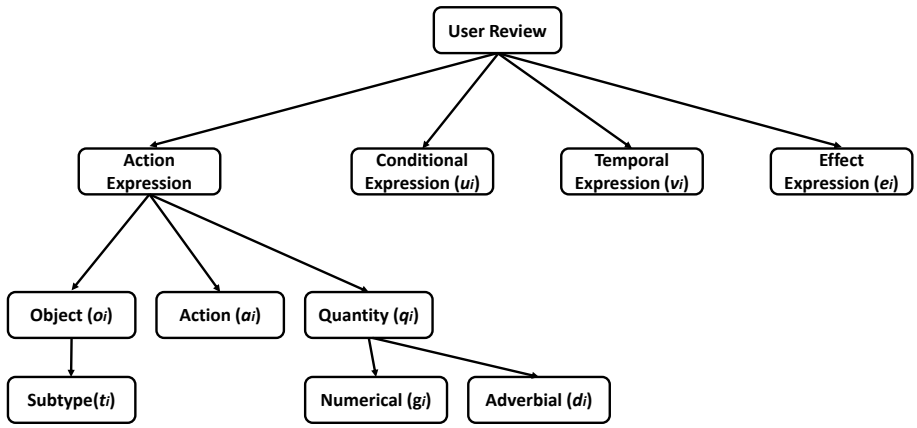

Figure 4. Semantic Breakdown of App Review

Based on the above definition of review text's components (in terms of Object $\bar{O}_i$, Action $\bar{A}_i$, and Effect $\bar{E}_i$), we can decompose and represent a review R_i as a tuple of semantic tokens as $Review_i : < \breve{t}_i^{\,k}, \bar{O}_i^{\,k}, \bar{A}_i^{\,k}, \breve{q}_i^{\,k}, \breve{u}_i^{\,k}, \breve{v}_i^{\,k}, \bar{E}_i^{\,k} >$, as shown in Figure 4. From the review's tuple of semantic token, k represents element's index in the tuple. A particular app review could have more than one object $\bar{O}_i$, action $\bar{A}_i$ and effect than $\bar{E}_i$. In this case, the review is decomposed and treated as multiple sentences, with functional implementation which is described with example in Chapter 4 of this thesis. Similarly, the quantity $\breve{q}_i^{\,k}$ can be a numerical quantifier $\breve{g}_i^{\,k}$, an adverbial quantifier $\breve{d}_i^{\,k}$, or both. It should be noted that an action $\bar{A}_i$ and its corresponding effect $\bar{E}_i$ on a review can be interpreted to be positively or negatively polarized in relation to the associated object $\bar{O}_i$. For instance, considering case 1 in Table 4, the action component in review 1 is negatively polarized with regards to the button object, whereas the effect $\bar{E}_i$ present in

Algorithm 1 : Conflict Detection Algorithm - **detectConflict(G,R_j)**

Input1: collection of review corpus
Input2: individual user's review R_j
Output: conflictSignifier, conflictType
H: the hashmap and corresponding token of individual review in G;
conflictSignifier ← false;
R'_j ← preprocess(R_j);
$C_{R'_j}$ ← fetchSemanticClauses(R'_j);
C_{Rj} ← fetchSemanticTokens($C_{R'_j}$);
for *all reviews* R_i *in G* **do**
  C_{Ri} ← H.fetchValue(R_i);
  CO_i ← collection of objects from (C_{Ri});
  CO_j ← collection of objects from (C_{Rj});
  if $CO_i \cap CO_j \neq \emptyset$ **then**
    ### check meta-data of review context
    $subj_i$ ← collection of subjects from review (R_i);
    $subj_j$ ← collection of subjects from review (R_j);
    if $compatible(subj_i, subj_j) == true$ **then**
      comp ← $CO_i \cap CO_j$;
      for *all object o in comp(*$subj_i, subj_j$*)* **do**
        conflictSignifier ← false ;
        pol_i ← allocatePolarity(C_{Ri}) ;
        pol_j ← allocatePolarity(C_{Rj}) ;
        if $pol_i \neq pol_j$ **then**
          conflictSignifier ← true;
          detectPrevalentConflict(C_{Ri}, C_{Rj}) ;
        else
          detectNominalConflict(C_{Ri}, C_{Rj}) ;
        end if
      end for
    end if
  end if
end for

review 2 contains positive polarity in relation to the button object.

3.8 Computational Definition of Conflict in App Review

In this section, we computationally define conflict between two reviews R_i and R_j. Similarly, the definition extends to defining conflicts between any number of n reviews. Table 4 presents different forms of conflict that could be identified in user reviews. We present the condition as well as the scenarios that must be satisfied to assert that conflict exists between any sample of reviews R_i and R_j. The condition and at least one of the scenarios must be satisfied to identify conflict in app reviews. In essence, the condition states that conflict exists for any given pair of reviews R_i and R_j if and only if there are comparable (or similar) objects between them (i.e $\breve{0}_i^{\,k} \approx \breve{0}_j^{\,k}$). Different scenarios for asserting conflicts between any two reviews R_i and R_j are presented below:

(i) The *action* ($\bar{A}_i$) between reviews $R_{i,j}$ contains diametrical polarity, i.e. actions $\breve{a}_i{}^p$ and $\breve{a}_j{}^q$ between reviews R_i^p and R_j^q contains a reverse polarity (e.g case 1 in Table 4).

(ii) The *effect* ($\bar{E}_i$) between reviews $R_{i,j}$ contains diametrical polarity, i.e. effects $\breve{e}_i{}^p$ and $\breve{e}_j{}^q$ between reviews R_i^p and R_j^q contains a reverse polarity (e.g case 2 in Table 4).

(iii) The *action* ($\bar{A}_i$) and *effect* ($\bar{E}_i$) relationship between reviews $R_{i,j}$ contains diametrical polarity, i.e. $\breve{a}_i{}^p$ and $\breve{e}_j{}^q$ or $\breve{e}_i{}^p$ and $\breve{a}_j{}^q$ between reviews R_i^p and R_j^q contains a reverse polarity (e.g case 4 in Table 4).

(iv) Reviews R_i^p and R_j^q have similar polarity but with a quantitative difference (e.g case 6 in Table 4).

(v) The cumulative effect(s) $\breve{e}_i{}^p$ and $\breve{e}_j{}^q$ of review pair actions $\breve{a}_i{}^p$ and $\breve{a}_j{}^q$ surpasses a defined threshold, although individual action $\breve{e}_i^p$ and $\breve{e}_i^q$ in the review pair does not

exceed the defined threshold (e.g case 7 in Table 4).

The scenarios described above illustrates an instance of quantitative and direct conflict between review pairs R_i^p and R_j^q. Furthermore, conflicts in reviews can be inferred too as temporal, conditional or sub-regular. An example of temporal conflict is case 3 in Table 4), where the reviews address the back-up features between the current and future app users. We illustrate a conditional type of conflict by considering scenario (iii) described above as it applies to case 4 in Table 4) contains diametrical polarity, even though condition exists between the review pair. Therefore, we infer this conflict to be conditional. Case 5 in Table 4 describes a sub-regular conflict, where conflict occurs for only one feature (i.e., the color scheme of the app). By analyzing the reviews presented in Table 4, it is worthy to note that conflict could occur due to contextual (as in case 7 of Table 4), and temporal (case 3 of Table 4) condition. Based on these descriptions, we have been able to provide a taxonomic description of different types and conditions for conflicts in app reviews.

3.9 Overview of Implementation Process

The summary of the implementation process involved in this study for answering our research questions is diagrammatically represented by Figure 5.

3.10 Conclusion

This chapter provides the methodology details and their application for answering the research questions. Further in Chapter 4, we will discuss implementation strategy and results obtained.

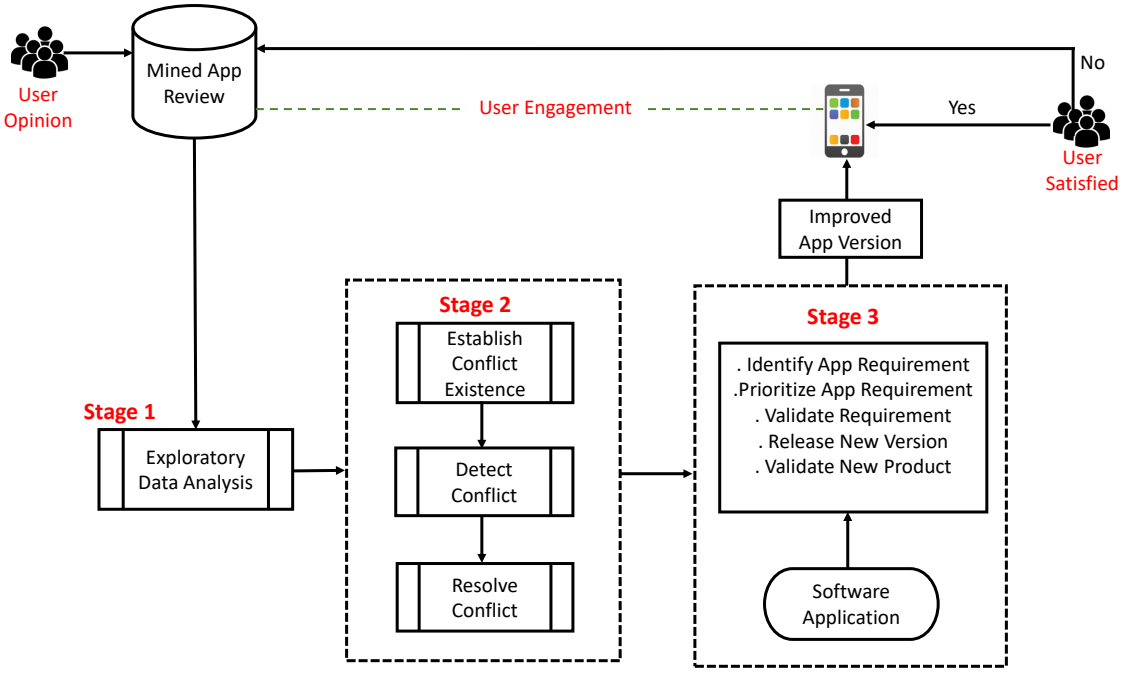

Figure 5. Methodology Pipeline for Analyzing App Reviews

4 Result Analysis

The focus of this study is to establish the notion that conflict exists between two or more user app reviews. In Section 3.5 of Chapter 3, we have established that conflict exists by first defining the components (Object $\bar{O}_i$, Action $\bar{A}_i$, and Effect $\bar{E}_i$) of a user review. Further, by using the rule-based conflict detection algorithm (as per Algorithm 1), we have been able to detect conflict in app reviews and classify the conflict to their respective types as defined in Table 4. In this chapter, we present our study findings and how we have been able to unpack the solution that answers the research questions highlighted in Section 1.5 of Chapter 1.

4.1 Data Pre-processing Result

Relating to the data pre-processing highlighted in Section 3.2 of Chapter 3, Table 5 presents the number of reviews transformed (or affected) by the data pre-processing analysis after the corresponding methodology and tools (described in their definitions) have been applied. The resulting dataset is explored and analyzed for our research experiment.

The rule-based conflict detection algorithm (Algorithm 1 examined in this study uses linguistic rules to decode the semantic representation of each app review. Empirically, the rules are predicated on the training and learning culture of the detection algorithm in order to learn the semantic rules guided by linguistic assertions which are associated with app reviews. The linguistic assertions used to train the conflict detection algorithm includes part-of-speech tagging, grammatical connections, and sentence structure (evidenced by the work of Klein *et al.* [69]).

To detect the conflict, a user review (R_x) is compared with N sets of review sample. Then, we parsed review (R_x) through the Stanford core NLP framework [70] to identify the dependency representation of such review. Next, possible conflicts between review

Table 5. Data Pre-processing Analysis Result

Method	**Google Play**	**App Store**	**Kaggle**
Removal of three-syllabic reviews	933	361	692
Removal of Accented Characters	6,594	294	414
Word Stemming	19,301	6,042	12,324
Lemmatization	17,627	4,728	13,728
Case Conversion	31,422	9,626	21,937
Removal of Repeated Character	1, 473	875	1,037
Expansion of Word Contractions	3, 792	994	1,439
Removal of Special Character	6,239	962	2,746
Removal of Stopwords	29,374	7,934	15,622
Correction of Mispelled Word	4,295	886	1,739

(R_x) and other reviews in the dataset sample is detected by considering the semantic rules and Algorithm 1. There are four key phases to describe the functionality of Algorithm 1, which are described below.

4.2 Phase 1 - Extraction of Semantic Clause

This is the first phase that we explored to detect conflict in app reviews. In this phase, the app review is segmented into four semantic representations. We opt to explore our own semantic segmentation components (action, effect, temporal, and conditional) in place of the pre-defined NLP clause segmentation tools. We developed the extraction rules for the semantic clause by using the dependency relationships between reviews which was learned from the training dataset (70% of the dataset) and linguistic representation of English language vocabulary-base [71].

Action Clause (a_g): As described in Section 3.7 in Chapter 3, the action clause $\bar{a}_g$ is an important part of a review. It represents the active intention of the app reviewer expressed in the review statement. In general, the action clause is identified by verb(s) as well as objects and their tokens. The token extraction method of the action clause is

described below in phase 2 of the conflict detection framework.

Effect Clause (e_g): The effect clause reflects the purpose of an action in a review text. In a presumptions review sentence, the action clauses are succeeded by an effect clause. In this case, a causative preposition (such as: *on account of, because of, so, and due to*) is used to represent an effect in a review text. Through our analysis of the review samples used as training dataset, we identified phrasal verbs that signifies an intent or purpose (e.g., help in, lead to). Therefore, this guided us in developing lexicon of effect indicators gleaned from the English language vocabulary-base [71] and training dataset. Further, to extract the effect clause in the review text, we created set of rules capable of filtering false positives from the review statement. For instance, we filter the word "*to*" from expressions such as: *have to, seem to, according to*, especially when it does not indicate or represents an effect.

Temporal Clause (t_g): A temporal clause reflects the temporal condition of an action which are found in prepositional phrases. Examples of temporal clause indicators are as soon as, whenever, before, and after. In this study, we formulated lexicon of temporal clauses by combining regular expression from SUTime [72] (a framework for identify and annotating statements with temporal expressions) and English language vocabulary-base [71].

Conditional Clause (c_g): The condition clause is used to stunt the action clause under certain conditions. Identifying conditional clause is usually predicated by subordinate expression that begins with verbal expressions such as make sure, or prepositions such as after, before, begin, and when.

4.3 Phase 2 - Extracting Semantic Token from Action Clause

After the first phase, the next phase explored for detecting conflicts is by tokenizing the action clause/expression by segmenting the action clause into object, action, quantity and subject tokens.

Object Clause (o_g): Identifying the objects is a major step to determine the conceptual overlap in a review statement which signifies a pre-condition for conflict existence between two or more reviews. In a review statement, object can be identified as noun phrase or nouns. To identify nouns in a review statement, we identified three major challenges, which are:

[1] Establishing the difference between noun phrases and objects (that is, disregarding non-object noun phrases).

[2] Retaining object order/ranking in reviews. For instance, the object in a review R_x could be a subtype in another review R_y f(or example, a review could generalize about colors while a conflicting review could be specific about monochromatic color); and

[3] Identifying compound objects because most objects could be compound words. To identify the conceptual overlap, we considered both simple and compound objects. For example, we mapped *"flashy colors"* as <sharp, colors, sharp colors>, which indicates that flashy colors or easily identifiable colors should not be considered for a certain component in the app's view.

In Chapter 5, we discuss how these challenges were addressed to satisfy with our research goals.

4.4 Phase 3 - Action and Effect Polarity Configuration

We used the polarity to examine if objects are criticized or not. We allocated polarity by developing verb lexicons (from training dataset) assisted with the verb synset framework from the WordNet language vocabulary module [73]. Table 6 shows the breakdown of initial positive and negative verb lists identified from the three dataset sources, as well as their values when the WordNet is applied.

Table 6. Breakdown of Positive and Negative Verb List

	Cases	Google Play	App Store	Kaggle
Without WordNet	Positive	3,692	841	1,749
	Negative	6,312	659	1,221
With WordNet	Positive	15,768	4,203	6,935
	Negative	18,486	749	12,835

Progressively, relating to the breakdown of postive and negative verb list (as presented in Table 6), we labeled every action identified in the test set as either positive or negative depending on their categorization (as positive or negative) on the verb list. But if an identified action is missing in the verb corpus, then, polarity is labeled as *null*, implying the re-allocation of polarity to an associated effect clause e_g.

We noted that an effect clause by default has positive polarity, whereas we identified two cases for determining negative polarity. First, the tuple <noun phrase (NP), verb phrase (VP)>, where the noun phrase represents different features or objects identified by the reviewer, and the verb phrase represents the verb responsible for the noun phrase. We represent the tuple <NP, VP> as negative markers. In addition, negative markers are also derived from lexicons formulated from the training dataset and MetaMap framework.

Secondly, we identified negative effect clause through the presence of negative adverbial/adjective phrases (such as not good). Furthermore, similar to the approach we used to allocate polarity to actions (described above), we constructed verb/adjective lexicon from the training dataset. We extended the lexicon by utilizing the WordNet synsets framework. We resolve that if one of the two conditions satisfies an effect clause, then we assert that it is negatively polarized. It is worth noting that, despite the rich lexicon-base (comprising positive and negative terms), their usage could possibly result in a skewed output because, empirically, we are also concerned with relevance of domain-specific application of the terms. For instance, verb phrases such as *lead to, and cause* is assigned a neutral and negative polarity for traditional configuration and training dataset respectively.

4.5 Phase 4 - Detecting Conflicts in Reviews

After polarity has been assigned to semantic tokens, next, the token sets are mapped to the probable conflict case (highlighted in Table 4). As shown in Table 7, for each instance, a set of rules is established. These rules are implemented after the semantic tokens is analyzed for identifying the conceptual overlap and polarity has been assigned. Executing the rule defined in Table 7 follows this temporal order.

First, we check if the polarity of two reviews is opposite (using rules 1-4). If this condition is satisfied, then they are identified as direct conflict. *Secondly*, after identifying the reviews as direct conflict, additional rules are applied to test if these conflicts could be simplified. Therefore, the rules for temporal, conditional and sub-regular conflicts are executed simultaneously. A conflict identified by these rules can possibly have more than one conflict type. For example, a conflict can both be sub-regular and conditional. *Thirdly*, if the polarity of the two reviews is identical (that is, no valid condition in rules 1-4), then, the overlapping object's quantitative tokens are examined to determine quantitative conflict. In this case, quantitative conflict occurs when two reviews contain identical polarity about the common object of reference but are different in quantity.

The quantity difference could be influenced by the adverbial quantifiers (such as, "more" and "many") with counter-polarity (for example, a review could suggest adding more features to the app, while the other suggests removing some features to avoid a busy app interface). In such cases, rule 14 is applied for resolving the conflict. Generally, there can be numerical conflict between two reviews with similar polarity based on these two conditions; (i) The tokens (quantitative) of the objects are numerical with exact unit of measurement. (ii) Quantitative tokens are not identical (meaning they have dissimilar or different range of reference as per rule 15). We would like to note that this study did not capture conflict cases as a result of numerical quantitative tokens with different dimensions.

The above mentioned are the four phases required to identify and detect conflicts in review for any two reviews R_x and R_y. We also noted that reviews have varying sentence length.

Table 7. Conflict Detection Rules Between Two App Reviews

Cases	Rules	Conflict Type
Case 1-4 (Similar Object)	$a_g \neq a_h$	Direct Conflict with Opposite Polarity
	$a_h = null, a_g \neq e_h$	
	$a_g = null, e_g \neq a_h$	
	$a_g = a_h = null, e_g \neq e_h$	
Case 5-7 (Direct Conflict)	$c_g \neq null, c_h \neq null$ c_g and c_h are not mutually exclusive	Conditional Conflict
	$c_g = null, c_h \neq null$	
	$c_g \neq null, c_h = null$	
Case 8-10 (Direct Conflict)	$t_g \neq null, t_h \neq null$ t_g and t_h are not mutually exclusive	Temporal Conflict
	$t_g = null, t_h \neq null$	
	$t_g \neq null, t_h = null$	
Case 11-13 (Direct Conflict)	$s_g \neq null, s_h = null$	Sub-regular Conflict
	$s_g = null, s_h \neq null$	
	$s_g \neq null, s_h \neq null, s_g \neq s_h$	
Case 14 (Similar Polarity)	$f_g \neq f_h, s_g = null, s_h = null$	Quantitative Conflict
Case 15 (Similar Polarity)	$unit(n_g) = unit(n_h), n_g \neq n_h$	Quantitative Conflict

4.6 Evaluation of Conflict Analysis in App Reviews

This section explains the evaluation and performance mechanism of different sections of our framework that was applied in the study. In addition, we present the implication of the context awareness of this framework in Section 5.5.

4.6.1 Annotation of Ground Truth

For experimental purpose, we used 70% and 30% of the dataset for training and testing respectively. Empirically, from the training set, we created semantic segmentation rules

as well as conflict detection techniques and we assessed their performance on the test set. There are about N^2 candidate pairs of reviews on the training and test set (where N is the number of training and test set). Through evaluation, conflict annotated with cases 1-6 from Table 4 would arise if the pair of reviews share at least one common object (or topic). Similarly, conflict associated with case 7 from Table 4 would arise if there is no common object/topic. Therefore, in order to efficiently annotate ground truth, we select review pairs that with dissimilar object as presented in Table 8.

Table 8. Statistics of Review Analysis

	Google Play	**App Store**	**Kaggle**
Number of Gold Labels	1,382	829	1,485
Percentage of Gold Labels	37%	23%	40%
Number of Conflict (Train and Test Set)			
Simplified Conflict	22,853	19,251	15,374
Direct Conflict	7,382	3,058	4,911
Numerical Conflict	1,216	2,372	3,959
Fleiss $Kappa$ Value			
Refined Conflict	0.82	0.79	0.73
Direct Conflict	0.79	0.84	0.77
Numerical Conflict	0.87	0.91	0.83

To label our ground truth, we worked with ten final year computer science students to assist us in extracting objects from the 20% of the app reviews, by using the classification rubrics to categorize the objects as positive, neutral, or negative (presented in Table 8). From Table 4, conflict arises for cases 1-5 if polarity of the review pairs in relation to object (O_i and O_j) is opposite. Similarly, quantitative conflicts arise if there exists a minimum of one similar object with exact polarity with different quantitative tokens. In Table 9, we present the breakdown of the number of conflicting features identified in our study, by referencing Table 4. We ensure that each of the review pair has opposite polarity and contains at least one similar object.

From Table 6, the simplified class refers to quantitative, temporal, conditional, and sub-regular conflict. We identified total of 7,511 direct conflicts, 2,196 numerical con-

Table 9. Conflict Feature Breakdown for Train and Test Set

Features	Train Set			Test Set		
	Google Play	App Store	Kaggle	Google Play	App Store	Kaggle
Action Polarity	12,593	2,920	4,954	2,487	384	1,940
Effect Polarity	9,322	1,291	3,584	3,902	729	2,403
Temporal Case	7,202	829	2,986	2,291	231	746
Sub-regular	3,284	769	1,202	982	693	943
Quantitative	3,724	609	293	1,033	648	632
Conditional	2,464	583	649	1,363	399	793
Cumulative	7,082	788	339	3,202	539	231

flicts, and 25,567 simplified conflicts across the three sources of dataset. The breakdown is presented in Table 8. From the result obtained from human annotators, we observed that 1,382 reviews, 829 reviews, and 1,485 reviews obtained a golden label (that is consensus label from human annotators) respectively from google play, app store, and kaggle dataset. We would also like to report that only 15% of the datasets was annotated because of the large amount of data. We used this as random sampling of the dataset. We estimated this agreement by applying the Fleiss Kappa (k) estimator function, which is a function used to compute the degree of agreement during categorization of objects[10]. We applied the python "*inter rater (irr)*" and "*fleiss kappa*" function. We scaled k to be between 0 and 1, where a high value of k signifies high correlation between the human annotators. The overall Kappa value for the reviews annotated is presented in Table 8. We computed a minimum $kappa$ value to be 0.76 and the maximum value of 0.91, which signifies a considerable agreement between the three annotators.

4.7 Performance of Conflict Detection Rule-Based Algorithm

As presented in Table 10, the performance analysis of different components of the conflict detection framework is highlighted. Ground truth presentation for the phrase (or token) is manually annotated, and its (phrase/token) performance measured in terms of accuracy.

[10]https://en.wikipedia.org/wiki/Fleiss'kappa

Although, object extraction offers more issues than action extraction, parsing operation introduced more error due to incorrect labels by the parser in relation to some verb phrase that are newly identified from the test data. However, we noted that quantitative tokens involve numerical, adverbial, and adjectival quantifiers.

Table 10. Evaluation of Conflict Detection Components

Component	**Accuracy**
Extraction of Temporal Clause	89%
Extraction of Conditional Clause	94%
Extraction of Effect Clause	87%
Action Extraction	86%
Object Extraction	96%
Extraction of Quantitative Token	85%
Assigning Polarity	93%

Further, we realized that the overall detection accuracy of quantitative tokens resulted to 84%. By this observation, lexicon derived based on the training dataset were supplemented with antonyms and synonyms. Lastly, we computed the overall polarity assignment accuracy as 93%.

In Table 11, we presented the overall recall computation of different conflict types based on the conflict detection algorithm (Algorithm 1) and the rules presented in Table 7. The recall or the (sensitivity) estimates the proportion of relevant cases that were detected between the training and test set [74].

4.8 Interpretability of Our Framework

In addition to detecting conflicts, our framework also helps identify causes and conditions contributing to different conflict type. The aim is to ensure that app reviewers are conscious of conflict ambiguities, and also assist relevant stakeholder responsible for managing the app in making intelligent conflict resolution decisions. For instance, the third conflict case identified in Table 4 identified the temporal circumstance (daily auto-

backup) as well as the sub-type (data reliability) as the source of the conflict. At the time this study was conducted, our framework indicated potential conflicts and their causes. Progressively, it could be possible to determine when conflicts are not important enough to be reported, which could reduce the number of conflicting reports to only relevant ones. For example, the effect clause, polarity, and the reviewer's context could also be used to assert the magnitude of a conflict.

4.9 Resolving Conflicts in App Reviews

One of the key focuses of this study is how to resolve conflicts identified in app reviews. Therefore, we will discuss the logical framework that was explored to fulfill the conflict resolution analysis. To characterize the sentences in app reviews which assisted the conflict resolution framework, we identified two sets of features, namely textual features and conversational features.

Textual features are those features extracted from the textual composition of a review expression, where each word in the review sentence represents a single feature after the data pre-processing phase. These features are then analyzed by the conflict detection rules and algorithm. Similarly, conversational features are features that describes the conversational context in which each word or phrase occurs in the review sentence. We identified different types of conversational features, which include (i) dimension feature, which represents the size (i.e., length of words or number of sentence) of each review sentence. (ii) chronological feature, which represent the duration (e.g. from app release date) which the review was given by the user; and (iii) participant feature, which expresses the active role of the reviewer in the review sentence (for instance, is the reviewer a collaborator in the app development's process or a third-party who only interacted with the app's functionality). These features contributed to the conflict resolution strategy that was considered on this study.

The computational approach in resolving conflicts in app reviews is largely deter-

mined by the primary stakeholders of the app in mention. This implies that the cognition of the primary purpose of the app (which formulates app's functionality) is crucial to app reviewers. Exploring different types of features (discussed earlier), and types of conflicts in app reviews contributes immensely to the conflict resolution strategy. Two key factors are examined in resolving conflicts in app reviews, namely; (i) conflict type; and (ii) conflict features. After careful examination of these two factors, the impact of these conflicts on the app's functionality and usability is assessed based on the prevailing notion for review analysis and success of the software product. Primarily, this study utilized weight assignment function which evaluates and measure the impact of conflict features (or types) based on developer and administrative stakeholder requirements.

4.9.1 Conflict Types

Investigating the conflict type identified in Table 4 is vital in resolving the conflict identified in app reviews. Our approach in using conflict types to resolve conflict is pipelined by using weight-assignment technique to evaluate the conflict impact on the requirement engineering process of the software application development. The usability of conflict type for influencing requirement engineering is primarily dependent on individual app's stakeholder goal. For instance, the weights w_i, w_j, w_k of a conflict c_i, c_j, c_k could imply an ascending order of positive impact for a particular app A_1, while they could also imply a descending order of positive impact for another app A_2. In essence, this method works by applying a parameterized weight assignment function to the conflict types detected for conflict-weight evaluation. This evaluation will assist app developers to prioritize their implementation features which will afford them the privilege to focus on the most relevant feature implementation.

4.9.2 Conflict Features

Conflict features refers to the deterministic or base feature that are responsible for the identified conflicts in app reviews. We applied summarization, and content analysis as instrument for resolving the conflicts conflicts.

Summarization: The process of summarizing review conflict helps the developers to convey a precise and concise information about the user intentions. For example, a conflict may be identified from N cluster, which have been grouped based on their target feature on an app. Summarizing their content helps in simplifying the identified conflict and is influenced by different two factors namely similar topics and reviewer's intuition. By so doing, it is easier for app developers to analyze and elicit useful information for the requirement engineering process. Although, summarization requires logical and good domain knowledge, we applied this technique as a conflict resolution strategy by assuming that we are in a closed domain. The summary output could be suggestions to app developers on maintenance or improvement strategy by considering the conflict cause or conflict type. Further, another summarization output could be identification and managing app's core features that is responsible for the conflict.

Content Analysis: By using content analysis for resolving conflicts in app reviews, we first noted that app reviewers have different insight to the app functionalities and features. Content analysis evaluates the presence and impacts of some key words, concepts or themes identified in the app reviews. For instance, we observed that there is a proportional relationship between vocabularies, and user ratings and the review length. We were able to decode recurring conflicts and their causes. Our conflict resolution findings using content analysis will assist app developers to determine the hybrid functionalities and maintains their purpose when cross-platform functionality in apps acquire consistency across the different platforms.

Generally, we infer that the impact of any identified conflict (in an app review) on the software application can be categorized as adding a feature, removing a feature,

modifying a feature, and retaining feature (i.e., other conflict types not relating to feature alteration). Figure 6 presents the stepwise approach undertaken in this study for resolving conflicts in app reviews.

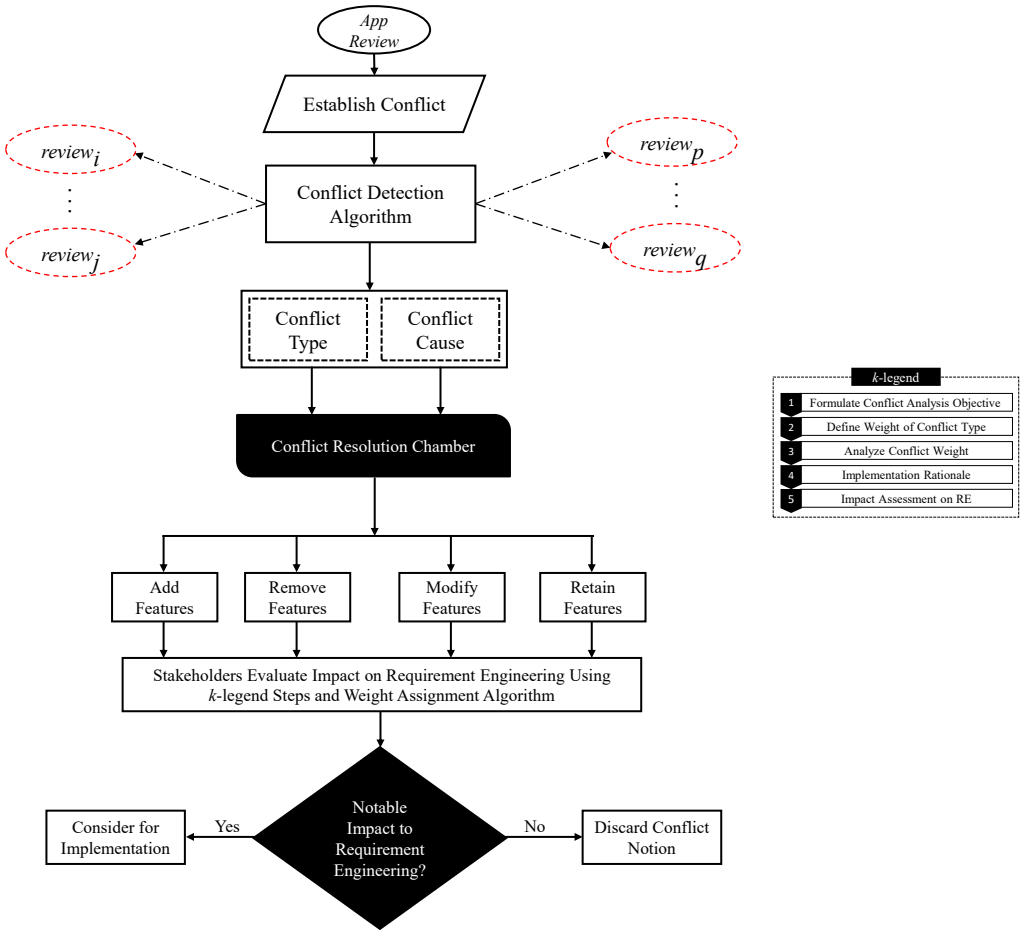

Figure 6. Conflict Resolution Procedure

4.10 Conflict Resolution Use Case

We presented the application of our conflict resolution strategy for two different use cases in Figure 7. The description is given by assuming that conflict has been detected for both cases based on our dataset.

* From the Figure 7, the seed for tables in use case 1 (UC1) and use case 2 (UC2) are referenced from table "abbc". Both UC1 and UC2 have followed the $k-$legend steps highlighted in Figure 6. Their respective objectives and conflict weight on their software product has been defined.

* The uniformity factor (weight * number of conflict) in UC1 and UC2 are highest for conditional conflict respectively. Similarly, the lowest uniformity factor for UC1 and UC2 are lowest for sub-regular and temporal conflict respectively. Based on this analysis, the object of the administrative stakeholder on product design and developer stakeholder on RE is evaluated.

* UC1 is focused on expanding the deliverable of her software product, while UC2 is only concerned about analysis for internal auditing. It is worth noting that the content analysis and summarization features comes into action in this stage after the uniformity factor has been evaluated. Based on the objective of UC1, content analysis will be prioritized, while UC2 will prioritize summarization.

* For UC1, the order of conflict type (based on uniformity factor) that is considered to satisfy their objective is given as: conditional $\rightarrow$ quantitative $\rightarrow$ direct $\rightarrow$ numerical $\rightarrow$ temporal $\rightarrow$ sub-regular conflict in descending order of influence. Similarly, for UC2, the order of conflict type (based on uniformity factor) that is required to satisfy ther objective is given as: conditional $\rightarrow$ sub-regular $\rightarrow$ numerical $\rightarrow$ direct $\rightarrow$ quantitative $\rightarrow$ temporal in descending order of influence.

* The key objective of this study findings is that conflict detection and resolution is domain-centric, which implies that its application requires solid knowledge of domain of interest as well as a good objective formulation.

4.11 Answering the Research Questions

The techniques employed in this study were aimed at answering the research questions. The following summarizes the procedures engaged in answering our research questions.

(i) **RQ1:** What are the applicable tools for detecting conflicts in app reviews, and how can the tools be optimally applied for feature extraction?

There are different computational means for detecting conflicts in app reviews, which includes natural language processing, and machine learning algorithms. In this study, we adopted a rule-based technique for detecting conflict. The rule-based algorithm is presented in Algorithm 1, and conflict detection rules are presented in Table 7. Furthermore, the rules are optimally applied by ensuring that the calibration of the algorithm's condition satisfies the problem definition to yield best solution.

(ii) **RQ2:** How do we validate the conflict detection and resolution instruments in RQ1?

Validating the conflict and resolution instrument involves an interactive process with app developers and administrative stakeholders to identify the impact of conflict analysis in app reviews. By so doing, the relevance of this study was measured. An interrogative interview with three app developers regarding conflict analysis has validate the relevance of our study in software RE, especially, the

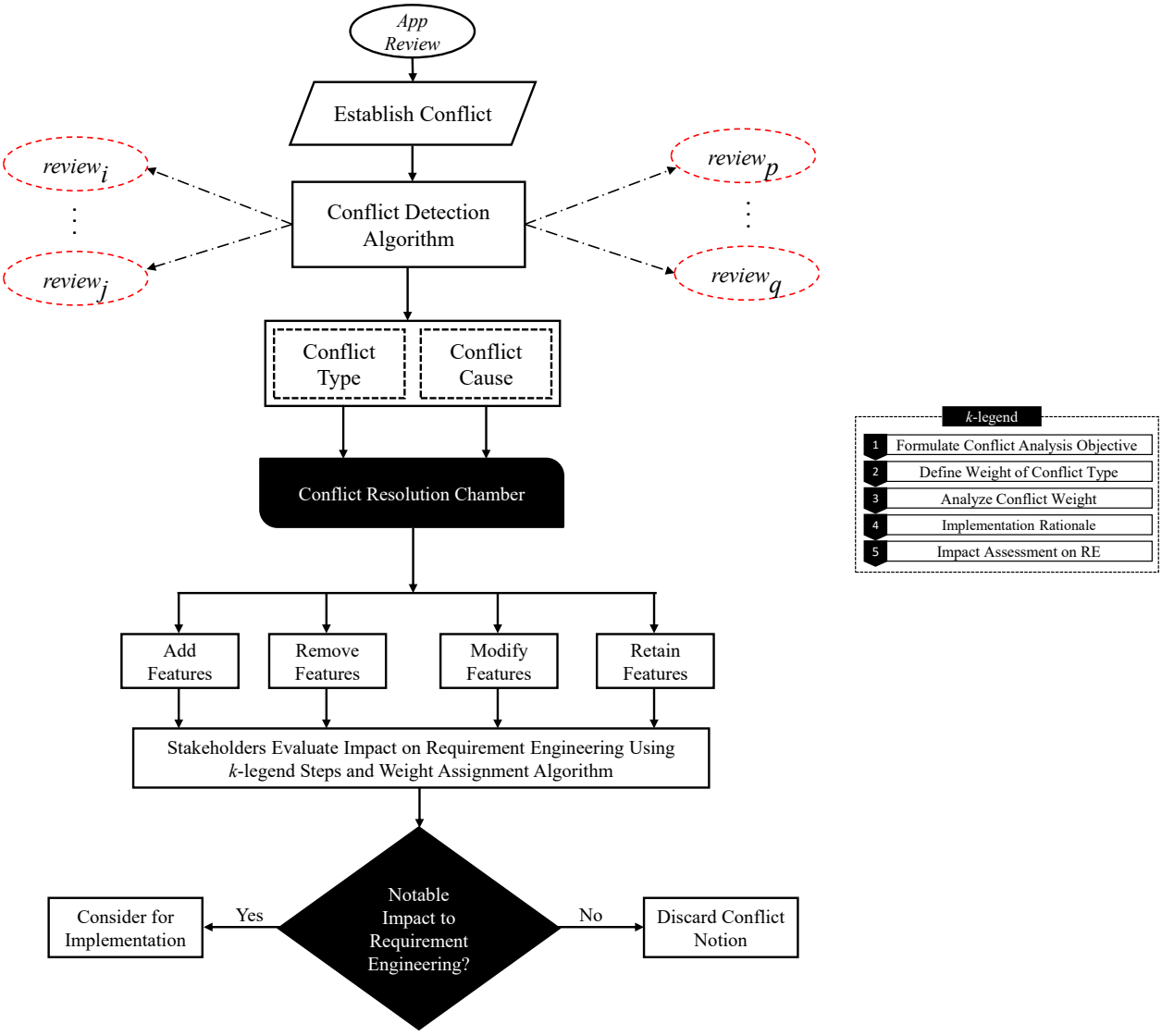

Figure 7. Conflict Resolution Use Case

conflict resolution framework.

(iii) **RQ3:** What is the impact of conflict detection and resolution on app review on RE process?

The impact of conflict detection and resolution on RE is noteworthy, especially for a sub-standard software product. To the administrative stakeholder of software application, conflict analysis helps them to decode users' perception about their software product. This in turn will help their administrative strategy for enhancing their product design. Similarly, to the app developers, conflict analysis helps in the implementation and coding features into a software product. The identified conflict in app reviews could guide developer's orientation in implementing or adjusting functional features of the software application.

4.12 Conclusion

This chapter presented the key functionality and how the results were obtained to meet the goal of the study. We presented different categories and features of conflict in app reviews as identified by the conflict detection algorithm. Further, we highlighted the conflict resolution framework which is a key component of this study. Finally, we discussed how we have been able to answer the research questions, and how we have unpacked and applied computational means to establish, detect and resolve conflict in app reviews.

Table 11. Number of Conflict Types Computed by Conflict Detection Algorithm

	Google Play				**App Store**				**Kaggle**			
Conflict Type	Actual	Detected	Recall	Accuracy	Actual	Detected	Recall	Accuracy	Actual	Detected	Recall	Accuracy
Direct	3,946	3,436	0.87	88%	1,573	1,485	0.94	94%	2,849	2,062	0.72	73&
Temporal	2,497	1,726	0.69	69%	3,292	2,018	0.61	61%	2,263	1,701	0.75	76%
Conditional	9,037	6,294	0.70	70%	4,643	2,853	0.61	62%	3,642	3,271	0.90	90%
Quantitative	2,217	1,082	0.49	50%	3,591	2,854	0.79	80%	1,621	1,274	0.79	80%
Sub-regular	1,537	923	0.60	60%	2,463	942	0.38	39%	973	629	0.65	66%
Numerical	692	524	0.76	77%	1,738	634	0.36	37%	2,921	1,038	0.36	37%

5 Discussion of Result

We presented the results obtained from this study in Chapter 4 of this thesis, as well as the impact of different components of the conflict detection and resolution framework. In this chapter, we will expand more on the results presented in Chapter 4.

5.1 Causes of Conflicts in App Review

One of the key findings of this study is to examine the causes of conflicts in app reviews. Notably, we have highlighted that conflict exists in app reviews especially, as it influences the requirement engineering process. Based on the app review analysis performed in this study, we identified factors that causes conflict in app reviews, such as human psychology (or sentiment), emotional, and theoretical factors.

Human psychology or sentiment contributes significantly to conflicts in app reviews. This is attributed to reasons such as user background knowledge and experience in interacting with software applications. For instance, a user with an extensive knowledge of app development process will have a different perspective about an app's functional implementation vis-a-vis a user who is not knowledgeable about app development process. Another psychological reasons could be difference in areas of interests in software application, which also influences the focus of a user as they engage the functions of a software application. For example, a user U_i can be concerned about the technical pipeline of a software, while another user U_j can be concerned about functional (or benefits) of the same software application. The dichotomy of these psychological reasons could influence their feedback, and the different perspectives could lead to conflict as their interest formulates their point of concerns when presenting their reviews.

Emotional factor involves the internal feeling or disposition of an individual when they interact with app functionalities. A broad example to illustrate how user emotions can influence conflicts in reviews could be: accumulation of mental stress of a user over

a certain period of time. Submitting app reviews with a stressed mental environment could lead to false or bias reviews about a software application. The implication of this type of review is that it may not portray an accurate app evaluation. Hence, conflicting notions in app reviews could be experienced based on this concern. **Theoretical** factor refers to the deductive or logical interpretation of an app's analysis by a user. This is largely influenced by variables such as domain knowledge or intellectual engagement of a user. Conflict could arise based on theoretical factors as analysis of user review context could be difficult to uncover in some cases.

5.2 Solving Challenges with Semantic Token Extraction

We tackled the challenges highlighted in Section 4.3 by using semantic rules and external grammatical knowledge-base. First, we used the MetaMap (a knowledgebase framework for identifying Metathesaurus in sentences [75]) to filter non-objects and objects in sentences. The MetaMap was configured to only identify relevant objects (such as buttons, and navigation control) applicable to our domain interest. Secondly, we applied multiple knowledg-bases to establish object hierarchy. Lastly, we extracted compound object by utilizing semantic rules. For example, we assumed that if the constituent terms are nouns in a compound object, then we regard all as candidate object. Objects are usually connected using modifiers which signify quantity and sub-type, thereby leading to quantitative and sub-regular conflicts.

Action Clause (a_g): We identified the action clause by examining expressions that begins with a verb phrase and contains either an imperative or declarative statements. An example of the imperative statement could be: *"include a toggle button on the welcome screen"*, while an example of a declarative statement could be *"adding a button to toggle the order and checkout view helps to navigate the app easily"*. By inference, it is worth noting that we regard phrasal verbs such as *stick to*, action verbs such as *open*, and negate verbs such as *avoid clicking* as action tokens.

Quantitative Clause (q_g): This clause represents the frequency of objects or duration of action, which could be indefinite (e.g., using quantifier like more and few) or exact (e.g., using numeric values). Although, core NLP assigns quantitative tokens as a cardinal number and adverbial quantifier phrase, explicit details are required to infer the right semantic of a review statement.

Subject Clause (s_g): The subject retains the context features to determine optimal interpretation of a review statement. The context features in a review represents the target subject. For example, *"the female player size in the game should increase by half of the current size"*. Here, the subject is the player (female) and size is (half-increament).

> **Note:** The semantic rules mentioned in phases 1 and 2 are steered by linguistic designs which enabled us to identify semantics and conflicts presents in the reviews.

5.3 Summary of Conflict Detection Phases

As presented in Section 4.1 of Chapter 4, empirically, we observed that conflicts occur in reviews containing more than one sentence representing different types of information, such as: (i) An action that is criticized in a sentence is typically initiated by at least one sentence containing another action(s). (ii) Successive expressions may indicate different actions with dissimilar objects. (iii) A review pair contains action-to-effect tuple, where a review conveys an action, while the other review exposes the effect of the identified action; and (iv) The action expressed in a statement is elaborated in the next statement(s). Considering cases (iii) and (iv), their semantic tokens are combined because the statement reveals the similar actions, whereas, for the other cases, each sentences generates a distinct tuple of semantic tokens.

Conflict Resolution Use Case We present the overall summary of the conflict resolution as it applies to cases presented in Section 4.10.

Step 1: Identify the various types of conflicts in app reviews using the rules presented in Table 7 based on Algorithm 1.

Step 2: Follow the five steps highlighted by the k-legend in Figure 6, noting that steps 1-3 are primarily applicable to the administrative stakeholders of the software product, while steps 4 and 5 are primarily applicable to the software developers. To ensure uniformity in conflict types, the weight assigned to each conflict is multiplied by the number of conflict pairs identified in the review. The weight assignment is best assigned by the administrative stakeholder and app developer for determining the impact of conflict features on their product.

Step 3: Examine conflict features (summarization, and content analysis) based on step 2 above to guide the conflict resolution strategy as it applies to the domain of interest.

A use case and application of the conflict resolution strategy steps is presented in Figure 7.

5.4 Overview of Conflict in User Review

Relating to this study, conflicts arise when their is an overlap in the opinions or interest of app users which could be caused by contextual, physiological, emotional, theoretical or intellectual reasons. Interestingly, proper analysis of conflicts contributes significantly to the overall success of an app that has been developed and is in use [32, 76]. The enhanced version of the app is achieved by analyzing the feedback of users who have interacted with the app's features and functionality. This can be achieved by examining the app's targeted features or components that has been identified by app users and which has been expressed in their reviews. Example of conflict obtained in app reviews includes (i) deception to determine if a user likes an app feature or not. (ii) estimating if an app is

a fraud app or not; and (iii) prioritization of app features in terms of addition, removal, modification, or retaining features, as highlighted in Figure 7.

5.5 Impact of Context Awareness

In this study, we have shown that conflict could occur between any n review, provided that the conditions mentioned in Section 3.7 of Chapter 3is satisfied. The context of a review given by a user may differ based on different factors (such psychological or emotional) or component of an app addressed in the review. However, the experiments performed in this study observed that the context of a review changes based on different user perspective, even for a similar app functionality. These changes also affect the conflict type, but does not affect the base architecture of framework responsible for conflict analysis.

5.6 Reproducibility of Study Findings

For reproducing our study's framework and to obtain similar result, our implementation details can be found by clicking here.

5.7 Conclusion

In this chapter, we expatiate on the results of different components and features of the conflict analysis framework. We describe how challenges associated with extracting semantic tokens was addressed. Further, we expatiate on the human-based factor responsible for conflicts in app reviews, and we summarize the conflict resolution strategy that was involved in this study. In Chapter 7, we will present the overall research summary, study contribution as well as the recommendation for further research.

6 Threats to Validity

To provide an extensive insight to our study, we identified four major threats to validity, which are highlighted below;

* *First*, the app reviews considered in this study were all assumed to contain a minimum of at least one conflicting parameter when analyzed with other app reviews. This claim may not always be true as some app review statement may not contain relevant information that is sufficient for identifying and asserting conflicts. An extension to this threat could involve integrating an analytical framework to detect reviews that does not contain conflict parameters.

* *Secondly*, the study did not center around any particular domain of interest (i.e field of business). Having a domain knowledge and applying the study findings to a particular field of interest (of a software product) would have ascertained relatable usability of the research, rather than the generic usability of conflict that was considered in this study.

* *Thirdly*, the use cases (highlighted in Figure 6) were not designed by app developers or the administrative stakeholders, but we only put ourselves in the position of the stakeholders to relate with their requirement engineering and design experience. However, we discussed the use case with two app developers working in the commercial space, and they assert the use case to be valid and applicable to their requirement engineering task. Conversely, this may not be generalizable to all requirement engineering tasks for all commercial products. We only presume the applicability and capabilities of our work with the use cases.

* *Lastly*, the conflict types and conflict features highlighted in Table 4 is based on our personal hypothetical claim. This may not reflect all the possible conflict types

(or cases) as there is possibility for an extended conflict case, especially as research in this field expands and more discoveries in app reviews are uncovered.

7 Conclusion and Future Work

Ensuring a secure and optimal functioning of software applications is critical, as their usability depends on the active functioning of different app components. The optimal procedures of these components can be achieved by retrieving feedback from app's users. The feedback is regarded as reviews given by users, and conflicts could arise in the reviews based on different factors (identified in Chapter 3). An applicable approach to resolve the identified conflicts is critical for a good requirement engineering process in software development task. This study identified different types and causes of conflict and the algorithmic rationale explored for their type-classification. In addition, this study focused on the how app review features were analyzed and examined to resolve the identified conflict(s) in app review in order to ensure that the requirement engineering component of the software application is best-enhanced. This research aims to satisfy the identified question to app review analysis to fulfil the objective of app-stakeholder goals.

To achieve these goals, we developed a framework which comprised a set of semantic rules (as in Table 7) and an algorithm (Algorithm 1) for detecting conflicts, which uncovers conflicts pairs as well as conflict types by evaluating the semantics of review statements. In contrast to statistical learning-based conflict detection frameworks; (i) our algorithm is able to detect conflict by using a context-aware technique with minimal training dataset. (ii) our algorithm could inform app developers and administrative-stakeholders about the cause and type of conflict identified in the app reviews. This will assist app developers in decision-making process of resolving the identified conflict, and also benefit administrative stakeholders in strategy planning for product optimization.

We explored a rule-oriented technique for detecting conflicts in app reviews, and we primarily considered ranking of conflicting features (based on impact on app deliverable and stakeholder goals) as the resolution strategy. Using rule-based conflict resolution procedures could be difficult, requiring a large number of rules to cover all possible

scenarios. In this study, we considered the standard and generic conditions that are applicable to the defined rules. Conversely, it could be impossible to ensure that the system will always perform as expected as there could be infiltration of human-based or system-based error which could alter the optimal performance of the framework.

7.1 Contribution to Body of Knowledge

Although, research pertaining conflict detection and resolution is just gaining attention and traction in recent decades, however, based on our findings and research insight, we have been able to provide a computational perspective to conflict analysis in app reviews. The findings of this study has contributed to the body of knowledge in the following ways.

(i) We provided a technique founded on computational rules for detecting conflicting notions that exist in app reviews. Further, we proposed a resolution strategy for resolving the identified conflict. Based on this, we were able to assert that not all reviews about a software application are conflicting. In addition, we identified different factors that could cause or influence conflicts identified in app reviews.

(ii) Another contribution of our work is its applicability and relevance to the administrative stakeholders and developers of a software product. To the administrative stakeholders, the findings of our conflict analysis can guide the system design strategy required for improving the user experience of their software product. Similarly, our findings is beneficial to the app developers in the area of requirement engineering, by helping them identify the system requirements needed for implementation to satisfy user expectations in a software product.

(iii) The broad influence of our study findings can be extended to its benefit in economical value. This is so because, one of the major end goals behind the development of software application is to derive economic benefit. Therefore, based on the

analysis of conflicts identified and resolved in reviews given by users to an existing (in-house or competitive product) software product, an enhanced version of the software is developed. Based on the enhanced product version, more users will be interested in engaging the application, which in turns directly or indirectly generate more economic benefit for the app owners (based on the primary function of the software).

7.2 Recommendation for Future Works

To expand on the findings of this research, we highlight two key recommendations that could be explored for further research as an extension to this study.

(I) Implementation of additional computational strategy that can determine when conflicts are not critical enough to resolve, limiting the number of conflicting cases to the necessary ones. For instance, the reviewer's context, the effect and review's polarity could be parameterized and considered to infer the intensity of a conflict.

(II) Another recommendation is the proactive resolution of conflicts between app reviews, irrespective of the source, type, and cause of conflict. This is important by considering a static analysis to determine whether a collection of app reviews adheres to functional impact of software requirement engineering demand.

References

[1] A. Ritter, S. Soderland, D. Downey, and O. Etzioni, "It is a contradiction. no, it is not: A case study using functional relations," 10 2008, pp. 11–20.

[2] W. Luiz, F. Viegas, R. Alencar, F. Mourão, T. Salles, D. Carvalho, M. A. Gonçalves, and L. Rocha, "A feature-oriented sentiment rating for mobile app reviews," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 1909–1918. [Online]. Available: https://doi.org/10.1145/3178876.3186168

[3] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 108–111, 2012.

[4] G. Qiu, B. Liu, J. Bu, and C. Chen, "Opinion word expansion and target extraction through double propagation," *Computational Linguistics*, vol. 37, pp. 9–27, 2011.

[5] F. A. Shah, K. Sirts, and D. Pfahl, "Simulating the impact of annotation guidelines and annotated data on extracting app features from app reviews," in *ICSOFT*, 2019.

[6] A. Kunaefi and M. Aritsugi, "Extracting arguments based on user decisions in app reviews," *IEEE Access*, vol. 9, pp. 45 078–45 094, 2021.

[7] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 749–759, 2015.

[8] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 2014, pp. 153–162.

[9] A. F. Araujo, M. P. S. Gôlo, and R. M. Marcacini, "Opinion mining for app reviews: An analysis of textual representation and predictive models," *Automated Software Engg.*, vol. 29, no. 1, may 2022. [Online]. Available: https://doi.org/10.1007/s10515-021-00301-1

[10] M. Hu and B. Liu, "Mining and summarizing customer reviews," *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.

[11] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," *2013 35th International Conference on Software Engineering (ICSE)*, pp. 582–591, 2013.

[12] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Ar-miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 767–778. [Online]. Available: https://doi.org/10.1145/2568225.2568263

[13] M. Chaa, O. Nouali, and P. Bellot, "Leveraging app features to improve mobile app retrieval," *Int. J. Intell. Inf. Database Syst.*, vol. 14, no. 2, p. 177–197, jan 2021. [Online]. Available: https://doi.org/10.1504/ijiids.2021.114530

[14] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA:

Association for Computing Machinery, 2013, p. 1276–1284. [Online]. Available: https://doi.org/10.1145/2487575.2488202

[15] D. Mukherjee, A. Ahmadi, M. V. Pour, and J. Reardon, "An empirical study on user reviews targeting mobile apps' security & privacy," *ArXiv*, vol. abs/2010.06371, 2020.

[16] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 106–117.

[17] J. Huang, O. Etzioni, L. Zettlemoyer, K. Clark, and C. Lee, "Revminer: an extractive interface for navigating reviews on a smartphone," *Proceedings of the 25th annual ACM symposium on User interface software and technology*, 2012.

[18] R. Afzalur, *Managing conflict in organizations*. New Brunswick Publisher, 2011.

[19] R. König, R. Renner, and C. Schaffner, "The operational meaning of min- and max-entropy," *IEEE Transactions on Information Theory*, vol. 55, pp. 4337–4347, 2009.

[20] Y. Jo and A. H. Oh, "Aspect and sentiment unification model for online review analysis," in *WSDM '11*, 2011.

[21] L. Zhuang, F. Jing, and X. Zhu, "Movie review mining and summarization," in *CIKM '06*, 2006.

[22] K. Yatani, M. Novati, A. Trusty, and K. N. Truong, "Review spotlight: a user interface for summarizing user-generated reviews using adjective-noun word pairs," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011.

[23] J. Amman. Students develop apps to tackle pressing social issues (the jordan times), unesco. [Online]. Available: https://en.unesco.org/news/students-develop-apps-tackle-pressing-social-issues-jordan-times

[24] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," *2013 21st IEEE International Requirements Engineering Conference (RE)*, pp. 125–134, 2013.

[25] S. Savage, A. G. Forbes, C. Toxtli, G. McKenzie, S. Desai, and T. Höllerer, "Visualizing targeted audiences," in *COOP*, 2014.

[26] W. J. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 123–133, 2015.

[27] R. Crouch, C. Condoravdi, V. De Paiva, R. Stolle, and D. Bobrow, "Entailment, intensionality and text understanding," *Information Systems and e-Business Management*, 12 2003.

[28] S. Harabagiu, A. Hickl, and V. Lacatusu, "Negation, contrast and contradiction in text processing." 01 2006.

[29] A. S. J and K. P. K, "A Literature Review on Application of Sentiment Analysis Using Machine Learning Techniques," *International Journal of Applied Engineering and Management Letters (IJAEML)*, vol. 4, no. 2, pp. 41–67, Aug. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3977576

[30] P. D. Turney, "Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews," in *ACL*, 2002.

[31] D. Kawahara, K. Inui, and S. Kurohashi, "Identifying contradictory and contrastive relations between statements to outline web information on a given topic," in *COLING*, 2010.

[32] S. Brown. Machine learning, explained by mit sloan school of management. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[33] C. Blake, "Beyond genes, proteins, and abstracts: Identifying scientific claims from full-text biomedical articles," *Journal of biomedical informatics*, vol. 43 2, pp. 173–89, 2010.

[34] A. Alamri and M. Stevenson, "A corpus of potentially contradictory research claims from cardiovascular research abstracts," *Journal of Biomedical Semantics*, vol. 7, 2016.

[35] W. W. Chapman, W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan, "A simple algorithm for identifying negated findings and diseases in discharge summaries," *Journal of biomedical informatics*, vol. 34 5, pp. 301–10, 2001.

[36] T. Hirave, S. Malgaonkar, M. Alwash, J. Cheriyan, and S. Surve, "Analysis and prioritization of app reviews," in *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*, 2019, pp. 1–8.

[37] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *EMNLP*, 2011.

[38] B. T. McInnes, T. Pedersen, and S. V. S. Pakhomov, "Umls-interface and umls-similarity : Open source software for measuring paths and semantic similarity," *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2009, pp. 431–5, 2009.

[39] H. A. J. Top 30 nlp use cases: Comprehensive guide for 2022. aimultiple. [Online]. Available: https://research.aimultiple.com/nlp-use-cases/

[40] C. Zhang, H. Wang, R. Wang, Y. Guo, and G. Xu, "Re-checking app behavior against app description in the context of third-party libraries," in *SEKE*, 2018.

[41] C. Condoravdi, D. Crouch, V. C. V. de Paiva, R. Stolle, and D. G. Bobrow, "Entailment, intensionality and text understanding," in *HLT-NAACL 2003*, 2003.

[42] M. Tavakoli, L. Zhao, A. Heydari, and G. Nenadić, "Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools," *Expert Systems with Applications*, vol. 113, pp. 186–199, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418303361

[43] Y. Liu, Z. Shi, and A. Sarkar, "Exploiting rich syntactic information for relationship extraction from biomedical articles," in *NAACL*, 2007.

[44] H. Yu and V. Hatzivassiloglou, "Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences," in *EMNLP*, 2003.

[45] F. A. Shah, K. Sirts, and D. Pfahl, *Using App Reviews for Competitive Analysis: Tool Support*. New York, NY, USA: Association for Computing Machinery, 2019, p. 40–46. [Online]. Available: https://doi.org/10.1145/3340496.3342756

[46] T. Johann, C. Stanik, B. AlirezaM.Alizadeh, and W. Maalej, "Safe: A simple approach for feature extraction from app descriptions and app reviews," *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 21–30, 2017.

[47] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, May 2013, pp. 41–44.

[48] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 582–591.

[49] D. Ravichandran and E. H. Hovy, "Learning surface text patterns for a question answering system," in *ACL*, 2002.

[50] D. Downey, O. Etzioni, and S. Soderland, "A probabilistic model of redundancy in information extraction," in *IJCAI*, 2005.

[51] R. Aralikatte, G. Sridhara, N. Gantayat, and S. Mani, "Fault in your stars: An analysis of android app reviews," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, ser. CoDS-COMAD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 57–66. [Online]. Available: https://doi.org/10.1145/3152494.3152500

[52] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, pp. 618–643, 2010.

[53] M. R. Islam, "Numeric rating of apps on google play store by sentiment analysis on user reviews," in *2014 International Conference on Electrical Engineering and Information Communication Technology*, 2014, pp. 1–4.

[54] C. Fitzgerald, E. Letier, and A. C. W. Finkelstein, "Early failure prediction in feature request management systems: an extended study," *Requirements Engineering*, vol. 17, pp. 117–132, 2012.

[55] P. Joames. Rating and review of android app success. [Online]. Available: https://www.kaggle.com/code/mmsant/ratings-and-reviews-and-android-app-success/data?select=googleplaystore_user_reviews.csv

[56] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 499–510. [Online]. Available: https://doi.org/10.1145/2950290.2950299

[57] L. V. G. Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," *2013 35th International Conference on Software Engineering (ICSE)*, pp. 582–591, 2013.

[58] T. Korenius, J. Laurikkala, K. Järvelin, and M. Juhola, "Stemming and lemmatization in the clustering of finnish text documents," in *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, ser. CIKM '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 625–633. [Online]. Available: https://doi.org/10.1145/1031171.1031285

[59] L. Hickman, S. Thapa, L. Tay, M. Cao, and P. Srinivasan, "Text preprocessing for text mining in organizational research: Review and recommendations," *Organizational Research Methods*, vol. 25, no. 1, pp. 114–146, 2022.

[60] Wikipedia. Wikipedia:list of english contractions. [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:List_of_English_contractions

[61] J. Chevalier and D. Mayzlin, "The effect of word of mouth on sales concentration," *International Journal of Research in Marketing*, vol. 32, no. 2, pp. 207–218,

2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167811615000336

[62] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS quarterly*, pp. 319–340, 1989.

[63] A.-M. Popescu and O. Etzioni, "Extracting product features and opinions from reviews," in *HLT*, 2005.

[64] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 108–111, 2012.

[65] M. Hu and B. Liu, "Mining opinion features in customer reviews," in *AAAI*, 2004.

[66] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 41–44, 2013.

[67] M. A. Thelwall, K. Buckley, and G. Paltoglou, "Sentiment strength detection for the social web," *J. Assoc. Inf. Sci. Technol.*, vol. 63, pp. 163–173, 2012.

[68] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data mining and knowledge discovery," *Data Mining and Knowledge Discovery*, 1997.

[69] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network." USA: Association for Computational Linguistics, 2003. [Online]. Available: https://doi.org/10.3115/1073445.1073478

[70] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of*

52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 55–60. [Online]. Available: https://aclanthology.org/P14-5010

[71] J. Eastwood, *Oxford Guide to English Grammar*. Oxford University Press, 2004.

[72] A. Chang and C. Manning, "Sutime: A library for recognizing and normalizing time expressions," *IEEE Working Conference on Mining Software Repositories (MSR)*, 11 2012.

[73] C. Fellbaum, *A Semantic Network of English: The Mother of All WordNets*. Dordrecht: Springer Netherlands, 1998, pp. 137–148. [Online]. Available: https://doi.org/10.1007/978-94-017-1491-4_6

[74] B. Juba and H. S. Le, "Precision-recall versus accuracy and the role of large data sets," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4039–4048.

[75] A. Aronson and F.-M. Lang, "An overview of metamap: Historical perspective and recent advances," *Journal of the American Medical Informatics Association : JAMIA*, vol. 17, pp. 229–36, 05 2010.

[76] N. Seyff, F. Graf, and N. A. M. Maiden, "Using mobile re tools to give end-users their own voice," *2010 18th IEEE International Requirements Engineering Conference*, pp. 37–46, 2010.

I. Glossary

Acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CSV	Comma Separated Values
ID	Identification
LDA	Linear Discriminant Analysis
ML	Machine Learning
NFR	Non-Functional Requirement
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NP	Noun Phrase
POS	Part of Speech
RE	Requirements Engineering
UC	Use Case
URL	Uniform Resource Locator
VADER	Valence Aware Dictionary and Sentiment Reasoner
VP	Verb Phrase

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Solagbade Ayodele Enitilo**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   A Model for Detecting and Resolving Conflicts in Features Extracted from App User Reviews,

   supervised by Ishaya Peni Gambo, PhD.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Solagbade Ayodele Enitilo

08/08/2022