Your Opinions Let us Know: Mining Social Network Sites to Evolve Software Product Lines

Nazakat Ali[1], Sangwon Hwang[2], and Jang-Eui Hong[1*]
[1] Department of Computer Science, Chungbuk National University
Cheongju, South Korea
[2]Department of Computer Science & Telecommunication Engineering
Yonsei University, Wonju, South Korea
[E-mail : arsenal@yonsei.ac.kr]
[E-mail : nazakatali@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr]
*Corresponding author : Jang-Eui Hong

Received February 27, 2019; revised April 29, 2019; accepted June 15, 2019; published August 31, 2019

Abstract

Software product lines (SPLs) are complex software systems by nature due to their common reference architecture and interdependencies. Therefore, any form of evolution can lead to a more complex situation than a single system. On the other hand, software product lines are developed keeping long-term perspectives in mind, which are expected to have a considerable lifespan and a long-term investment. SPL development organizations need to consider software evolution in a systematic way due to their complexity and size. Addressing new user requirements over time is one of the most crucial factors in the successful implementation SPL. Thus, the addition of new requirements or the rapid context change is common in SPL products. To cope with rapid change several researchers have discussed the evolution of software product lines. However, for the evolution of an SPL, the literature did not present a systematic process that would define activities in such a way that would lead to the rapid evolution of software. Our study aims to provide a requirements-driven process that speeds up the requirements engineering process using social network sites in order to achieve rapid software evolution. We used classification, topic modeling, and sentiment extraction to elicit user requirements. Lastly, we conducted a case study on the smartwatch domain to validate our proposed approach. Our results show that users' opinions can contain useful information which can be used by software SPL organizations to evolve their products. Furthermore, our investigation results demonstrate that machine learning algorithms have the capacity to identify relevant information automatically.

Keywords: Software product line evolution, Social network sites, Requirements-driven, Architecture design, classification, machine learning

A preliminary version of this paper was presented at ICONI 2018, and was selected as an outstanding paper. This research was supported by Next-Generation Information Computing Development Program through the NRF of Korea funded by the Ministry of Science, ICT (NRF-2014M3C4A7030505, NRF-2017M3C4A7066479).

http://doi.org/10.3837/tiis.2019.08.021 **ISSN : 1976-7277**

1. Introduction

Software Product Line Engineering (SPL) is one of the most important paradigms for the development of software systems. The SPL is "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment" [1]. Different abstraction levels are required to be deemed both in SPL and its derived products. On SPL level, the reference architecture is built to address the commonalities while the variabilities are defined to address product-specific requirements. There are a number of reasons that trigger the evolution of SPLs. The evolution of SPL can happen on three various levels [2, 3]; 1) When a set of new SPL features are added or deleted, it leads to the evolution of SPLs. Additionally, SPLs can be merged if SPLs became a similar system over time [2]. The Split of SPLs can also lead to its evolution when some parts of an SPL are evolved in a different direction over time [3]. 2) When some set of products or a new product is added into an SPL or deleted the old deprecated one, then it triggers SPL evolution as mentioned in [4]. 3) When some change occurs in the core asset level, meaning when a new requirement is added, deleted or modified then these changes have to reflect in SPL products. **Fig. 1** shows the level of evolutions with strategies and the cause of evolution. According to Seva Maye [5], the evolution in software is triggered due to the emergent of needs from its user. According to the author, 67 percent of software evolution happens due to user new demands. The SPL products must be evolved continuously to address the current demands of its end-users. Hence, the addition of emerging requirements are a common symptom of software evolution in SPL. To cope with rapid change, we propose a requirements-driven SPL evolution process that employs Social Network Sites (SNS) to get rapid feedback from its end-users and then present a methodology to create and evolve SPL's reference architecture. Therefore, our evolution strategy is a proactive strategy triggered by user new demands. In summary, the literature did not present a systematic process that would define activities in such a way that would lead to the rapid evolution of SPLs.

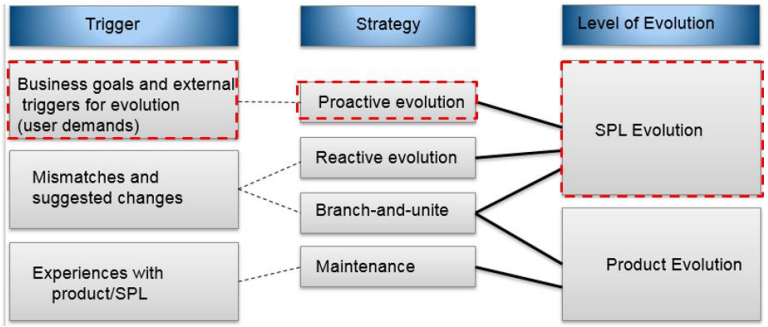

Fig. 1. SPL evolution strategies with respect to trigger and level of evolution [47]

Mainly, the contribution of this work is threefold. First, we present a process for the rapid evolution of SPLs, where we use SNS as a platform to extract user requirements and propose an architecture design methodology to reflect rapid changes to the SPL products. Second, we investigated users' opinions related to smartwatch domain and categorized them into bug reports, new feature demands, and quality attributes. Finally, we applied supervised Machine Learning (ML) algorithms to see up to what extent software related opinions can be classified automatically into bug reports, new feature demands, and quality attributes (non-functional requirements). We also apply topic modeling to see whether it helps to elicit requirements or

not. We extract sentiment of users' opinions to see overall mode of users for our dataset. This paper is organized as follow: Section 2 discuss related work relevant to our research. Section 3 discuss the proposing approach in detail, in Section 4 we conducted a case study to validate our approach. In Section 5, we discuss threats to the validity of this study while in Section 6, discuss conclusion remarks of this paper.

2. Related Work

Software evolution always remained a challenging job for the research community. The research [10] presented a taxonomy for requirements-driven evolution of SPL systems. The authors have presented a foundation for the identification of SPL requirements to support evolution. The authors of studies [11, 12] have used Twitter feeds and user comments respectively to get instant feedback as requirements to evolve software systems. The authors just provided a mechanism to get requirements from Twitter feeds and user comments but did not discuss the architecture. The rapid evolution of SPL requires both a fast approach to get instantaneous feedback and appropriate architecting methodology to achieve evolution. The study [13] has proposed a development process for building adaptive software architecture. The process did not consider the evolution of SPLs at all and did not address the requirements engineering phase. Jiang *et.al* [14] proposed an approach for requirements evolution from an economic perspective. The authors have examined many online reviews and combined the techniques of machine learning, opinion mining, and text clustering with a utility-oriented econometric model to find system aspects related to software marketing and sales for the revising requirements. Furthermore, their approach was useful for system requirement analysts by suggesting economically valuable requirements. Itezel *et.al* [15] conducted research that automatically analyze textual messages from app store reviews, user forums, and open source software mailing-lists. The authors applied some NLP (Natural Language Processing) techniques to filter out unnecessary data then applied automatic classification techniques and text mining techniques user comments into various categories. Furthermore, the authors used 40872 comments from OpenOffice community for automatic categorization. Wei *et.al* [16] presented an approach for eliciting evolutionary requirements by investigating online reviews using various techniques of syntactic relation-based propagation approach, user satisfaction analysis, and S-GN. The authors' presented methodology helped software developers with finding evolutionary requirements related to software revenue. Marc *et.al* [17] presented a framework that combined user feedback and monitoring data in web and mobile context to support continuous requirements elicitation. The user feedback mechanisms allow end-users to express their problems, experiences, and opinions, while monitoring brings valuable information about runtime events. Noor *et.al* [18] proposed an approach to aid requirements reuse. The proposed approach used NLP and information retrieval techniques to determine user requirements. The authors extracted thirty-two software reviews for online learning software which were compiled by experts. Fuzzy C-Means clustering and Semantic Analysis with Singular Value Documentation were used to cluster the similar review documents. Souza *et.al* [19] discussed the *evolution requirements* that specify the ripple effect of change of one requirement to other requirements when certain conditions apply. The authors proposed a technique to model such kind of requirements and to operationalize them at runtime to achieve runtime adaptability. This approach permitted to explicit modeling the changes to other requirements models in response to certain term and conditions like requirements failures. Guzman *et al.* [50] used app stores to elicit user requirements. The authors analyzed 2,560 app reviews written by users from eight countries. Perini [51]

proposed a tool that enables a data-driven software engineering process. The tool can collect user feedback from SNS, analyze and decide about the collected feedback for software evolution plan. Morales-Ramirez *et al.* [52] proposed a process to elicit user requirements from online discussions. The authors used linguistic technique based on speech acts for the analyis of discussions with the ultimate goal of discovering requirements-relevant information. The authors classified messages into Features and Others with F-measure 0.81 and 0.84 respectively.

3. Proposed Approach

SPLs need to be evolved quickly to respond to the emerging demands of its end-users. Therefore, we present an approach that collects user feedback from SNS periodically. The collected corpus is then interpreted as requirements using a number of NLP and ML techniques. Then the requirements are categorized into variability and commonality. Based on the variability and commonality, the variability model is then identified and finally, a reference architecture is built. **Fig. 2** shows a global overview of our process. Our process has three phases: phase 1) Requirements engineering 2) Feature Modeling 3) Reference architecture designing. Our requirements elicitation process (❶), is a cyclic process where we get new requirements on every iteration. If new requirements are identified and decided to be added into variability model (❷), then the adaptive architecture (❸) becomes aware of variability model change and configures itself to reflect newly added requirements in its reference architecture.

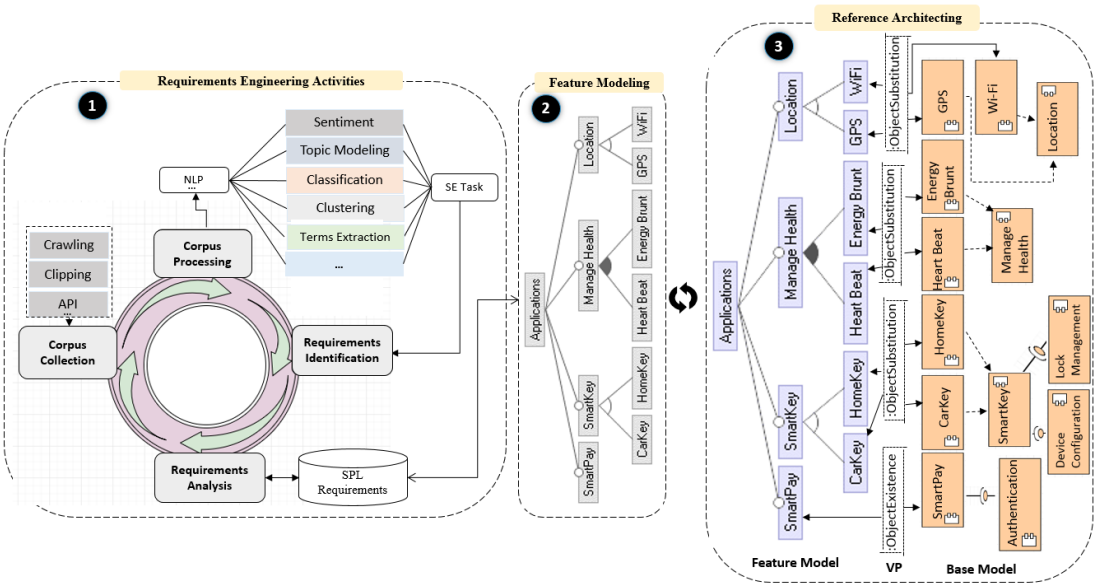

Fig. 2. Proposed Approach: SPL architecture evolution approach based on SNS user feedbacks

2.1 Requirements Elicitation Phase

The needs of users are rapidly changing over time and SPL organizations have to address those needs promptly. Otherwise, the users will look for alternatives that could harm the SPL organizations financially. Therefore, SPL organizations have to search for fast ways to get feedback from their users in order to address their current demands. SNS have enabled a huge population of end-users of any software product to openly share their concerns and

experiences. It also provides a way for millions of users across the globe to share their experiences and problems through a real-time status update. This feedback can be collected and processed to help SPL developers to interpret users' requirements, uncover bugs, and plan the evolution of their software system.

We have conducted primary studies [6, 7] to know the suitability of SNS to get user requirements. Our studies revealed that SNS have a number of benefits to use as a platform to extract user requirements including a quick response from users, access worldwide users, continuous user follow-up, and cost-effectiveness. However, extracting requirements from the collected corpus still remained a big challenge. We employ a number of NLP techniques along with ML techniques to fully utilize the corpus to get user requirements (**Fig. 2** ❶). The techniques including topic modeling, opinion classification, and sentiment analysis are utilized to identify the user concerns raised in software-relevant feedback. In this paper, we only implement sentiment analysis, topic modeling, and classification to elicit user requirements. After elicitation of user requirements, the requirements are then analyzed in order to determine commonality and variability along with the verification of identified requirements. For requirements elicitation, we extract user opinion from Facebook and Twitter for smartwatch domain. The opinions are classified by applying supervised ML techniques. The details of how to elicit user requirements are described in the data analysis phase of our case study.

2.2 Feature Modeling Phase

When requirements are elicited and analyzed, then the Feature Model (FM) is built manually. A number of techniques are presented both by academia and industries to manage the variability [8]. We have decided to use Common Variability Language (CVL) [9] to model the features due to some reasons: it does not need a complex formalism and offers a base model to be represented along with variability, it provides the capability of variability addition into the base model without remodeling it. **Fig. 3** shows an overview of how CVL works.

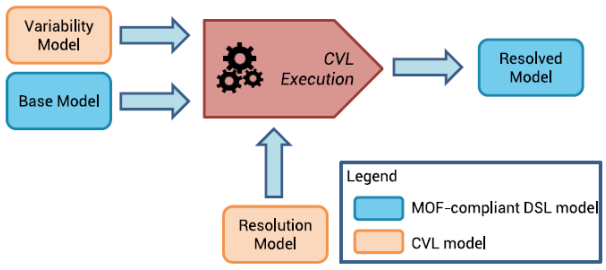

Fig. 3. CVL approach overview (adopted from [9])

The CVL is composed of three components: the variability model, the base model, and the resolution model. A specific product can be produced by utilizing CVL execution which is consuming CVL model. The base model represents concrete elements for producing various products of an SPL. The variability and commonality of SPL are represented using the variability model. The variability model is composed of three parts: the variation points (VP), Object Constraint Language (OCL), and variability specification tree (VSpec tree). Resolution model shows the specific configuration of an SPL where each VSpec tree is resolved. **Fig. 4** shows an example of the variability model, a resolution model, variation points, base model and resolved model for our smartwatch case study.

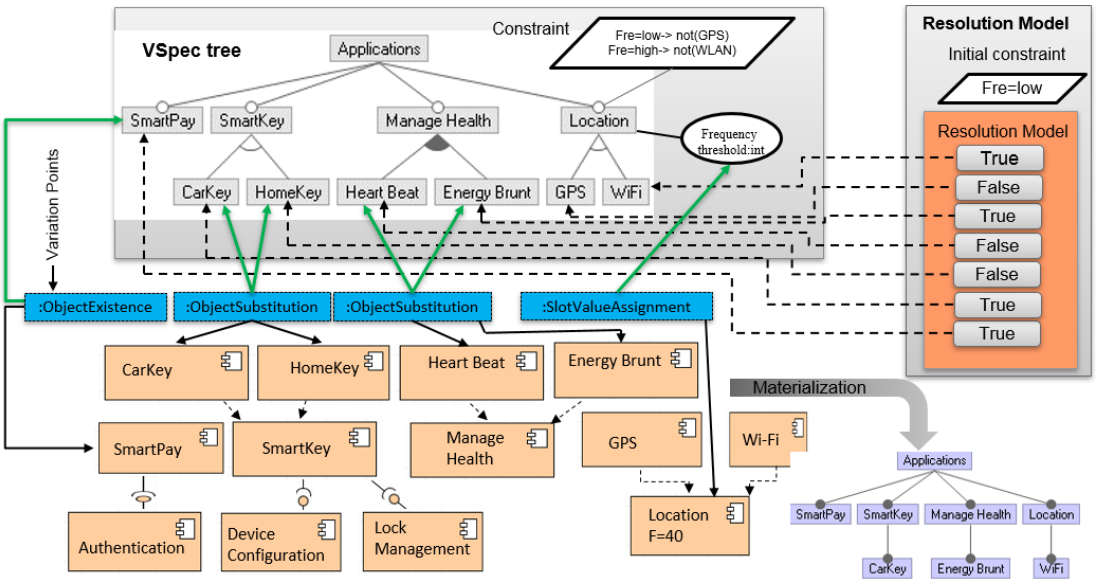

Fig. 4. Variability model(top left), resolution model(top right), variation points(center), base model(bottom left) and realization or resolved model(bottom right)

The variation points connect VSpecs to the elements of the base model influenced by the variability. The variation points also tell us what elements of the variability model are added, modified or removed. We have mentioned three variation points that lead to the change in the base model. These are *ObjectExistence*, *ObjectSubstitution*,and *Value assignment* (**Fig. 4** and **Fig. 12**). The OCL put a constraint on the elements of VSpec tree, e.g., *X* implies *Y* tells that if *X* is realized to true, *Y* must also be realized to true. For instance, the *location* feature in VSpec tree (**Fig. 4**) have a frequency as a constraint on it. The *location* feature is responsible to provide a location for smartwatch user for tracking her/his position, which can be realized either by *GPS* or *WiFi* variant. Tracking by *GPS* is more precise but costly because it consumes more battery of smartwatch device. Therefore, if the location by *GPS* is decided to be false, it will be removed from the configuration.

2.3 Reference Architecture

There are a number of motivations to evolve SPL systems including a change in the system context, a change in system resource, or emerging requirements from its users. Therefore, the system has to evolve to reflect those changes. Users' emerging demands over time are the major motivations of evolution in our study. After elicitation of user requirements (from SNS) and feature modeling, we design a reference architecture to reflect elicited requirements.

Our proposed approach presents a methodology to design a system in such a way that will evolve over time. **Fig. 5** shows an overview of our methodology. The variability model has to evolve in every iteration of the requirements elicitation step as shown in **Fig. 2** (❶). When the variability model is modified, then the changes are propagated into whole architecture. This phase takes the variability model as input and the result is a specific product architecture that comprises of required components for its adaptation.

Variability modeling takes place in domain engineering phase which has an important role in the production of specific products from SPL. In our approach, the below steps should be followed to construct the adaptive architecture.

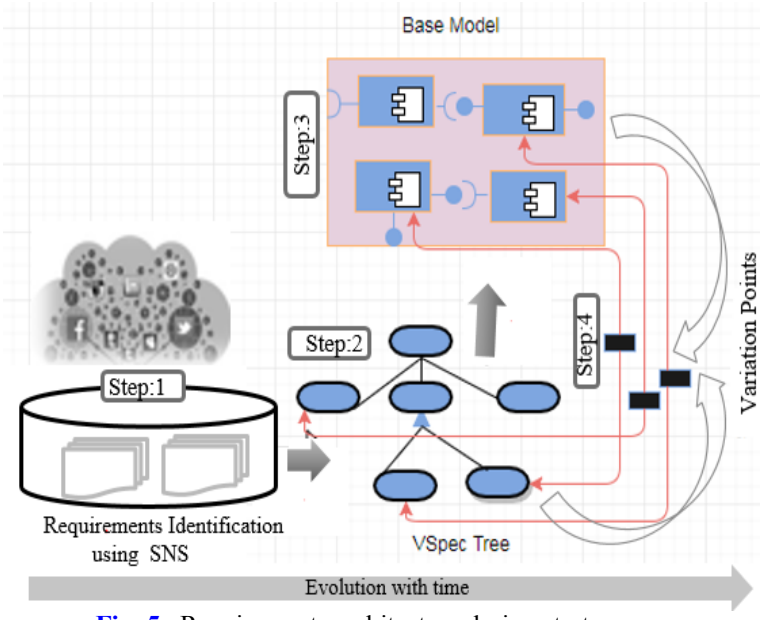

Fig. 5. Requirements architecture design strategy

Step 1: Identify user requirements from SNS corpus(phase ❶ in **Fig. 2**).

Step 2: Identify the VSpec and its constraint in elements of VSpec tree.

Step 3: Constructing a base model

Step 4: Realizing the VSpec tree and base model with the help of VP.

All steps are interrelated with one another. The first step provides a foundation for the second step and the third step takes the result of the second step as input to specify the base model. Once the VSpec tree and base model are identified, then the VSpec tree is realized with the base model with the help of VP. Phase ❸ in **Fig. 2** includes an example of reference architecture while **Fig. 12** shows the architecture evolution of smartwatch case study where two features *Language* and *Speaker* are added as new requirements. **Fig. 4** shows an example of architecture designing process where feature model, base model, resolution model, variation points, and resolved model are illustrated.

4. Case Study

To validate our proposed approach, we carried out a case study on the smartwatch domain. The smartwatch has become part of the life of millions of users around the world. The smartwatch domain was taken as a general example to know the general requirements instead of a specific product. Facebook and Twitter are selected as platforms to conduct our case study. The English of SNS were used in our case study. The guidelines of Runeson and Easterbrook [20, 21] are followed to conduct this case study.

3.1 Research Questions

The main objective of this study is to help software requirements engineers to hear directly from their end-users, and finally, remain ahead of the highly competitive and volatile market. Based on these assumptions, the following research questions are formulated to evaluate our proposed methodology.

RQ1: What type of user opinions exits in terms of bugs reports, new feature demands, and quality attributes?

RQ2: How ML algorithms help to extract user requirements?

RQ3: What challenges are faced by requirements engineers to elicit user requirements using SNS such as Facebook and Twitter?

3.2 Research Method

The research method of our study is consisting of four phases: data collection, data preprocessing, data analysis phase, and architecture design phase.

3.2.1 Data collection

We use Twitter's search API to collect the tweets for Pebble smartwatch. We also used two Facebook pages (Watch-wearables and Smartwatch) to collect the user comments about Pebble smartwatch. Facebook also provides open API to collect the user comments from a specified Facebook page. The Twitter API was customized to search for a particular term or hashtag (#) in Twitter feeds. The hashtag (#) allows users to search for desired terms explicitly to extract a list of opinions about a specific topic of interest. Based on the hashtag, the opinions of users are mined at a large scale to infer the discussions of public towards a specific topic, e.g. learning public opinion for a public figure or recent event [22]. Similarly, Facebook comments are also analyzed to know the sentiment of people on a specific topic or to know the user opinion for a specific product [6]. At the end, we collected 30633 tweets from Twitter and 18482 comments from Facebook. As a result, a dataset of 49115 unique opinions were gathered. We collected users' opinions from Facebook and Twitter from 21 December 2018 to 20 January 2019. **Fig. 6** shows the number of collected opinions per day over the process of data collection.

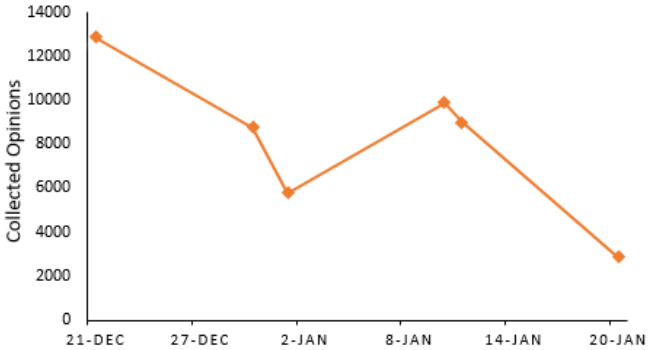

Fig. 6. Number of opinions collected per day

3.2.2 Data Preprocessing

The collected data both from Twitter and Facebook was noisy. Therefore, we applied some NLP techniques to clean the dataset. First of all, we tokenized all the opinions to get the desired keywords. After tokenization, we converted all the opinions into lower case, then n-gram(unigram, bigram, and trigram) were extracted (for classification). We also removed stopwords and applied stemming on our dataset to obtain a cleaned dataset.

3.2.3 Data Analysis

Based on the preprocessed data, we apply some ML algorithms to see up to what extent ML techniques classify the user requirements in terms of new feature demands, bug fixing request, or some non-functional requirement. In this section, we first model the topics in our dataset to know the overall discussion of users. For this purpose, along with topic modeling, we use the Wordcloud to know the most discussed keywords in the users' discussion. Then we extract the sentiment of our collected corpus to know the overall sentiment of users. The negative sentiment of users will help software developers look into the matter to address the issues to fulfill the user demands. Lastly, we classify the opinions in our dataset by using some popular ML algorithms.

A. Topic Modeling

Topic modeling is an ML statistical model to discover the abstract topics that exist in a collection of documents [39]. It frequently used in text mining to discover hidden semantic relation in a text document given that the document is about a specific domain [40]. Topic modeling is the first phase of our opinion analysis to discover user requirements. A summary of topics from the whole document can be a compact description that includes the main theme of collected opinions related to a specific topic. In this section, we used two NLP techniques to get the insights of discussed topics in our dataset. We used jsLDA [41] tool to model the related topics for smartwatch domain. The jsLDA is a tool that makes the running topic model easy with a modern web browser. It also demonstrates the potential of statistical computing in JavaScript. With standard stopwords, we developed our own stopword list to increase the accuracy of the model. We trained our dataset for 100 topics with 401 iterations. **Fig. 7** shows the result of our topic modeling with jsLDA.

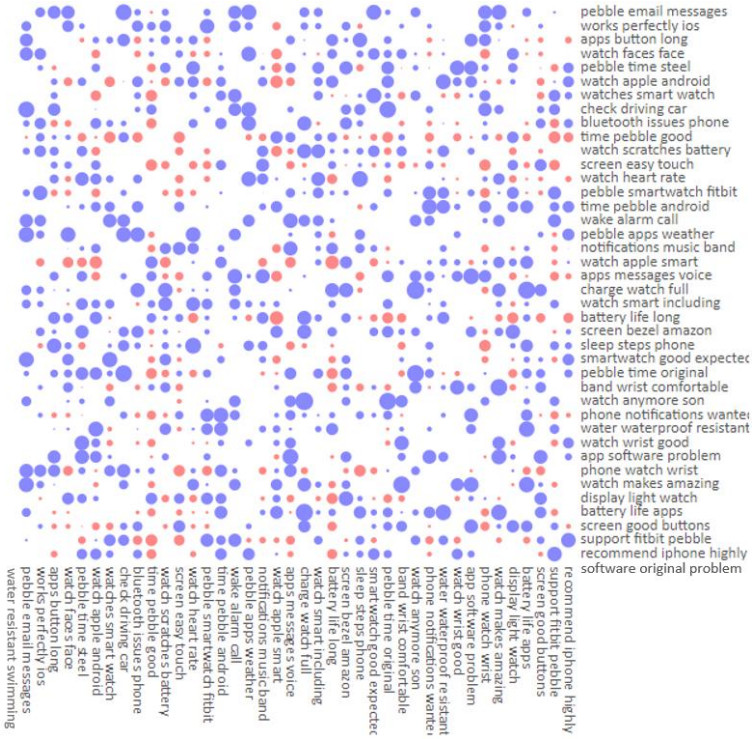

Fig. 7. Topic modeling for smartwatch domain

When a topic combination appeared more frequently than the model expected, then the circles are appeared as blue. The circles become large when the frequency increases. For instance "*software original problem/buttons touch interface*" are correlated and marked as a blue circle. It means that whenever the user discusses a software problem, they discuss the touch button of smartwatch and it is more frequent. While, when topics occur together less than model expected are marked as red. For instance, *"water resistant swimming/ app weather"* has a correlation. It means that, when user talk about the weather (probably hot), they talk about the watch to be a water-resistant one. The model was not expected this correlation but it has predicted. These correlations help software developers consider user scenarios while designing their products. Therefore, topic modeling can significantly contribute to elicit user requirements and software development organization can reflect these requirements in the next release of their product.

Along with correlation analysis of topics, we analyzed the summaries of topics with jsLDA and visualized result with the help of excel pivot charts. **Fig. 8** shows the result of our topic summary analysis. The x-axis shows a list of topics while the y-axis shows a number of occurrences of the topic in the dataset. The user demands and their bad experiences can be clearly seen in **Fig. 8**. For instance *"connection problem", "messaging issue", "music control app stoped"* clearly show users intent which can be easily interpreted as requirements.

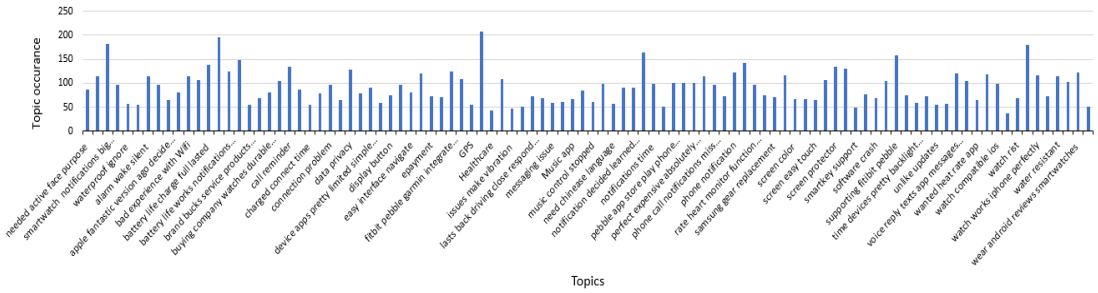

Fig. 8. Example of Topic summary

Before classification, we also used some other visualizations to our dataset to see the most discussed keywords in our dataset, the wordcloud is a good choice to do so. The wordcloud displays list of words regarding their frequency of occurrence. We used python wordcloud library for its implementation. **Fig. 9** shows an example of wordcloud in our case study. The figure clearly shows the most discussed topic is a smartwatch, connection, battery, and notification etc. **Table 1** shows an example of user opinions related to a smartwatch, connection, and battery in keywords in our dataset.

Fig. 9. wordcloud example of smartwatch domain

Table 1. Example of opinions related to a smartwatch, connection and battery keywords

Opinions related "*smartwatch*"	Opinions related "*connection*"	Opinions related "*battery*"
"I'm buying a smartwatch, and watches need to look good"	"I have been using this watch for a while and was reasonably happy with it until suddenly stopped being able to connect to my Android phone"	"outstanding battery life"
" great watch loved it"	"Another unexpected feature is the distance of the Bluetooth connection"	"I am going to let the battery drain down to pretty much nothing to see just how much battery life it has"
" ugly and expensive"	" no problem with connecting iphone 5"	" battery is the real problem"

B. Classification

Classification is the study of identifying to which of a set of categories a new observation belongs based on a trained set of data that includes observations whose classification is known [48]. In this section, we classify users' opinions into various categories by applying some famous ML classification algorithms. This section answers our research question **RQ1** and **RQ2**. First of all, we manually analyzed a sample of 600 users' opinions to know what information exists relevant to user requirements, which can be beneficial for software development organizations for their software evolution. We used [28] as a basis to categorize the user opinions. We enhanced the list of categories by adding new categories by examining 600 opinions (tweets and comments) from our dataset. Additionally, we assume that the opinions that expressed on the account of smartwatch domain can be categorized into relevant or irrelevant opinions.

A group of five postgraduate students of computer science with a research interest of requirements engineering are selected for our manual classifications. We were providing the sampled opinions and leading the process of classification. Majority voting was carried out to resolve the conflicts. Conflicts were reported while analyzing opinions which convey dual meaning. For instance, "*the notifications from my watch disappears, can anyone tell me why it happens? anyway, need a better solution*" can be classified as a bug report (*notification disappears*) or new feature demand (*anyway need better solution*). In our whole manual classification process, we dealt with 210 conflicts. **Table 2** shows the findings of our manual classification. The manual classification of opinions is briefly described as below:

Bug Reports: The opinions which are classified as bug reports potentially report user bad experience with either smartwatch design (hardware and interface), provided applications, or operating system. For instance "*when I use animations or scroll layers my application crashes*" reports a bug regarding animations and scrolling. Similarly " *my touch does not work well in winters, I am pissed off with it*" reports a bug regarding the touch screen of a smartwatch in some specified scenario while "*I am satisfied with the purchase but the one complaint I have is that the heart rate monitor gets a crazy high number during long runs. It seems to work well up to about 145 bpm and then if my real rate goes above that the heart rate falsely goes up to about 170 bpm*" reports a bug in heartbeat application provided my Pebble smartwatch.

New feature demands: The opinions in this category usually request for a new feature in the system. The users demand new feature and share innovative ideas or express their dissatisfaction for a specific feature. For instance *" I have a niece in college who is blind and I wish pebble would provide audio notification for calls and messages*" demands new feature

for blind people in pebble smartwatch. This kind of opinions can help to evolve the software systems and software development organizations can easily plan their next release. This also let software development organizations which feature they have to add, correct, enhance, or omit from the system.

Table 2. Number of bug reports, new feature demands, quality attribute, and irrelevant opinions collected from SNS

Domain	Bug Reports	New Feature Demands	Quality Attributes	Irrelevant
Smartwatch Design	45	67	23	251
Smartwatch App	89	123	46	176
Smartwatch OS	113	34	18	161

Quality attributes: In a software system, functional requirements describe the intended actions of the system, while non-functional requirements define the overall behavior and constraints of the system. Early identification of non-functional requirements can reduce software development cost and time while boosting user satisfaction [29, 30]. We classified all those opinions into quality attributes which tell us about software constraints, hardware constraints or talk about quality characteristics and sub-characteristics defined by ISO 9126 [31]. For instance "*did not sync with my phone. I had tried every possibility and followed instructions on pebble's forums*" talk about portability issue of pebble smartwatch. Similarly, "*my watch died on me after less than two weeks (horizontal lines of death) which after spending a few weeks on this sub I've come to realize is a common way for the watch to die"* talks about display and reliability of pebble smartwatch. Another user says somewhere in our dataset " *according to bbc report, EU recalled children's smartwatch over data security concerns"*. This opinion clearly talks about the data security of children's smartwatch. All these kinds of opinions were classified as quality attributes in smartwatch domain.

Irrelevant: A significant number of opinions in our sample dataset were classified as irrelevant because they did not convey any information about the requirements of the smartwatch. These opinions might include advertisements, general praise, news, or general information. For instance " *glad to see the new version of pebble OS*" and "*thank you pebble for providing quick charging*" although praise the product which can be used in sentiment analysis but at this stage, we classified these opinions as irrelevant opinions.

Fig. 10 shows a summary of our manual analysis. The result of our manual classification shows that out of 600 opinions analyzed, 53% opinions were relevant (bug reports, new feature demands, and quality attributes) while 47% opinions were classified as irrelevant opinions. These investigations answer our **RQ1**. The user opinions contain meaningful information for software developers including bug reports, new feature demands, and quality attributes. A significant amount of noisy text also includes in user opinions which was hard to categorize and named as an irrelevant category.

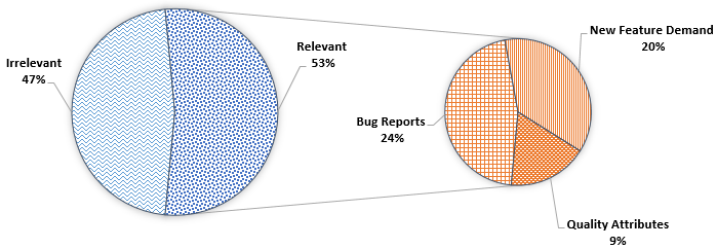

Fig. 10. Result of our manual classification for opinions

Classifiers: To address our **RQ2**, first we analyzed some ML classifiers such as logistic regression, Support Vector Machine (SVM), Multinomial Naive Bayes (NB), Random Forest (RF), and AdaBoost to select the classifiers that are most effective in our scenario. Secondly, what classification features produce the most suitable results in our opinions classification? Therefore, the primary screening was carried out to select the most effective classifiers in the context of Facebook comments and Twitter data [33, 45]. After a primary screening, we selected Multinomial NB, RF, and SVM classifier due to their promising results in text classification. Before moving forward, a brief introduction of selected ML classifiers is necessary for readers' understanding.

- **Support Vector Machine(SVM):** It is a supervised ML algorithm introduced by [25] which addresses two-group classification problems. Taking advantage of handling large feature, SVM works well for text classification. SVM has proved its importance by getting promising results on short text analysis in previous literature [26, 27]. It finds a line in multidimensional space that classifies classes.
- **Multinomial NB:** It is a probabilistic classifier of supervised ML with strong independent assumptions between the features [32]. It implements the NB algorithm for distributed data. In multinomial NB, data is typically represented by a word vector counts. It calculates likelihood to be a count of a token or a word. The decision of using multinomial NB over NB was based on [33] and its promising results in [34, 35].
- **Random Forest (RF):** It is a supervised ML classifier which creates a first and makes it random. It can be used for both regression and classification problems. It has already proved its need by producing promising results in text classification [36, 37]. In order to understand RF, we need to understand the decision tree, which is the foundation of an RF.

Classification Features: Selection of features in classification enables ML algorithms to train faster, reduce model complexity, and improves the accuracy of the model. It also reduces model overfitting. Therefore, to achieve more accurate results, we are using a combination of text features such as the content of the text itself (DBOW), text preprocessing, and sentiment analysis.

- **Opinion Content:** The main goal of our automatic classification is the words of opinions. The Distributed Bag-of-Words (DBOW) is simplifying text representation in NLP. In this model, the text is presented as a bag (multiset) of its words, irrespective of grammar and even word order but keeping multiplicity.
- **Text preprocessing:** In this feature, text reduction techniques of NLP are applied such as stop-word (SW) removal, stemming (ST). Stop-word removal is an NLP technique by which those words are removed from the text which is considered too generic for instance, *will*, *in*, *the*, *shall*, etc. The stemming is an NLP data preprocessing technique to obtain the root form of a word. This technique reduces the number of features in the text because only one form of a word appears when applied.
- **Sentiment Extraction (SE):** Sentiment extraction has proved its importance in the field of data science where it is used to correlate with users' opinions regarding a specific topic [22, 23]. In our study, we consider that a negative sentiment might reflect a user's a bad experience with the system or bad experience with a particular feature of a system. It might also show the feature request, a bug report in case of a user's a bad experience. Similarly, a positive sentiment might tell a good experience of a user or just users' satisfaction [24].

C. Evaluation

For the implementation of SVM, Multinomial NB, and RF, we use the scikit-learn library of python [38]. We used our truthset of 600 opinions which were manually analyzed for opinion classification. We trained and tested our classifiers by applying 10-fold cross-validation. This creates 10 different partitions of the dataset that takes 90% of the instances as a training set and 10% as an evaluation set for each partition. Additionally, we compared all three classifiers with each other to know their performance. We used precision, recall, and F-measure to evaluate the performance of different classifiers with different classification features. **Table 3** shows a summary of opinion classification accuracy in terms of precision, recall, and F-measure.

On average, all classifiers were able to get competitive results. The performance is based on the dimensionality of our dataset. The Twitter messages were short in size but the comments from Facebook were a little long that made the feature space (number of words) very large. The users use informal language in their tweets and comment that also drastically increased the number of features that classifier has to process. Therefore, the selected features made a difference in the result.

The sentiment did not affect the performance of classifier as was expected, because unlike political tweets or Facebook comments which tend to be more polarized as compared to the opinions collected for smartwatch domain. **Fig. 11** shows the sentiment score for different categories of opinions. The figure shows new feature demand tend to be slightly positive (+1), while bug reports and quality attributes also known as non-functional requirements (NFRs) tend to be slightly negative (-1). As we see, the difference was not significant that would affect the classification accuracy. The opinions which were classified as others got a neutral state (+1, -1).

The sentiment extraction phase did not contribute a significant step to get user requirements but we were able to see the overall mode of users for a specific product. We see that the negative sentiment was slightly useful to get user requirements. For instance, " *ahh, the poor battery of this watch sucks ! we need a long battery*" helped to get to know the problem about battery life. We have seen that the only sentiment cannot determine what users really want, but when it is used with other NLP techniques such as classification and topic modeling, can give more insights about the user opinions.

Table 3. Classification Result

	Bug Reports			New Feature Demands			Quality Attributes		
Combinations	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
RF:	0.78	0.57	0.66	0.77	0.58	0.66	0.61	0.62	0.61
RF+ ST	0.72	0.81	0.76	0.73	0.79	0.76	0.58	0.59	0.58
RF+ SW+ST	0.69	0.74	0.71	0.67	0.74	0.70	0.57	0.59	0.58
RF+SE	0.73	0.75	0.73	0.69	0.73	0.71	0.65	0.61	0.63
NB:	0.71	0.77	0.74	0.76	0.57	0.65	0.70	0.68	0.69
NB+ ST	0.70	0.79	0.74	0.74	0.58	0.65	0.69	0.71	0.70
NB+ SW+ST	0.67	0.74	0.70	0.72	0.55	0.62	0.63	0.69	0.66
NB+SE	0.74	0.77	0.75	0.77	0.55	0.64	0.68	0.70	0.69
SVM:	0.77	0.74	0.75	0.71	0.59	0.64	0.71	0.69	0.70
SVM+ ST	0.76	0.78	0.77	0.72	0.60	0.65	0.73	0.69	0.71
SVM+ SW+ST	0.75	0.67	0.71	0.69	0.58	0.63	0.71	0.62	0.66
SVM+SE	0.77	0.75	0.76	0.73	0.61	0.66	0.70	0.58	0.63

P: Precision, R: Recall, and F: F-measure

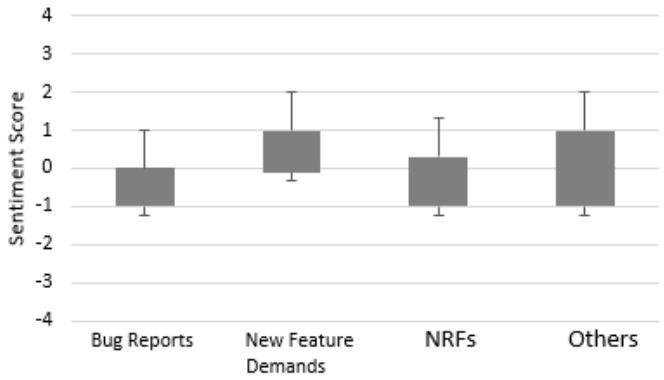

Fig. 11. Sentiment score for bug reports, new feature demands, and quality attributes

As a result, this investigation answered our **RQ2**. For **RQ2**, we concluded that opinions include very useful information that can lead to evolving software products. Because the opinions include bugs reports, new feature demands, and quality attributes which shows a piece of significant rich information for software developers. Our investigation confirms that our used classifiers are most suitable and robust for short text mining [46].

3.2.4 Reference Architecting

In this section, we show the evolution of smartwatch product as a result of our requirements elicitation from SNS. The early and quickly elicitation of requirements for the evolution of software is the main goal of this study. Therefore, our focus was mainly on requirements-driven evolution of software product lines. In this section, we provide an abstract view to designing reference architecture using our proposed architecture design strategy.

The requirements are identified as a result of our data analysis phase of the case study. At the end of our data analysis phase, we elicited 1430 user requirements. The requirements are then modeled through VSpec tree and architecture is designed. We used our architecture designing strategy mentioned in **Fig. 5** to design reference architecture. **Fig. 12** shows an example of our implementation where VSpec tree (FM 1) of the smartwatch is evolved (FM 2) by identifying new requirements. As mentioned in section 2.3, our architecture strategy has four basic steps. The first step was to elicit user requirements from SNS which easily saves time at requirements elicitation phase and as a result, products are evolved quickly [42]. For requirements elicitation step, we have applied ML algorithms and identified user requirements. After identifying user requirements, the VSpec tree is drawn manually. Based on the requirements elicited in the first step, the constraints are identified. An example of a constraint is explained in **Fig. 4**. In step 3, we constructed the base model. The base model does not contain any variability information. Lastly, variation points are defined to realize them with VSpec tree and elements in the base model. This step can be manual. As a result, the reference architecture is built. To generate different applications on the basis of the evolved reference architecture, a number of techniques [43, 44] can be applied. Our aim is to evolve a reference architecture which can impact the whole family of the products of the product lines.

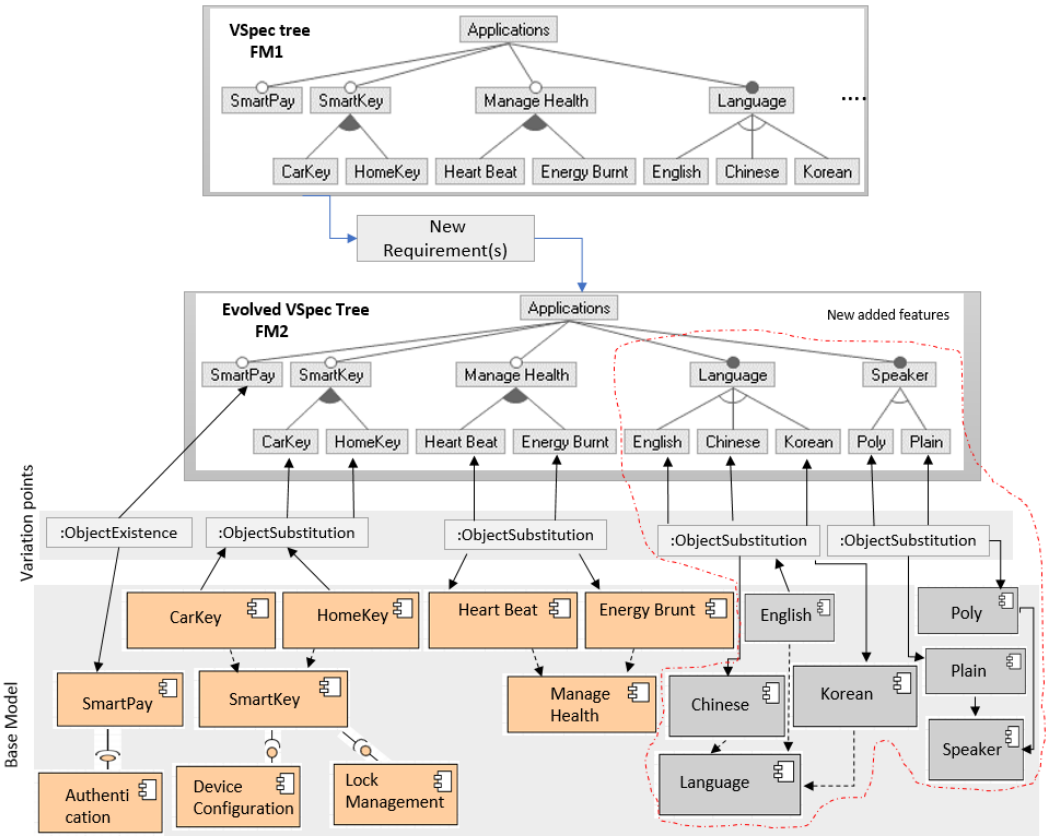

Fig. 12. Architecture evolution based on our approach

3.3 Overall Discussion

This section discusses the result of our evaluation and discusses the challenges faced during the elicitation of user requirements by using SNS. Developers, requirements engineers and SPL engineers can use the outcomes of classification, topic modeling, and sentiment analysis. The topic modeling gives collections of words that convey user demands together. When we used jsLDA for topic modeling, we were not expecting such promising results initially, but when we trained our dataset we saw a piece of significant and useful information that can be used by software developers or software development organization. For instance, "*software original problem/buttons touch interface*" was extracted as correlated terms. This means that whenever the users discuss about a software problem, they discuss on the touch button of smartwatch and it is more frequent. This type of information is very significant for requirements engineers. Our data was consisting of tweets and comments, the tweets are very short text while comments were long. The whole dataset was a mixture of both tweets and comments. The jsLDA would perform better if our data would be bigger. Meaning that the extra topic layer does not add anything to the classification when we work on a smaller document. If we have really short documents, like tweets, it is very hard to break the document into topics. As a result, it affects the prediction results. We learned that bigger size of the dataset, more predictions in topic modeling.

The classification helped us to identify opinions that contain bug reports, new feature demands, and quality attributes. These opinions are useful for software development organizations, as these opinions contain very useful information to elicit user requirements or

to identify a big to be fixed or to identify a non-functional requirement. For instance, *" if your watch does allow you to enjoy swimming with it, then it is wastage of money! Need water resistant watch"* report a designing issue smartwatch. The user demands a watch with water resistant feature. Similarly, "*when i received text messages, my watch stucks and requires a restart to work !"* reports a bug in a smartwatch. Classifying bug reports and new feature demands were slightly easy than classifying quality attributes. For instance, *" when I click on the main button it responds late, ridiculous, it should respond within a fraction of seconds"* implicitly express the usability requirements (quality attribute). In such type of opinions, it was difficult for our classifiers to predict the quality attributes. This is because the classifier was limited to predict quality attributes for which it has been trained. We only trained our dataset to classify those opinions which explicitly talk about the quality attributes. For instance, *"according to bbc report, EU recalled children's smartwatch over data security concerns*" explicitly talk about data security and it can be easily classified as quality attribute class. As mentioned in [49], identifying NFRs through automated classification is a difficult task. We also learned that a semi-automated methodology is required to fully identify all types of NFRs from raw text like tweets and comments. Summarizing the discussion, we faced a number of challenges including the classification of quality attributes. It was difficult to classify all possible quality attributes from our dataset. This is because indicator terms were trained to classify the quality attributes. Although, our approach identifies the quality attributes it still requires domain analysts to evaluate the correctness of identified quality attributes or NRFs. Finally, this section answered our **RQ3** in detail by mentioning the problems faced during our whole process of identifying requirements from SNS.

5. Threats to Validity

We first analyzed opinions manually to make their categories. This manual analysis was based on the human judgment which is an error-prone process because human bias can affect in deciding if an opinion falls within a particular category or related to a particular domain. To cope with these challenges, we conducted this process by involving well-trained five postgraduate students. Additionally, we made a clear definition of defined categories and circulated this document to make sure the understandability of definition and concept. The conflicts were resolved by a majority voting scheme.

This study was conducted to elicit requirements for smartwatch design and application domain. We extract user's opinions and discussions without considering some special events that could affect the user opinions, such as new product release or a new product recall. Furthermore, for our automated classification, we relied on our manual classification. However, conducting manual classification on the whole dataset is unrealistic. Therefore, we used a sample of opinions for manual classification. The address generalizability threats, we selected the sample opinions using random sampling.

The proposed approach depends on the quality of unstructured data, such as tweets and comments. Particularly, if the opinions contain many slangs and meaningless words, then it would be difficult to annotate comments or tweets to a particular category. This is an inevitable part of this study, but it can be mitigated by applying comprehensive NLP techniques.

6. Conclusion

This paper presents a process that supports the rapid evolution of SPL products by eliciting requirements from popular social network sites. To elicit requirements from SNS, we applied classification, topic modeling, and sentiment analysis. All outcomes from these steps were meaningful and useful for requirements engineering and software development organizations. For classification, we selected SVM, RF, and multinomial NB classifiers. We selected three features as classification features for classifications. These include bag-of-word, sentiment extraction, and stopword removal. We observed that all classifiers produce almost similar results. The result for quality attribute class was a little disappointing due to some reasons, mentioned in the discussion section.

Additionally, we have presented an adaptive architecture designing strategy to reflect the new requirements in the system quickly. It is very useful to quickly reflect user and market needs, and to keep up-to-date reference architecture to quickly develop an SPL-based product family.

In the future, we want to investigate our architecture designing strategy in detail for dynamically reflecting emerging requirements in SPL. Although this future research on dynamic evolution of SPL architecture may produce diverse architecture models with variabilities, we believe that it will be suitable for SPL engineering of developing a series of product families.

References

[1] Linda M. Northrop and Paul C. Clements, “A Framework for Software Product Line Practice version 5.0” 2010, http://www.sei.cmu.edu/productlines/framework.html

[2] W. Maalej, M. Nayebi, T. Johann and G. Ruhe, "Toward Data-Driven Requirements Engineering," *IEEE Software*, vol. 33, no. 1, pp. 48-54, 2016. Article (CrossRef Link)

[3] Svahnberg, Mikael, and Jan Bosch, “Evolution in software product lines: two cases,” *J. Software Maintenance and Evolution: Research and Practice*, vol. 11, no. 6, pp. 391–422, 1999. Article (CrossRef Link)

[4] Hotz, L. et al., “Configuration in Industrial Product Families,” *The ConIPF Methodology. IOS Press*, p. 296, 2006. Article (CrossRef Link)

[5] Meyer, Sava, “Understanding Software Adaptation and Evolution," pp.1-20, 2015. Article (CrossRef Link)

[6] N. Ali, S. Kim and J. Hong, "Listen closely, respond quickly: Enhancing conformity of SPL domain requirements through SNS," in *Proc. of Int. Conf. on Information Science and Communications Technologies (ICISCT),* pp. 1-5, 2016. Article (CrossRef Link)

[7] N Ali, N., & Hong, J. E., “Creating adaptive software architecture dynamically for recurring new requirements,” in *Proc. of Open Source Systems & Technologies (ICOSST), International Conference on pp.* 67-72, 2017. Article (CrossRef Link)

[8] Berger, Thorsten, et al., “A survey of variability modeling in industrial practice,” in *Proc. of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, pp.7, 2013.

[9] OMG. Common Variability Language (CVL). OMG Revised Submission, 2012

[10] Schmid, Klaus, and Holger Eichelberger, "A requirements-based taxonomy of software product line evolution," *Electronic Communications of the EASST*, vol. 8, pp.1-13, 2007. Article (CrossRef Link)

[11] Guzman, Emitza, Mohamed Ibrahim, and Martin Glinz, "A little bird told me: mining tweets for requirements and software evolution," in *Proc. of IEEE 25th Int. Requirements Engineering Conference (RE). IEEE*, pp. 11-20, 2017. Article (CrossRef Link)

[12] Galvis Carreño, Laura V., and Kristina Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *Proc. of the Int. Conf. on Software Engineering. IEEE Press*, pp. 582-591, 2013. Article (Cross Ref Link)
[13] Huynh, Ngoc-Tho, Maria-Teresa Segarra, and Antoine Beugnard, "A development process based on variability modeling for building adaptive software architectures," *Computer Science and Information Systems (FedCSIS)*, vol. 8, pp. 1715-1718, 2016. Article (CrossRef Link)
[14] Jiang, Wei, Haibin Ruan, and Li Zhang, "Analysis of economic impact of online reviews: an approach for market-driven requirements evolution," *Requirements Engineering. Springer, Berlin, Heidelberg*, pp. 45-59, 2014. Article (CrossRef Link)
[15] I. Morales-Ramirez, K. Fitsum Meshesha, and P. Anna, "Analysis of online discussions in support of requirements discovery," in *Proc. of International Conference on Advanced Information Systems Engineering. Springer, Cham*, pp. 159-174, 2017. Article (CrossRef Link)
[16] W. Jiang, H. Ruan, L.Zhang, P. Lew, and J. Jiang, "For user-driven software evolution: requirements elicitation derived from mining online reviews," in *Proc. of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 584-595, 2014. Article (CrossRef Link)
[17] M. Oriol et al., "FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring," in *Proc. of 26th International Requirements Engineering Conference (RE), Banff, AB*, pp. 217-227, 2018. Article (CrossRef Link)
[18] NH. Bakar, ZM. Kasirun, N .Salleh, HA .Jalab, "Extracting features from online software reviews to aid requirements reuse," *Applied Soft Computing*, vol. 49, pp. 1297-1315, 2016. Article (CrossRef Link)
[19] VE. Souza, A. Lapouchnian, K. Angelopoulos, J. Mylopoulos, "Requirements-driven software evolution," *Computer Sci-Research and Development*, vol. 28, no. 4, pp.311-29, 2013. Article (CrossRef Link)
[20] P. Runeson, M. Host, A. Rainer, B. Regnell, "Case study research in software engineering: Guidelines and examples," *John Wiley & Sons*, Mar 7, 2012. Article (CrossRef Link)
[21] S. Easterbrook and J. Aranda, "Case studies for software engineers," in *Proc. of 26th International Conference on Software Engineering*, pp. 736 – 738, 2004. Article (CrossRef Link)
[22] O'Connor, Brendan, et al., "From tweets to polls: Linking text sentiment to public opinion time series," in *Proc. of Fourth International AAAI Conference on Weblogs and Social Media*, pp. 122-129, 2010. Article (CrossRef Link)
[23] D. Ediger et al., "Massive Social Network Analysis: Mining Twitter for Social Good," in *Proc. of International Conference on Parallel Processing*, pp. 583-593, 2010. Article (CrossRef Link)
[24] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proc. of Int. Requirements Engineering Conference (RE)*, pp. 125-134, 2013. Article (CrossRef Link)
[25] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. Article (CrossRef Link)
[26] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: Sentiment classification using machine learning techniques," in *Proc. of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, pp. 79–86, 2002. Article (CrossRef Link)
[27] E. Martinez Camara et al., "Tecnicas de clasificacion de opiniones aplicadas a un corpus en espanol," *Procesamiento de Lenguaje Natural*, pp. 163–170, 2011. Article (CrossRef Link)
[28] E. Guzman, R. Alkadhi and N. Seyff, "A Needle in a Haystack: What Do Twitter Users Say about Software?," in *Proc. of 24th International Requirements Engineering Conference (RE), Beijing*, pp. 96-105, 2016. Article (CrossRef Link)
[29] A. Mahmoud, "An information theoretic approach for extracting and tracing non-functional requirements," in *Proc. of International Requirements Engineering Conference (RE), Ottawa*, pp. 36-45, 2015. Article (CrossRef Link)
[30] P. Eugenio et al., "A methodology for the classification of quality of requirements using machine learning techniques," *Information and Software Technology*, vol. 67, pp. 180-195, 2015. Article (CrossRef Link)
[31] ISO 9126 Software Quality Characteristics. http://www.sqa.net/iso9126.html
[32] Naive Bayes classifier. https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[33] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” in *Proc. of the Int. Conf. on Machine Learning*, pp. 161–168, 2006. Article (CrossRef Link)
[34] N. Chen, et al., “AR-Miner: Mining informative reviews for developers from mobile app marketplace,” in *Proc. of the International Conference on Software Engineering*, pp. 767–778, 2014. Article (CrossRef Link)
[35] A. Bacchelli, et al. “Content classification of development emails,” in *Proc. of the International Conference on Software Engineering Pages*, pp. 375–385, 2012. Article (CrossRef Link)
[36] Xu, Baoxun, et al., "An Improved Random Forest Classifier for Text Categorization," *JCP 7*, no. 12, pp. 2913-2920, 2012.
[37] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D Penta, “Release planning of mobile apps based on user reviews,” in *Proc. of the International Conference on Software Engineering*, pp. 14–24, 2016. Article (CrossRef Link)
[38] scikit-learn. https://scikit-learn.org/stable/
[39] Topic model. https://en.wikipedia.org/wiki/Topic_model
[40] Liu, Lin, Lin Tang, Wen Dong, Shaowen Yao, and Wei Zhou, "An overview of topic modeling and its current applications in bioinformatics," *SpringerPlus*, pp. 1608, 2016. Article (CrossRef Link)
[41] Mimno David, “jsLDA: In-browser topic modeling,”. Article (CrossRef Link)
[42] N. Ali, and JE Hong, "Using Social Network Service to determine the Initial User Requirements for Small Software Businesses," *International Journal of Applied Business and Economic Research*, vol 15, 2017. Article (CrossRef Link)
[43] Pascual, Gustavo G., Mónica Pinto, and Lidia Fuentes, "Self-adaptation of mobile systems driven by the common variability language," *Future Generation Computer Systems*, vol. 47, pp.127-144, 2015. Article (CrossRef Link)
[44] Huynh, Ngoc Tho, "A development process for building adaptative software architectures," *PhD diss., Ecole nationale supérieure Mines-Télécom Atlantique*, 2017. Article (CrossRef Link)
[45] Sebastiani, Fabrizio, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1-47, 20002. Article (CrossRef Link)
[46] S. Wang and C. Manning, “Baselines and bigrams: Simple, good sentiment and topic classification,” *Annual Meeting of the Association for Computational Linguistics*, vol. 2, pp. 90–94, 2012. Article (CrossRef Link)
[47] B. Goetz, and A. Pleuss, “Evolution of software product lines," *Evolving Software Systems. Springer, Berlin, Heidelberg*, pp. 265-295, 2014. Article (CrossRef Link)
[48] T. Jiliang, S. Alelyani, and H. Liu, “Feature selection for classification: A review," *Data classification: algorithms and applications*, pp. 1-33, 2014. Article (CrossRef Link)
[49] C. Agustin, D. Godoy, and M. Campo, “Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436-445, 2010. Article (CrossRef Link)
[50] E. Guzman, L. Oliveira, Y. Steiner, L. C. Wagner and M. Glinz, "User Feedback in the App Store: A Cross-Cultural Study," in *Proc. of International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS), Gothenburg*, pp. 13-22, 2018. Article (CrossRef Link)
[51] A. Perini, "Data-Driven Requirements Engineering. The SUPERSEDE Way," in *Proc. of Annual International Symposium on Information Management and Big Data*, pp. 13-18, 2018. Article (CrossRef Link)
[52] I. Morales-Ramirez, F. Meshesha Kifetew, and A. Perini, “Speech-acts based analysis for requirements discovery from online discussions," *Information Systems*, vol. 86, pp. 94-112, 2019. Article (CrossRef Link)

Nazakat Ali is a PhD student in Department of Computer Science, School of Electrical and Computer Engineering, Chungbuk National University, Korea. He received his MS in Computer Science from Chungbuk National University, Korea in 2017. His research interests include software requirements engineering, data mining, ontology, software architecture and Cyber-Physical Systems.

Sangwon Hwang is a research professor of Artificial Intelligence and BigData Medical Center at the Yonsei University Wonju College of Medicine, Korea. He received his Ph. D in computer science from Yonsei University, Korea, in 2014. His research interests include software engineering, data mining, ontology, code reuse, information extraction, machine learning, and artificial intelligence.

Jang-Eui Hong is a professor of Computer Science Department at the school of Electrical and Computer Engineering, Chungbuk National University, Cheongju, Korea. He received his Ph. D in computer science from KAIST, Korea, in 2001. He served as a research member at ADD (Agency for Defense Development) from 2000 to 2002, and also served as a principal consultant at Solution Link, Co., Ltd. His research interests include software quality, embedded software architecture, low-energy software model, and software process improvement.