Understanding Feature Requests by Leveraging Fuzzy Method and Linguistic Analysis

Lin Shi *‡, Celia Chen¶§, Qing Wang*†‡[1], Shoubin Li*‡, and Barry Boehm¶
*Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
†State Key Laboratory of Computer Sciences, Institute of Software Chinese Academy of Sciences, Beijing, China
‡University of Chinese Academy of Sciences, Beijing, China
{shilin,wq}@itechs.iscas.ac.cn, shoubin@iscas.ac.cn
§Department of Computer Science, Occidental College, Los Angeles, CA
¶Center for Systems and Software Engineering, University of Southern California, Los Angeles, USA
{qianqiac,boehm}@usc.edu

***Abstract*—In open software development environment, a large number of feature requests with mixed quality are often posted by stakeholders and usually managed in issue tracking systems. Thoroughly understanding and analyzing the real intents that feature requests imply is a labor-intensive and challenging task. In this paper, we introduce an approach to understand feature requests automatically. We generate a set of fuzzy rules based on natural language processing techniques that classify each sentence in feature requests into a set of categories: Intent, Explanation, Benefit, Drawback, Example and Trivia. Consequently, the feature requests can be automatically structured based on the classification results. We conduct experiments on 2,112 sentences taken from 602 feature requests of nine popular open source projects. The results show that our method can reach a high performance on classifying sentences from feature requests. Moreover, when applying fuzzy rules on machine learning methods, the performance can be improved significantly.**

I. Introduction

Online issue tracking systems such as Bugzilla [1], JIRA [2], and Github [3] are widely used in the open software development environment. These systems enable users to easily submit new feature requests in the format of issue reports and greatly improve the efficiency for organizations when gathering new ideas. In the software development process, these feature requests serve as critical inputs to software maintenance and evolution. Developers rely on them to figure out the high priority feature requests that should be implemented in the next release [4], to trace whether all requested features are implemented [5] [6], and to update documentation for implemented features [7] [8].

However, identifying all the feature requests from issue tracking systems is challenging. First, users tend to incorrectly differentiate feature requests from bug reports when posting. Herzig *et al.* [9] reported that rather than referring to feature requests, users are likely to inaccurately posted their desired features as bug reports in the issue tracking systems. Moreover, the process of identifying feature requests is workload-intensive and time consuming as it requires the system developers to understand and analyze a large amount of textual artifacts. For communities that have a large number of users, the number of new incoming issues can be massive on a daily basis. The voluntary and loose management nature of open source community makes this task even more impossible [10] to accomplish.

Existing literatures point out that collecting the reports in a structured manner could offset this burden [11], and several studies [12][13] have been proposed to structure the contents of bug reports into code samples, stack traces, *etc*. Unfortunately, they do not support structuring feature requests.

In this paper, we propose a method to automatically analyze and structure the contents of feature requests based on fuzzy rules and natural language processing techniques. The contents of feature requests are categorized, measured and classified. The classification results are used as the optimization objective to incrementally generate fuzzy rules. Our method is then validated through experiments conducted on nine popular open source projects to evaluate to what extent the generated rules can classify the sentences from feature requests correctly.

The experiment results show that the classification performance of fuzzy rules is increasing as new rules are introduced. The generated fuzzy rules are able to classify feature requests from the nine projects correctly. Additionally, after transforming the fuzzy rules into boolean variables, we apply the fuzzy rules onto various machine learning methods. The results show that these methods have significant improvement in their performances when using fuzzy rules as model variables.

The major contributions of this paper are as follows.

1) We propose an automated solution to structure feature requests that can facilitate deeper content mining on feature requests.
2) We introduce an incremental approach that combines fuzzy rules with natural language processing techniques.
3) We conduct an empirical evaluation of the generated fuzzy rules.
4) We provide a publicly available tool and dataset[1] to replicate our experiments.

The remainder of the paper is organized as follows. Section II elaborates the approach. Section III presents the experi-

[1]The corresponding author

[1]http://goo.gl/aWOvIX

978-1-5386-2684-9/17$15.00 © 2017 IEEE

ASE 2017, Urbana-Champaign, IL, USA
Technical Research

mental setup. Section IV describes the results and analysis. Section V discusses the implications and future work. Section VI shows the threats to validity. Section VII introduces the related work. Section VIII concludes our work.

II. Approach

In Section II-A and II-B, we introduce the definition of categories and heuristic linguistic patterns that are used in the proposed approach. Second, we illustrate the related definitions of fuzzy rules in Section II-C. Then we provide the process of how we generate fuzzy rules by leveraging fuzzy method and linguistic analysis in Section II-D.

A. Categories Definition of Sentences in Feature Requests

Categories for sentences in feature requests are initially identified from practice guidelines [14][15]. For example, we identify *benefit* and *drawback* from the practical guideline "Avoid being vague about the benefits or possible drawbacks". Then we manually inspect a sample of 488 sentences taken from the PMA and Hibernate projects. During this process, we notice that users are more likely to present ideas or actual needs to explain the reason why they request the feature. They also include references and supporting materials via hyper-links, and scenarios where the desired features will be used. Some users even provide possible solutions to implement the feature. Feature requests also contain trivial sentences such as "Looking forward to your reply". As a result, we define a list of six categories for sentences in feature requests. The categories include intent, explanation, benefit, drawback, example, and trivia. The definition of each category is stated in Table I along with its importance. We define the importance of a category as the level of attention needed when analyzing feature requests. We consider the category *Intent* as the most important and the category *Trivia* as the least important. These categories are defined to help structure feature requests and highlight contents that deserve more attention.

TABLE I
Definitions of Sentence Categories

Category	Importance	Definition
Intent	1	Descriptions about ideas, needs, or expectations to improve the system and its functionalities.
Benefit	2	Descriptions about good or helpful results or effects that the proposed feature will deliver.
Drawback	3	Descriptions of disadvantages or the negative parts of the current system behavior.
Example	4	Descriptions of examples or references in support of the proposed feature.
Explanation	5	Detailed information about the current behavior, scenarios, or solutions related to the proposed feature.
Trivia	6	Other information that are not related to the propose feature nor the system.

B. Definition of Heuristic Linguistic Patterns

Users are likely to use recurrent linguistic patterns in their sentences when requesting new features[16]. Given a textual sentence from a feature request, we can heuristically identify its linguistic patterns from three levels:

TABLE II
Examples of heuristic linguistic patterns and 18 rules from 81 generated fuzzy rules

ID	*Antecedents (Heuristic Linguistic Patterns)*	Level	C	CF
1	action_"propose" = 1	LEX, SYN	intent	81
2	action_"mean" = 1	LEX, SYN	explanation	80
3	start_"please"=1	LEX	intent	79
4	start_"unfortunately","actually" = 1	LEX	explanation	76
5	contain_"really"=1, question=1	LEX	explanation	74
6	{"hello", "thank", "regards", "look forward"}	LEX	trivia	65
7	{"would like", "wish for", "I need" }	LEX	intent	52
8	[SYS-NAME]+{"may","need","should"}	LEX	intent	51
9	first_sentence=1, start_VB=1	LEX,SYN	intent	41
10	vaild_words = 0, positive_good=0	LEX,SEM	trivia	40
11	start_"however"=1, contain_"only"=1	LEX	drawback	39
12	start_"however"=1, negative=1	LEX,SEM	drawback	38
13	start_"for example" = 1	LEX	example	36
14	{"save" | "reduce"} + {"memory" | "effort"} = 1	LEX	benefit	31
15	{"helpful", "useful", "convenient", "awesome"}	LEX	benefit	30
16	negative_good =1	SEM	drawback	27
17	positive_bad = 1	SEM	drawback	26
18	positive_good=1	SEM	benefit	20

1) Lexical Patterns (LEX): We define lexical patterns as the words or phrases that frequently appear in the sentences of a certain category. We construct ten glossaries to help us identify lexical patterns: (1) six glossaries for the six categories we defined; (2) two glossaries for "Good" and "Bad" words; (3) and two glossaries for "stop words" and "stop verbs". Table II lists some examples of the glossaries. Sentences that contain "would like", "wish", or "want" are more likely to be classified into class *Intent* (#7 in Table II). The valid_word=0 means that all the words in the sentence belong to the "stop words" glossary (#10 in Table II). The start_"please" means that the sentence starts with the word "please" (#3 in Table II). Sentences that contain the name of the system followed by the word "may","need", or "should" are more likely to be classified into class *Intent* (#8 in Table II).

2) Syntax Patterns (SYN): We define syntax patterns as the specific sentence structures that frequently appear in a certain category. After parsing sentences via Stanford NLP parser [17], we are able to obtain the sequence of Penn part-of-speech (POS) tags [18] and Stanford dependencies (SD) for each sentence. The sequence of POS tags and Stanford dependencies can be used to identify these patterns. For example, Rule #9 in Table II has a syntax pattern of "start_VB=1", which means that the first word in the sentence is a verb. We can also identify the subject and action of this sentence by analyzing "nsubj" and "cop" dependencies in SD. As shown in Table II, action_"propose"=1 means that the action of the sentence is "propose" and the sentence is describing an intent.

3) Semantic Patterns (SEM): We define semantic patterns as the meaning of linguistic expressions that frequently appear in the sentences of a certain category. More specifically, we are interested in the patterns of meaningfulness that we find in words. Taking the sentence "The frameset of phpmyadmin is not pretty." as an example, we can obtain its dependencies by using the Stanford NLP parser as shown in Figure 1.

The semantic pattern of this sentence is *negative_good*, which means the sentence describes something that is not good. This semantic pattern is identified by having the "*neg(pretty-7,not-6)*" dependency (7 and 6 represent the word positions). This dependency indicates that the sentence has the negation modifier on the adjective word "pretty" and the word "pretty" belongs to the "Good" glossary.

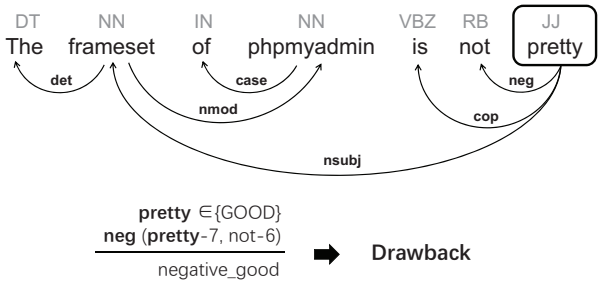

Fig. 1. An example of semantic pattern for drawback category.

C. Definition of Fuzzy Rules

1) Fuzzy Rules for Text Classification: Fuzzy rules, which are widely used in artificial intelligence and control systems to solve pattern classification problems, have also been used for text classification problems [19].

$$\begin{aligned} Rule\ R_k:\ IF\ x_1\ is\ A_{k1}\ AND\ ...\ AND\ x_n\ is\ A_{kn}, \\ THEN\ class = C_k\ with\ CF_k, \end{aligned} \tag{1}$$

where R_k is the k^{th} fuzzy rule, $x_1, ..., x_n$ are the linguistic variables, A_{ki} are the values used to the represent linguistic variables, C_k is a consequent class, and CF_k is a rule weight. For classifying an input text that satisfies multiple rules, the rule that has the maximum CF_k is the winner rule. Taken rules in Table II as an example, when Rule #1 and Rule #18 are both satisfied in one sentence, we classify the sentence to be "Intent" since Rule #1 has a higher *CF* value.

2) Confidence of Fuzzy Rules: We rank the rules according to the *confidence* measurement, which is defined as follows:

$$conf(A_k \Rightarrow C_k) = \frac{|S(A_k) \cap S(C_k)|}{|S(A_k)|} \tag{2}$$

where S is the training dataset, $S(A_k)$ is the subset of training dataset compatible with A_k, $S(C_k)$ is the subset of training dataset that are compatible with C_k.

3) Performance Measurements of Fuzzy Rules: We define two measurements to evaluate the performances of fuzzy rules in our cases, which are *correctness* and *misclassification*, Since one fuzzy rule can be viewed as a specific kind of association rule [20] of the form $A_k \Rightarrow C_k$, we measure the correctness of classification results by leveraging the commonly used measurements in machine learning methods: *precision*, *recall*, and *f_measure* [21]. We use the average weighted f-measure of all categories to indicate the correctness of the fuzzy rules. The $Correctness$ is defined as follows.

$$\begin{aligned} &Correctness(Rules) = \\ &\sum_{i=1}^{n}(\frac{max\{P(i)\} - P(i) + 1}{\sum P(i)} * f\_measure(i)) \\ &f\_measure(i) = \frac{2 * Precision(i) * Recall(i)}{Precision(i) + Recall(i)} \end{aligned} \tag{3}$$

where $P(i)$ is the set of importances of all categories.

We measure the misclassification of classification results for fuzzy rules by leveraging the *confusion matrix*, also known as error matrix. The confusion matrix is a specific table layout that allows visualization of the mislabeling one as another[22]. We use the average weighted error rate of all the categories to indicate the misclassification. Given the confusion matrix $C = (x_{ij})_{n \times n}$, where n is the number of categories, x_{ij} is the number of sentences that are classified as j but actually belongs to i. By using the importance difference between two category ($|P(i) - P(j)|$) as the weight, we define the $misclassification$ as follows.

$$\begin{aligned} &Misclassification(Rules) = \\ &\frac{1}{n}\sum_{i=1}^{n}(\frac{\sum_{j=1}^{n}(|P(i) - P(j)| \times x_{ij})}{\sum_{j=1}^{n}(|P(i) - P(j)| \times x_{ij}) + x_{ii}} \end{aligned} \tag{4}$$

D. The Process of Fuzzy Rules Generation

Figure 2 illustrates how we generate fuzzy rules. We divide the process into two phases: initial phase and incremental selection phase. In the initial phase, we aim to generate a set of initial fuzzy rules by heuristically identifying linguistic patterns, and define a reasonable set of CF_k for those rules. In the incremental selection phase, we aim to improve the initial fuzzy rules iteratively according to the incorrect classification results on dataset that includes new projects.

1) Initial Phase: We initiate the rule set by identifying the heuristic linguistic patterns (*identifyHLP* in Figure 2) from sentences taken from the first $project$. For each sentence in project P_1, we identify its lexical, syntactic, or semantic patterns in a heuristic way as introduced in Section II-B. These linguistic patterns can work as the antecedent $A_{k1}...A_{kn}$ for a rule r, and the C_k of rule r is the category that the sentence belongs to. Since categories are not equally important, we consider the rules with more important consequences to deserve higher weights. When rules have the same consequence, we consider the rules with higher confidences to have higher weights. Therefore, we rank the initial rules by their importances and confidences, and set the weight CF_k of each rule r according to its position in the rank. The output of the initial phase is a set of initial fuzzy rules R_0.

2) Incremental Selection Phase: In this phase, we assemble the dataset T_i by adding a new project at a time. Then we classify T_i by applying fuzzy rules R_{i-1}. By comparing with the ground truth labels in T_i, we obtain the correct and incorrect classification results. We consider the fuzzy rules fail to classify correctly for two reasons:

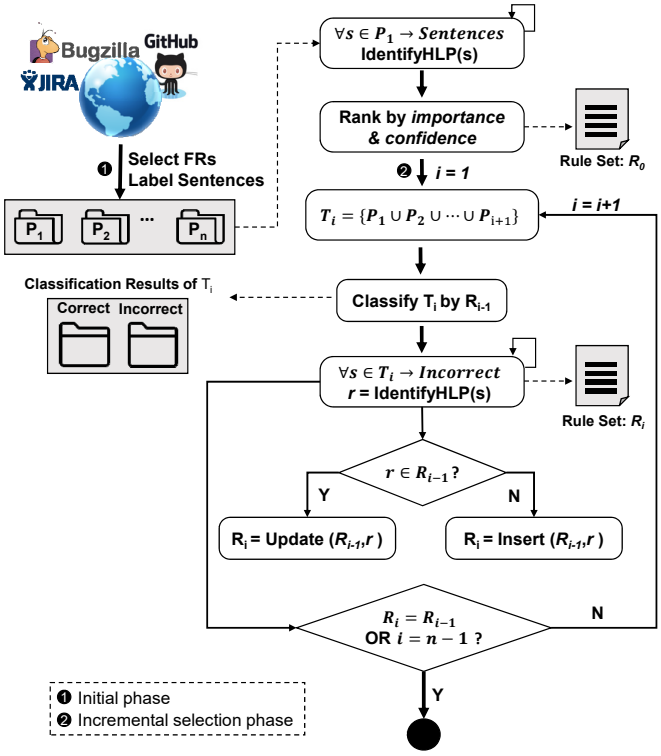

Fig. 2. The Process of Fuzzy Rules Generation.

- Inappropriate CF_k of existing rules. We define such rules as "Fake winner rules".
- Missing rules that are compatible with negative results.

Therefore, we analyze each sentence in the incorrect classification results to identify new rules or tune weights of the existing rules. As shown in Figure 2, we identify heuristic linguistic patterns and generate one fuzzy rule r for each sentence in the incorrect classification. If r has already existed in R_{i-1}, then we update the existing fuzzy rule set by tuning weights. If r is a new rule, then we insert it into R_{i-1}.

We use the maximum correctness as the optimization objective to control the trade-off between inserting new rules and tuning weights of existing rules. As shown in Algorithm 1, for each negative result that is caused by inappropriate CF_k of fake winner rules, we update the rules by tuning their weights. We first identify the real winner rule and the fake winner rule respectively. There are two operations in the function *update*. First we swap the weights between the real winner rule and the fake winner rule. Then we deduct the weights of the rest of the rules by 1 so that they have lower weights than the real winner rule. We use the $Correctness$ measurement defined in section II-C as the optimization objective, which means that the $update$ action is accepted only when the $correctness$ does not decline. Similar to the $update$ action, we insert new rules to all the possible positions in the existing rule set, and find the maximum $Correctness$. If the new $Correctness$ is larger than the current maximum Correctness, then the $insert$ action is accepted.

We repeat this step iteratively until there is no more new rules identified or weights tuned. As a result, we obtain a set of updated fuzzy rules that is more compatible with both the correct and incorrect classification results.

Algorithm 1 Insert or Update rules

```
1: procedure INSERT(Rule newRule, List ruleSet)
2:     positions ← |ruleSet| + 1
3:     oldCorrectness ← Correctness(ruleSet)
4:     maxCorrectness ← 0
5:     newRuleSet ← 0
6:     for each position p : positions do
7:         ruleSet.add(p, newRule)
8:         tmpCorrectness ← Correctness(ruleSet)
9:         if tmpCorrectness > maxCorrectness then
10:            maxCorrectness ← tmpCorrectness
11:            newRuleSet ← ruleSet
12:    if (newCorrectness > oldCorrectness) then
13:        return newRuleSet
14:    else
15:        return ruleSet
1: procedure UPDATE(List neg, List ruleSet Sentence text)
2:     true ← {r|r ∈ neg, r.C = text.class}
3:     realWinner ← {r|r ∈ true, r.CF = maxCF(true)}
4:     fakeWinner ← {r|r ∈ neg, r.CF = maxCF(neg)}
5:     oldCorrectness ← Correctness(ruleSet)
6:     newCorrectness ← 0
7:     newRuleSet ← ruleSet
8:     updates ← {r|r ∈ newRuleSet, realWinner.CF <
   r.CF ≤ fakeWinner.CF}
9:     for each Rule r: updates do
10:        r.CF = r.CF − 1
11:    realWinner.CF = fakeWinner.CF
12:    newCorrectness ← Correctness(newRuleSet)
13:    if (newCorrectness > oldCorrectness) then
14:        return newRuleSet
15:    else
16:        return ruleSet
```

E. Tool Support: Automated Structuring Feature Request

Based on the proposed approach, we implement an automated Feature Request Analyzer (FRA) online system that can automated structure the contents of a given feature request into a tree format, as shown in Fig. 3. We also provide a web API for other applications to access FRA. The analyzed results are returned in JSON format. More details can be found on our project site: http://goo.gl/cL00vd

III. Experimental Setup

Three research questions are proposed in Section III-A. In order to answer our research questions, we select and analyze nine open source projects in III-B and III-C. More specifically, we extract and tag sentences from feature requests that are found in the nine projects. Then fuzzy rules are generated and optimized incrementally until no rules can be added or changed. We evaluate the performances of these generated rules in Section IV.

A. Research Questions

The goal of the experiment is to evaluate the performance of the generated fuzzy rules, with the purpose of investigating

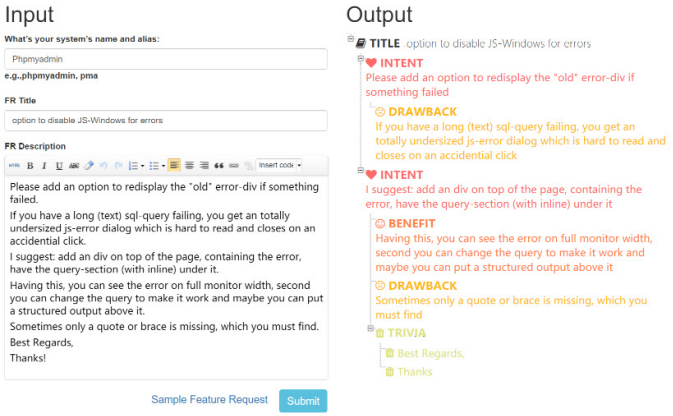

Fig. 3. Tool Support for automatically structuring feature requests.

when the approach can generate a stable set of fuzzy rules, how effective the fuzzy rules are when applied on datasets, and the performance of various machine learning methods when using the fuzzy rules as model variables. Specifically, the experiment aims at addressing the following research questions:

RQ1: *When can the generated fuzzy rules reach the state of convergence?* This research question aims at investigating how many projects it will take for the approach to generate a stable set of fuzzy rules (*i.e.* no new rules added and no existing rules changed). We apply an incremental method to add rules generated from one project at a time. Each increment is considered as an iteration. Then we use the correctness of each iteration as the measurement to determine the state of convergence. Moreover, we use the misclassification of each iteration to determine the stable set of fuzzy rules.

RQ2: *How effective are the fuzzy rules on classifying feature requests gathered from nine open source projects?* This research question aims at empirically evaluating the performance of the generated stable fuzzy rules on the nine selected projects. We first focus on classifying sentences from feature requests from the projects we used to generate these rules. Then we apply the rules on the remaining projects.

RQ3: *How do machine learning methods perform when using the fuzzy rules as their model variables?* This research questions aims at evaluating the performance of machine learning methods that use the generated stable fuzzy rules as their model variables. We use correctness and misclassification as the performance measurements to compare these methods with and without using the rules.

B. Subject Projects

We selected the subject projects from four representative open source communities: Apache, Eclipse, Github, and Hiberante, starting from a list of popular projects. From these, we selected 9 projects that (i) had at least 100 feature requests in their issue tracking systems, (ii) had active contributions in the last one month, (iii) covered 9 different domains, and (iv) covered 4 open source communities. Requirements (i) and (ii) aim to ensure that the data we collect are sufficient and up-to-date. Requirements (iii) and (iv) aim to keep the diversity of the data so that the fuzzy rules we generated can be general. The subject projects we selected and used in our experiments are as follows. The characteristics of subject projects and selected feature requests are listed in Table III.

- *Phpmyadmin(PMA)*: A web-based MySQL management tools written in PHP.
- *Mopidy*: An extensible music server written in Python.
- *Activemq*: An open source message broker that is written in Java.
- AspectJ: An aspect-oriented programming (AOP) extension created at PARC for the Java programming language.
- *Hibernate ORM*: An object-relational mapping tool for the Java programming language, written in Java.
- *SWT*: A UI toolkit used by the Eclipse Platform and most other Eclipse projects written in Java.
- *Log4j*: A Java-based logging utility.
- *HDFS*: The primary storage system that is used by Hadoop applications.
- *Archiva*: An extensible repository management software written in Java.

We then randomly selected these projects into training and testing datasets.

C. Data Preparation

Step 1: identify feature requests. The nine selected projects come from four different communities. All these communities manage feature requests via issue tracking systems. Feature requests are treated as one specific type of issues, typically tagged by "enhancement" or "new feature" labels. We retrieved these feature requests by using the search engine in the issue tracking systems, and then exported the title and the description of the retrieved issue as one feature request. For projects from Eclipse, Apache, and Hibernate communities, we were able to export feature requests easily, since the issue tracking system they use provide the export functionality. However, for projects from Github, we implemented a data collection tool that can automatically collect feature requests from search results.

Step 2: filter feature requests. For projects contain more than 200 feature requests, we randomly selected 200 of them as the dataset for deeper analysis. As reported by Herzig *et al.*[9], issue reports classification is not one hundred percent reliable, and more than 40% of the issue reports are inaccurately classified. Therefore, we conducted a manual inspection on feature requests to filter out the feature requests that are: (i) misclassified; (ii) low in readability/clarity; and (iii) with incomplete descriptions. We also replaced any source code and URL found in request descriptions with $< CODE >$ and $< LINK >$ respectively to maintain the natural language state of the requests. URLs were identified through regular expressions, while source code was filtered out automatically by using *infoZilla* found in [23].

Step 3: tag sentences. After obtaining a set of qualified feature requests as the subject data, we aimed at tagging each sentence with an appropriate category. The tagged sentences were used as the ground-truth dataset for generating

TABLE III
DATASET OF NINE SUBJECT PROJECTS

Project	Community	Domain	#FRs	#Selected FRs	#Sentences
PMA	Github	database tool	1740	57	193
Mopidy	Github	music server	312	62	189
Activemq	Apache	message broker	486	62	167
Log4j	Apache	logging	177	59	249
HDFS	Apache	storage	402	98	394
Archiva	Apache	repository tool	137	42	112
AspectJ	Eclipse	programming	472	96	362
SWT	Eclipse	UI toolkit	1540	44	147
ORM	Hibernate	mapping tool	778	82	299

fuzzy rules and evaluating the classification performances. To guarantee the correctness of the tagging results, we built an inspection team consisted with two senior researchers and four Ph.D candidates. All of them either have done intensive research work with issue tracking systems or have been actively contributing to open source projects. We divided the team into two groups. Each group consisted a leader (senior researcher) and two members (Ph.D candidates). The leader trained members on how to tag and provided consultation during the process. The two members tagged the same set of sentences. We only accepted and included the sentence to the dataset when the sentence received the same category from both members. When a sentence received different tagging results, we hosted a discussion with all six people and decided through voting. Overall, 89% of the 2,112 sentences obtained agreements between labelers. Among the intent, benefit, drawback, example, explanation, and trivia categories, the agreement rates were 90%, 87%, 82%, 85%, 92%, and 98%.

The size of tagged sentences and corresponding feature requests are shown in Table III. "#FRs" lists the number of feature requests of the project we retrieved by labels "new feature" or "enhancement". "#Selected FRs" represents the number of feature requests we selected after applying the data preparation process. "#Sentences" denotes the number of sentences collected from the selected feature requests. In total, we acquired 602 feature requests and 2,112 sentences. All the data can be downloaded from our project site: http://goo.gl/aWOvIX.

IV. Results and Analysis

This section reports the analysis of the results achieved aiming at answering our three research questions.

A. Results of RQ1

As described in section II, fuzzy rules are generated by incrementally introducing new projects until the rule set is stable. In our experiments, we gradually added projects one at a time to generate fuzzy rules until the rule set was stable. We used the correctness of each iteration as the measurement to determine the convergence of the fuzzy rules. By observing the growth trend of the rules in each iteration and the measurement of misclassification, we assessed and proved the validity of the final fuzzy rule set.

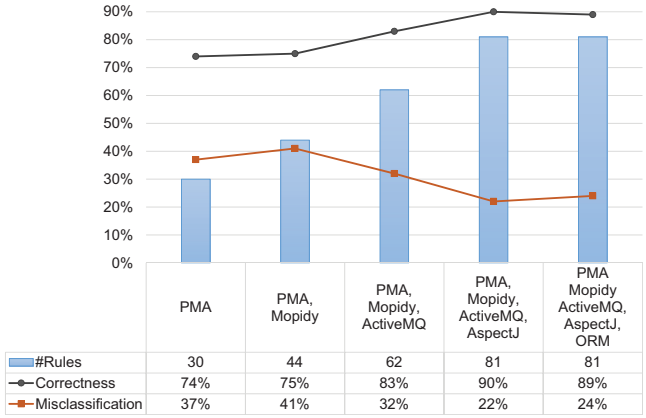

Fig. 4. Performance changes and growth of rules in each iteration

Figure 4 shows the number of the fuzzy rules in the first five iterations. We can see that the rules are growing from 30 to 81 in the first four iterations. The correctness also increases from 75% to 90% respectively. However, when introducing a new project during the fifth iteration, the correctness remains almost the same, which means that inserting new rules or updating existing rules cannot make any improvement on the rule set. Therefore, we concluded that with four projects, the fuzzy rules converged at a high correctness level of 90%.

Figure 4 also shows the changes in misclassification between iterations. The misclassification decreases from 37% to 22% in the first four iterations, and slightly increases to 24% at the fifth iteration. The results can support the conclusion on the state of convergence.

Furthermore, we verified that such performance trend in each iteration was found in individual projects. Figure 5 shows the correctness and misclassification for each project that have been used to generate fuzzy rules. Trend-lines end with "_C" represent correctness, and "_M" represent misclassification.

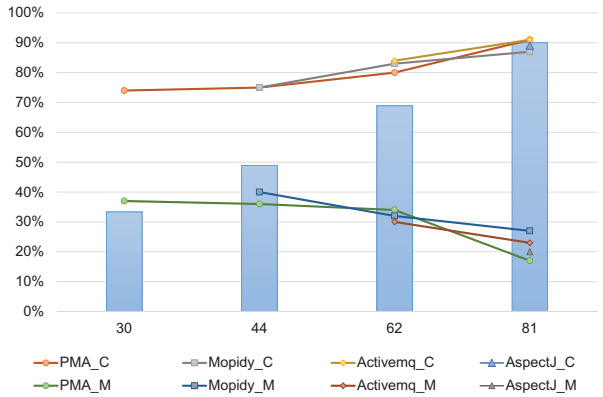

Fig. 5. Performance changes on individual projects

As the size of the fuzzy rules increases, the performance of these rules on each project is improving with higher correctness and lower misclassification.

Overall, these observations show that the fuzzy rule set can reach the state of convergence at the end of the fourth iteration with high performance. More specifically, our approach only took four projects to generate a stable set of 81 fuzzy rules.

B. Results of RQ2

As shown in Figure 4 and Figure 5, the stable set of fuzzy rules can achieve high performance when applied on the four projects where the rules were generated from. However, such high performance may caused by the fact that the rules were generated from the sentences in those projects. Therefore, we evaluated the performance of these fuzzy rules on projects that were not used in generating the rules.

Figure 6 shows the correctness and misclassification of the 81 rules in all projects. The left four data points represent the projects that were used in rules generation, and the right five data points represent the projects that were not used in rules generation. We distinguished those two types of projects into *fitted dataset* and *unfitted dataset* respectively. The correctness slightly fluctuates between 84% and 91% among the nine projects, and the average correctness is 87% , while the fluctuation of the misclassification is between 15% and 27% among all the projects and with an average of 22%. The results show that the 81 fuzzy rules perform steadily well on both the fitted dataset and unfitted dataset.

We further looked into the classification results on unfitted dataset for each category in terms of *precision*, *recall*, and *f-measure*. We used three boxplots to graphically illustrate the detailed classification results on the six categories (shown in Figure 7). The details of precision, recall and f-measure for each category are shown in Figure 8. Categories *Intent* and *Explanation* have the highest and most stable precisions of nearly 90%, while other categories have less stable yet still accurate precisions. The average precision on unfitted dataset among all categories is 85%. Each category has an average of 88% in recall, while *Intent* has the highest recall among all. F-measure considers both precision and recall and is used to

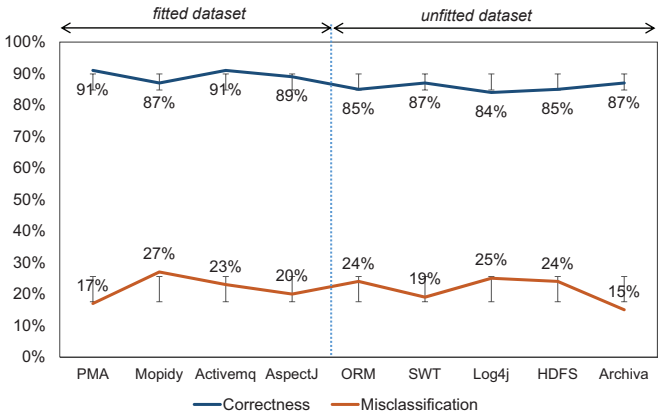

Fig. 6. Correctness and misclassification of the 81 rules in all projects.

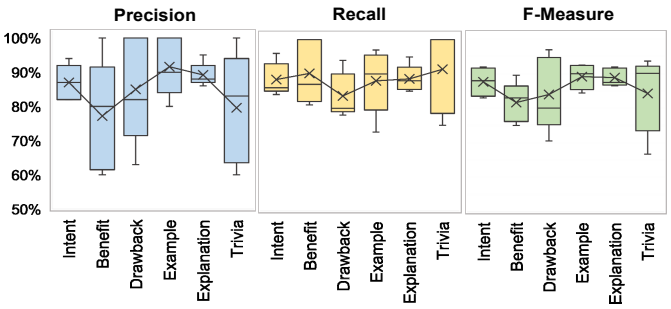

Fig. 7. Classification results on the six categories.

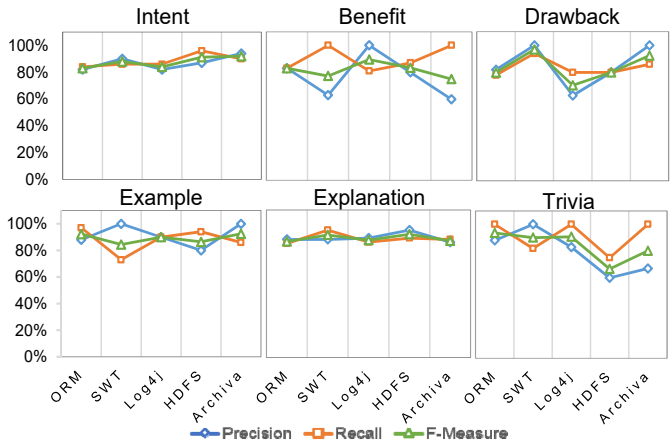

Fig. 8. Detail classification results on the six categories

show the predictive capability of the fuzzy rules. Categories *Intent* and *Explanation* have the highest and most stable f-measures while others are less stable but relatively good.

C. Results of RQ3

It is widely known that machine learning methods are commonly used for resolving classification tasks [24]. In this study, the following methods are used: Random Forest (RF), Naive Bayes (NB), J48, Logistic Regression (LR), and Support Vector Machine (SVM).

Choosing informative and discriminating features is important to apply machine learning methods effectively. In order to ensure that machine learning methods can use the features that are as informative and discriminating as the fuzzy rules, we constructed a dataset for each project as follows.

First, we mapped the 81 rules into 81 boolean variables. Then, for each sentence in the feature requests, we assigned 1 to the variable if the corresponding rule is satisfied. Otherwise, 0. We used the boolean datasets that are transformed from the unfitted five subject projects as the training datasets, and built classifiers for the five machine learning methods through *Weka* [25]. Since we leveraged modal verbs and NLP parsing results from complete sentences in 81 rules, we did not stem non-stop words from the feature requests for machine learning methods. We used the vector format and *TF-IDF* weights as features for machine learning methods without fuzzy rules [26]. Instead of using the four fitted datasets (40%), we selected 10-fold (90%)

on the unfitted five datasets when training machine learning classifiers to obtain higher performances. We evaluated the classification results of each machine learning model by *precision, recall, f-measure, correctness* and *misclassification*.

Figure 9 shows that the average precision, recall, and f-measure of the proposed fuzzy rules are the highest, following by the four machine learning classifiers with rules, and then four machine learning classifiers without rules.

As shown in Figure 10 and Figure 11, the proposed fuzzy rules can achieve the highest correctness and the lowest misclassification. Since the proposed approach considers correctness as the optimization objective while ML methods typically

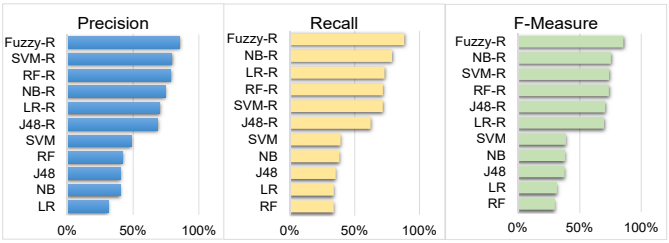

Fig. 9. Average performances for the nine classifiers.

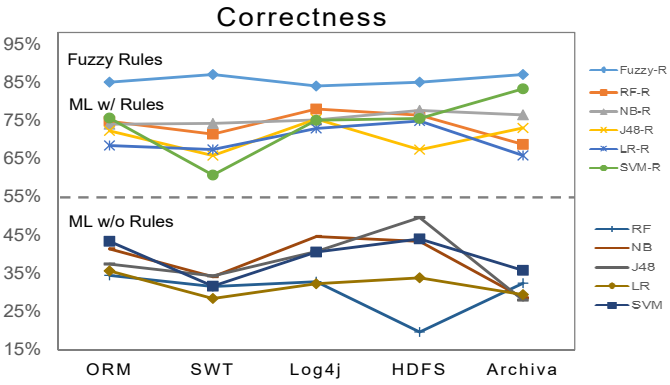

Fig. 10. Correctness for the nine classifiers.

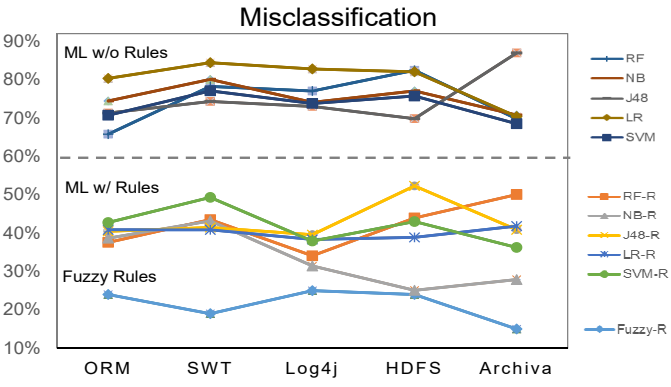

Fig. 11. Misclassification for the nine classifiers.

rely on probability, the proposed fuzzy rules outperformed the machine learning classifiers with rules. Most of the machine learning methods without rules have under 50% in correctness, and all of them have above 60% in misclassification. These methods might be incapable to classify feature requests. One main reason for having low performance is because that these methods can predict well on categories that mainly contain lexical patterns but not those with mainly syntax and semantic patterns. For instance, all of the nine methods can achieve a pretty good precision (69%) and recall (51%) in the category *Intent* but not in the category *Benefit*. However, when we incorporated the fuzzy rules as the model inputs, there was an over 50% improvement in performance for each machine learning method. These results indicate that the fuzzy rules can benefit the machine learning methods by providing more significant model variables.

V. Discussion and Future Work

In this section, we discuss the implications of our results and possible ideas of future work.

A. How to Use the Classification Results?

(For practitioners) Feature requests are more likely to be inaccurately submitted as bug reports [9], and buried under the intensive textual communications such as mailing lists [27]. Practitioners who want to acquire a complete sense of requested features need to spend a lot of effort on analyzing the massive unstructured textual artifacts. We offer a semantic paradigm of feature requests, including six categories that are extracted from confirmed feature requests from open source communities. By automatically tagging each sentence into one of the six categories, the classification results can provide practitioners with an initial understanding of the unstructured textual artifacts. The tagged contents can help practitioners easily locate the information they need (*e.g.,*benefit and drawbacks) and ignore the unwanted parts (*e.g.,*examples and trivial information) from feature requests.

(For *researchers*) The tool and results can be used to conduct further empirical studies on feature requests. We noticed that there are certain sequence patterns, which requesters like to follow when propose feature requests. For example, some requesters like to start by describing the drawbacks of the current system, and then explain what they want, while some requesters first describe what they want, and then show the corresponding benefits if the features they want are implemented. It would be interesting if the following questions can be answered: (1) How people describe software feature requests? What kinds of "patterns or manners" they follow? (2) What kinds of feature requests are easy/difficult to understand? In addition, the researchers can also use the classification results for further content mining in feature requests or other related textual artifacts, such as locating feature requests from massive textual artifacts by using patterns, and analyzing quality concerns in feature requests or bug reports.

B. Optimizing the Generated Fuzzy Rules

In order to optimize the fuzzy rules to improve their generalization in the open source communities, there is a need to analyze more feature requests from the enormous projects

available in these communities. However, it is not feasible to identify new linguistic patterns and optimize the fuzzy rules by only human efforts with such big datasets. It becomes beneficial to automate the process to learn linguistic patterns through leveraging data mining techniques. As discussed earlier, there are three types of linguistic patterns: lexical, syntax, and semantics patterns. In most cases, lexical patterns are the phrases that commonly appear among sentences in the same group. It is possible to extract lexical patterns by checking similarity in N-Gram phrases. Syntax patterns describe a higher level of knowledge of sentences in terms of POS tags. The sequences of POS tags for the sentences (*e.g.*, {VB, DT, NN}) can be considered as the transaction records for association rule mining. It is possible to extract syntax patterns by using association rule mining to discover interesting relations between POS tags. The combination of Stanford dependencies and heuristic glossaries can indicate the meaning of a sentence. Stanford dependencies can help us build the relationship between two words, and the heuristic glossaries can help us define the meaningfulness similarity between two words. If any relationship is found between two words, a graph can be built for a given sentence based on the Stanford dependences between two words. The maximum common edge subgraph[28] of sentences in the same group can be used to calculate of graph similarity and as the semantics patterns. We will explore this possibility in our future work.

In addition, we plan to improve and extend the 10 glossaries we build in our approach by leveraging automated extraction and clustering techniques [29].

C. Feature Requests/Bug Classification on Software Qualities

As mentioned in previous Section V-A, the classification results can be used for further content mining. We plan to use the results from further content mining of feature requests and bug reports to examine the current quality status of the software. We believe that user requirements and existing bugs in the system can reflect the quality status of the software. Boehm *et al.*[30] proposed a System and Software Qualities (SQs) ontology that shows the relationship between a set of key software qualities. He also identifies the key roles of maintainability, not just for systems and software life cycle costs, but also for systems and software changeability and availability. We plan to use the ontology as a guideline to conduct the study with a focus on software maintainability, changeability and availability.

VI. Threats to Validity

External Validity The results of our study may not generalize beyond the projects we evaluated. To mitigate this threat, we selected nine popular and well-known open source projects from four different communities as well as nine different domains. To test the generality of our results, we used the fuzzy rules extracted from four projects, and tested their performance on the other five projects. We observed that the fuzzy rules constructed from the four projects were useful and could classify correctly in other five projects as well. In our future work, we plan to optimize the fuzzy rules by analyzing feature requests in a large scale as discussed in Section V-B.

The subject projects that were used in this study are very well established. The conclusions we drew may not reflect in feature requests found in newly established projects. As future work, we plan to take a deeper look at the characteristics of feature requests during aging, while comparing the differences between feature requests found in mature projects with fresh projects. Moreover, compared to traditional projects, this kind of feature request management approach may not be needed in agile and continuous delivery and development ops situation. We plan to expand our projects to include projects that follow agile process to examine whether the proposed approach is applicable to those projects.

Internal Validity Threats to internal validity might come from the process of manual inspection and tagging. We understand that such a process is subject to mistakes. To reduce that threat, we build an inspection team to reach agreements on different options.

Construct Validity Threats to construct validity are on how the performance and misclassification measurement are defined. Both correctness and misclassification rely on the importance of the categories to tune weights. We define the importance by using the natural number sequence from one to six, which might not reflect the actual importance in reality. However, our experiments show that the fuzzy rules extraction is effective on the nine subject projects. For projects that do not define the importance in natural sequence, our existing rules cannot apply. By following the approach and steps of our experiments, a new set of fuzzy rules can be extracted and applied.

Conclusion Validity When drawing the conclusion for *RQ3*, there might exist a threat in the comparison between the machine learning methods with and without applying the generated fuzzy rules. We use a lot of linguistic variables generated from syntax and semantic patterns, but information retrieval based machine learning methods mainly use variables at lexical level such as tf-idf weights and cosine similarity[31]. We plan to extend the comparison to include more advanced rule-based classification methods to minimize the threat in the near future.

VII. Related Work

Automated Support for Feature Request Management. Cleland-Huang *et al.* [32] proposed an automated forum management system for organizing discussion threads and grouping feature requests so that users can easily navigate through all the posts and find the location to place new feature requests. Rahimi and Cledland-Huang [33] proposed an partially automated approach to create personas from a set of feature requests in open forums by applying association rule mining. Gill *et al.* [34] offered a framework that solves the natural language ambiguity problem in the Open Source Software Development by combining the positive attributes of automation-oriented domains with manual supports from

developers. Shi *et al.* [35] conducted an empirical study on Hibernate Community with 11 project and 1898 feature requests. They proposed a Feature Tree Model to automatically identify redundant feature requests. Thung *et al.* [36] proposed an automated recommendation framework that uses the textual description of a feature request as input, then outputs a list of recommended methods from a set of library APIs that can be used to implement the feature request. Many studies have been conducted in the area of supporting feature request management automatically. Most of the existing studies adopt machine learning techniques. Yet our work takes on a different direction and focuses on structuring and analyzing individual feature request through generating fuzzy rules.

Detecting Feature Request. Di Sorbo *et al.*[37] proposed a classification method to classify development emails according to their purposes by using natural language parsing techniques. The proposed approach also can be used for a wider application domain, such as the preprocessing phase of various summarization tasks. Merten *et al.*[38] evaluated and compared multiple preprocessing techniques and machine learning techniques to detect software feature requests in issue tracking systems. Panichella *et al.*[39] presented an automated approach to analyze user reviews into categories such as feature request, opinion asking, *etc.* through natural language processing, text analysis and sentiment analysis. Maalej and Nabil[40] proposed a classification method that classifies reviews from App stores into bug reports, feature requests, user experience, and ratings by combining probabilistic techniques with text classification, natural language processing and sentiment analysis techniques. Other studies have been found to also capture user needs from app reviews automatically[41][27][42]. Kochhar, Thung, and Lo [43] proposed an approach that extracts various feature values from a bug report. Based on the values, they predicted whether a bug report needed to be reclassified. Panichella *et al.*[44] presented a tool that combines natural language parsing, text analysis and sentiment analysis to automatically classify useful feedback from app reviews in order to perform software maintenance and evolution tasks. Compared to the existing studies, our approach combines fuzzy theory with linguistic analysis and uses the maximum of correctness as the optimization condition to generate fuzzy rules in an incremental way, so that we can provide new solution to find rules and facilitate feature requests detection.

Structuring Issue reports. Panichella *et al.*[45] mined external communication channels such as emails and bug reports to structure them into source code documentation. The approach used the textual similarity measurement between each paragraph in those emails and bug reports and the method content in the source code. Bettenburg *et al.*[12] built a tool to detect structural information from bug reports. Their approach used patch information, stack trace frames, source code and enumeration as filters to extract structures from bug reports.

Although feature requests are typically managed as issues in the issue tracking systems, methods to structure issue reports can not support the structuring of feature requests. Our work provides a stable set of fuzzy rules to structure feature requests.

VIII. Conclusion

In this paper, we have presented an approach based on fuzzy method and natural language process to automatically classify the content of feature requests according to the six categories: intent, benefit, drawback, example, explanation, and trivia. Our approach builds on the Stanford NLP Parser and applies an incremental optimization algorithm. To evaluate the our approach, we have conducted an empirical study on 2,112 sentences taken from 602 feature requests of nine popular open source projects. The results indicate that our approach can reach high performance with the correctness of 87% and misclassification of 22% on the classification results. Moreover, when using fuzzy rules as model variables, the machine learning methods with fuzzy rules can also improve performances over 50% compared to the methods without rules. Therefore, we conclude that the fuzzy rules we generated are effective and useful in classifying feature requests. Additionally, we have implemented a tool that uses the approach to automatically structure the contents of feature requests. The tool is publicly available along with our experiment datasets on the project website. We believe that our contributions can make the feature requests easier to understand and analyze, thus improves feature request management in existing issue tracking systems.

Acknowledgments

We would like to thank Mingzhe Xing, Muhammad Ilyas Azeem and Kamonphop Srisopha for their help during development of the ideas presented in this paper. This material is based upon work supported by National Natural Science Foundation of China No. 61432001, 61602450, the National Science and Technology Major Project under Grant No. 2014ZX01029101-002, and the Youth Innovation Promotion Association CAS under Grant No. 2016105. It is also supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. It is also supported by the National Science Foundation grant CMMI-1408909, Developing a Constructive Logic-Based Theory of Value-Based Systems Engineering.

References

[1] Bugzilla, "Bugzilla, a Bug-Tracking System." [Online]. Available: https://www.bugzilla.org

[2] JIRA, "JIRA: plan, track, and release software." [Online]. Available: https://www.atlassian.com/software/jira

[3] Github, "Github, a Web-based Git or Version Control Repository and Internet Hosting Service." [Online]. Available: https://github.com/

[4] X. Franch and G. Ruhe, "Software release planning," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, 2016, pp. 894–895.

[5] Y. Shin, J. H. Hayes, and J. Cleland-Huang, "Guidelines for benchmarking automated software traceability techniques," in *8th IEEE/ACM International Symposium on Software and Systems Traceability, SST 2015, Florence, Italy, May 17, 2015*, 2015, pp. 61–67.

[6] P. Mäder, R. Oliveto, and A. Marcus, "Empirical studies in software and systems traceability," *Empirical Software Engineering*, vol. 22, no. 3, pp. 963–966, 2017.

[7] G. Uddin and M. P. Robillard, "How API documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015.

[8] L. Shi, H. Zhong, T. Xie, and M. Li, "An empirical study on evolution of API documentation," in *Proc. International Conference on Fundamental Approaches to Software Engineering (FASE 2011)*, March-April 2011, pp. 416–431.

[9] K. Herzig, S. Just, and A. Zeller, "It's not a Bug, it's a Feature: How Misclassification Impacts Bug Prediction," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 392–401.

[10] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, 2017.

[11] A. J. Ko, B. A. Myers, and D. H. Chau, "A Linguistic Analysis of How People Describe Software Problems," in *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, 2006, pp. 127–134.

[12] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting Structural Information from Bug Reports," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, 2008, pp. 27–30.

[13] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What Makes a Good Bug Report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2008, pp. 308–318.

[14] G. Trinder, "How to Write a Feature Request," 2006 (accessed May 5, 2017). [Online]. Available: https://blogs.msdn.microsoft.com/tdragger/2006/01/24/how-to-write-a-feature-request/

[15] Werner, "How do I Write a Good Feature Request?" 2015 (accessed May 5, 2017). [Online]. Available: https://meta.stackexchange.com/questions/7656/how-do-i-write-a-good-answer-to-a-question

[16] T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*. Intellect Books, 1986.

[17] M.-C. De Marneffe and C. D. Manning, "The Stanford Typed Dependencies Representation," in *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*, 2008, pp. 1–8.

[18] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[19] E. Hllermeier, "Fuzzy Methods in Machine Learning and Data Mining: Status and Prospects," *Fuzzy Sets and Systems*, vol. 156, no. 3, pp. 387 – 406, 2005.

[20] R. Agrawal, T. Imieliński, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," in *Acm sigmod record*, vol. 22, no. 2, 1993, pp. 207–216.

[21] C. D. Manning, P. Raghavan, and H. Schtze, "An Introduction to Information Retrieval," *Journal of the American Society for Information Science & Technology*, vol. 43, no. 3, pp. 824–825, 2008.

[22] S. V. Stehman, ""Selecting and Interpreting Measures of Thematic Classification Accuracy"," *Remote Sensing of Environment*, vol. 62, no. 1, pp. 77 – 89, 1997.

[23] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," in *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 2008, pp. 27–30.

[24] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[25] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016.

[26] G. Salton and C. Buckley, "Term-weighting Approaches in Automatic Text Retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[27] 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*, 2016, pp. 499–510.

[28] J. W. Raymond, E. J. Gardiner, and P. Willett, "Rascal: Calculation of Graph Similarity using Maximum Common Edge Subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.

[29] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated Extraction and Clustering of Requirements Glossary Terms," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2016.

[30] B. Boehm, C. Chen, K. Srisopha, and L. Shi, "The Key Roles of Maintainability in an Ontology for System Qualities," in *INCOSE International Symposium*, vol. 26, no. 1, 2016, pp. 2026–2040.

[31] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using IR methods for Labeling Source Code Artifacts: Is it Worthwhile?" in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 193–202.

[32] J. Cleland-Huang, H. Dumitru, C. Duan, and C. Castro-Herrera, "Automated Support for Managing Feature Requests in Open Forums," *Communications of the ACM*, vol. 52, no. 10, pp. 68–74, 2009.

[33] M. Rahimi and J. Cleland-Huang, "Personas in the Middle: Automated Support for Creating Personas as Focal Points in Feature Gathering Forums," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 479–484.

[34] K. D. Gill, A. Raza, A. M. Zaidi, and M. M. Kiani, "Semi-automation for Ambiguity Resolution in Open Source Software Requirements," in *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on*, 2014, pp. 1–6.

[35] L. Shi, C. Chen, Q. Wang, and B. Boehm, "Is It a New Feature or Simply Don't Know Yet? On Automated Redundant OSS Feature Requests Identification," in *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, 2016, pp. 377–382.

[36] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic Recommendation of API Methods from Feature Requests," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, 2013, pp. 290–300.

[37] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "Development Emails Content Analyzer: Intention Mining in Developer Discussions (T)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, 2015, pp. 12–23.

[38] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech, "Software Feature Request Detection in Issue Tracking Systems," in *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, 2016, pp. 166–175.

[39] S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How Can I Improve my App? Classifying User Reviews for Software Maintenance and Evolution," in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, 2015, pp. 281–290.

[40] W. Maalej and H. Nabil, "Bug report, Feature Request, or simply Praise? on Automatically Classifying app Reviews," in *2015 IEEE 23rd international requirements engineering conference (RE)*, 2015, pp. 116–125.

[41] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release Planning of Mobile Apps based on User Reviews," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 14–24.

[42] F. Palomba, M. L. Vásquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "User Reviews Matter! Tracking Crowdsourced Reviews to Support Evolution of Successful Apps," in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, 2015, pp. 291–300.

[43] P. S. Kochhar, F. Thung, and D. Lo, "Automatic fine-grained issue report reclassification," in *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*. IEEE, 2014, pp. 126–135.

[44] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "Ardoc: App reviews development oriented classifier," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 1023–1027.

[45] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora, "Mining Source Code Descriptions from Developer Communications," in *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, 2012, pp. 63–72.