NERO: A Text-based Tool for Content Annotation and Detection of Smells in Feature Requests

Fangwen Mu[1,2], Lin Shi[1,2*], Wei Zhou[1], Yuanzhong Zhang[1], Huixia Zhao[1]
[1]Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China.
[2]University of Chinese Academy of Sciences, Beijing, China.
{fangwen, zhouwei, yuanzhong, huixia}@itechs.iscas.ac.cn, {shilin}@iscas.ac.cn

***Abstract*—Utilizing massive user feedback, e.g. feature requests from Bugzilla, JIRA, or GitHub, to motivate software evolution has become a new trend in RE community. However, manually understanding and analyzing feature requests from issue tracking systems is a time-consuming and labor-intensive task. In this paper, we present NERO (coNtent annotation and smElly Feature Requests detectiOn), an automated tool to support analysts to understand the semantic meaning of feature requests and detect the smells in feature requests. It can also provide an overall score based on the smell detection results to help analysts quickly judge the quality of feature requests.**

***Index Terms*—Feature Request, Natural Language Process, Smell Detection**

I. Introduction

In the last years, a new trend in requirements engineering is to motivate software evolution by gathering and analyzing user feedback from the issue tracking systems. As a form of user feedback, feature requests play an important role in software maintenance and evolution. However, understanding and analyzing feature requests from a large number of user feedback in the issue tracking systems is a time-consuming and labor-intensive task. First, due to there is no clear specification that restricts the content or format of the feature requests in open-source communities, expressiveness problems (such as ambiguities, poor readability) inevitably appear in the feature requests[1]. Second, the content of the feature requests is often complex or confusing (may include hyper-links, code, or even trivial sentences)[2], which also requires analysts to spend a lot of time to understand the real intent of these feature requests.

In this paper, we propose NERO, a text-based tool that supports automatic analysis of feature requests. First, we classify each sentence in the feature requests into six semantic categories (intent, explanation, advantages, disadvantages, examples, and trivia) based on our previous work [2]. Then, we summarize three categories of quality defects (what we call feature requests smells) for expressiveness problems in feature requests. We use natural language processing technologies to detect these smells that we define. Finally, we leverage the fuzzy comprehensive evaluation method to assess the quality of feature requests based on the results of the smells detection.

II. The NERO Tool

The architecture of NERO is shown in Fig 1. Below, we discuss its three main functionalities.

*Corresponding author.

A. Content Annotation

The content annotation of the feature requests can help the analysts to obtain the structural information of the feature requests and to understand the semantic meaning of each sentence. As shown in the "Content Analysis" part of Figure 1, we use a "rule matcher" containing the 81 fuzzy rules proposed in the previous work [2] to classify each sentence into six categories (intent, explanation, advantages, disadvantages, examples, and trivia) that represent its semantic meaning. The detailed definitions of the six categories are shown in Table III in the appendix.

B. Smells Detection

Existing literatures propose quality criteria for feature requests or the quality characteristics that "good" feature requests should possess[1][3]. Unfortunately, they do not support corresponding automated quality checks. Some quality problems in the quality criteria for feature requests require specific domain knowledge to solve. Therefore, in this paper, we focus on expressiveness problems in detecting feature requests. Through the detection of smells about expressiveness problems and the overall score can help requirements analysts improve the detected smells and quickly filter out some inaccurate and difficult to understand feature requests. We summary ten smells that can be divided into three categories based on existing quality criteria for feature requests[1] and requirement quality models[4][5].

As shown in the "Smells Detection" part in Figure 1, we can automatically detect these smells from textual feature requests by leveraging various NLP techniques, such as GPT2 language model, POS tagging, dependency parsing, Coleman formula. The detailed explanations and detection methods of smells are shown in the Table II in the appendix.

C. Overall Assessment

The overall assessment of the feature requests quality allows the analysts to have a preliminary judgment on the quality of the feature requests. They can prioritize clearly stated (high-scoring) feature requests based on the overall score. As shown in the "Overall Assessment" part in Figure1, we leverage the fuzzy comprehensive evaluation (FCE) method and the analytic hierarchy process (AHP) to assess the quality of feature requests based on the results of the smells detection.

FCE method is a mathematical method based on fuzzy set theory developed by Zadeh[6]. It comprehensively evaluate

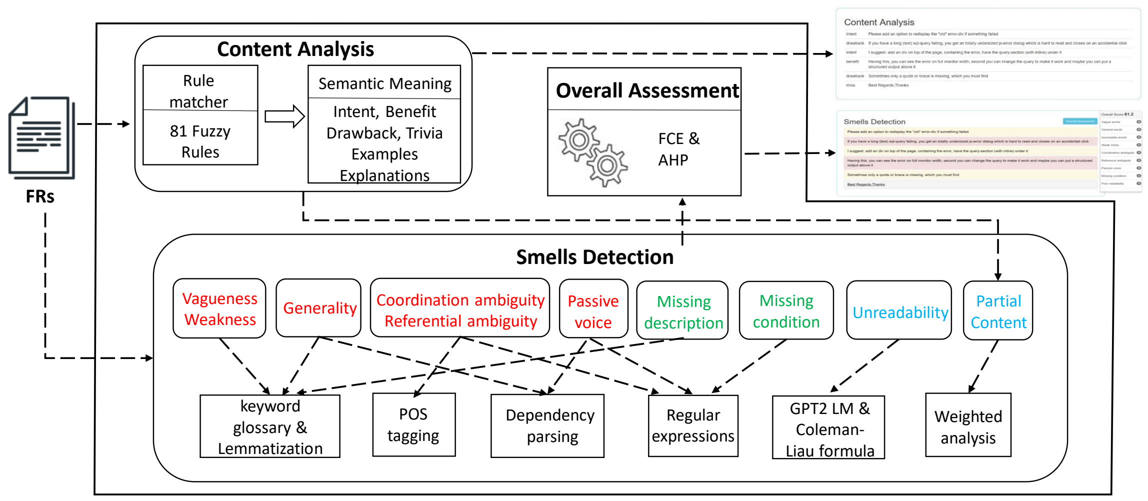


Fig. 1. The Overview of *NERO*

things that are not easy to be clearly defined in the real world by using the thinking and methods of fuzzy mathematics. The AHP was developed by Saaty[7] and has been used to determine the weights of different influencing factors during the evaluation process. In this paper, the quality of feature requests is affected by three smells, and the impact of each of these three smells on quality is difficult to estimate quantitatively. Therefore, we use FCE method to assess the quality of feature requests. We follow the steps of the FCE method in order: determine a factor set, determine an evaluation set, use AHP to determine the factor weights, determine the membership function, and complete the overall score.

III. Evaluation

To evaluate whether our tool can effectively assess the expressiveness quality of feature requests, we rank ten feature requests from the issue tracking system through manual scoring by human analyst and automatic scoring by our tool and get two ranked lists respectively. The ranking results are shown in Table I. Then, we use Spearman's ρ metrics to evaluate the ranking performance of our tool. Spearman's ρ is a common rank correlation metrics for measuring the similarity of two ranked lists[8]. The more similar the two ranking lists are, the closer the value of the Spearman's ρ is to 1, the better the ranking performance. We use the SPSS tool[9] to calculate the Spearman of two ranked lists. The result is that the value of Spearman's ρ is 0.891, and a significance level of 0.01. It shows that there is a significant positive correlation between these two ranked lists and our tool can effectively assess the quality of feature requests.

TABLE I
Ranking of ten feature requests by analysts and NERO

Feature Request Id	1	2	3	4	5	6	7	8	9	10
Ranking by Analyst	1	2	3	4	5	6	7	8	9	10
Ranking by NERO	2	1	4	6	3	5	9	7	8	10

IV. Related Work

Heck and Zaidman[1] develop a framework for the quality assessment of just-in-time (JIT) requirements and instantiate this framework for feature requests. Scacchi et al.[3]argues that the requirements artifacts in open source software development can be evaluated by (1) encouragement of community building; (2) freedom of expression; (3) readability; (4) and implicit versus explicit structures for organizing. Compared to the existing studies, our work focuses on the expressiveness problems in feature requests and summarizes three categories of smells that affect their expressiveness quality. We also provide automated methods for content annotation, smells detection, and quality assessment of feature requests.

V. Conclusion and Future Work

This paper presents a tool that allows users to better understand the content of feature requests and improve the quality of elicited and elaborated requirements from such feature requests. Besides, users can make a preliminary judgment on the quality of feature requests based on the overall score. As for future work, we will extend the tool to enable automatic identification of feature requests from user feedback and to support batch processing of a large number of identified feature requests instead of the current manual editing input.

Acknowledgement

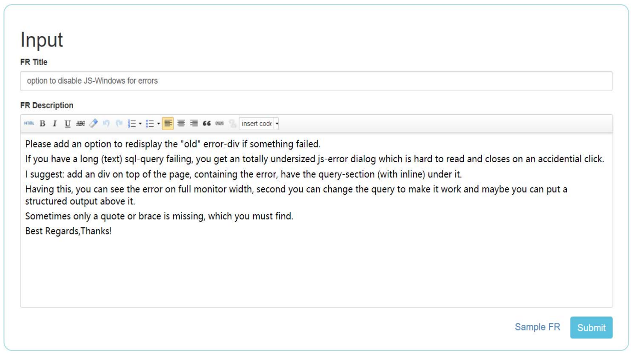
This work is supported by the National Science Foundation of China under grant No.61802374, No.61432001, No.61602450, and the National Key Research and Development Program of China under grant No.2018YFB140-3400.

References

[1] P. Heck and A. Zaidman, "A framework for quality assessment of just-in-time requirements: the case of open source feature requests," *Requirements Engineering*, vol. 22, no. 4, pp. 453–473, 2017.
[2] L. Shi, C. Chen, Q. Wang, S. Li, and B. W. Boehm, "Understanding feature requests by leveraging fuzzy method and linguistic analysis," in *ASE 2017*, pp. 440–450, 2017.
[3] W. Scacchi, "Understanding requirements for open source software," pp. 467–494, 2009.
[4] G. Lami, "Quars: A tool for analyzing requirement," Tech. Rep. CMU/SEI-2005-TR-014, SEI, CMU, 2005.
[5] S. F. Tjong and D. M. Berry, "The design of SREE - A prototype potential ambiguity finder for requirements specifications and lessons learned," in *REFSQ 2013*, vol. 7830, pp. 80–95, 2013.
[6] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.
[7] T. L. Saaty, "What is the analytic hierarchy process?," in *Mathematical models for decision support*, pp. 109–121, 1988.
[8] D. K. Agarwal and B.-C. Chen, *Evaluation Methods*, p. 55–78. 2016.
[9] I. Spss *et al.*, "Ibm spss statistics for windows, version 20.0," *New York: IBM Corp*, vol. 440, 2011.

Appendix

The design of the NERO's user interface follows the principle of simplicity. The purpose of this is to allow users to more intuitively obtain structural information that can help them to understand and analyze the feature requests. As mock-analysts, participants are encouraged to use the tool demo to automatically analyze the input feature request and they will get a general overview of (1) the content annotation, (2) the smells detection, and (3) the overall assessment.



Input

FR Title

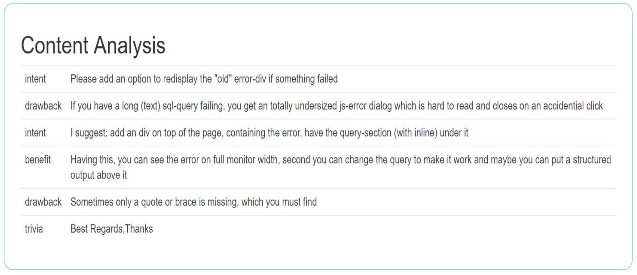
option to disable JS-Windows for errors

FR Description

Please add an option to redisplay the "old" error-div if something failed.
If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click.
I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it.
Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it.
Sometimes only a quote or brace is missing, which you must find.
Best Regards,Thanks!

Fig. 2. Input of feature request

Participants first need to enter the feature request to be analyzed in the Input section, which includes two parts: title and content description (see Figure 2). They can also click on the Sample FR link in the lower right. After clicking, the tool will automatically type in the input box a sample, which is a real feature request from the issue tracking systems. Then, participants can click the Submit button for analysis to trigger the content annotation functionality.

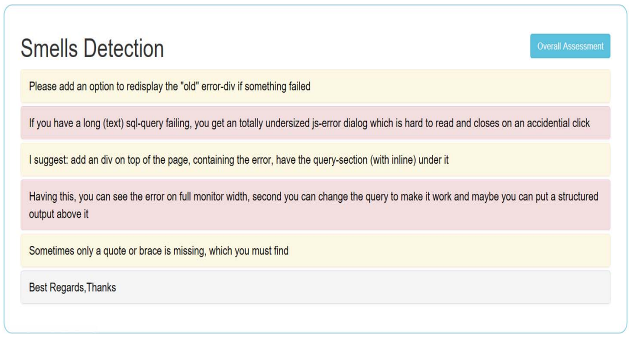


Content Analysis

intent	Please add an option to redisplay the "old" error-div if something failed
drawback	If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click
intent	I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it
benefit	Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it
drawback	Sometimes only a quote or brace is missing, which you must find
trivia	Best Regards,Thanks

Fig. 3. Content Analysis of feature request

Figure 3 shows the content analysis of a sample feature request entered by participants. Each line presents the annotation result of each sentence in the input feature request, where the first half represents the semantics of the sentence, and the second half is the original sentence. The six categories (intent, explanation, advantages, disadvantages, examples, and trivia) are defined in Table III.



Smells Detection

Please add an option to redisplay the "old" error-div if something failed

If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click

I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it

Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it

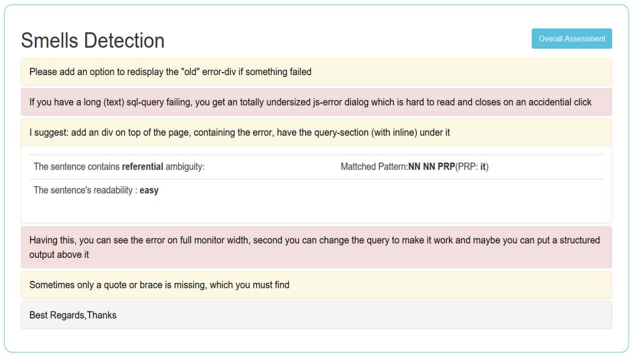
Sometimes only a quote or brace is missing, which you must find

Best Regards,Thanks

Fig. 4. Detection of smells in feature request

As shown in Figure 4, each line is a collapsible panel in the text box of smells detection, and we select a different color background to show the original sentence. The tool classifies the sentence into three levels according to the number of smells in the sentence: zero, low, and high, which correspond to the three background colors (white, yellow, and red). Three error levels and their corresponding background colors are defined in Table IV.

If the background color of a row is displayed as yellow or red (i.e., there are smells in the sentence of feature requests), the participants can see all the detected smells by clicking the sentence link on the panel title (see Figure 5 and 6). Each line in the content part of the panel represents one of the ten smells detected in feature request. For smells in lexical level (e.g. vague, weak, and incomplete), the tool will display the extracted keywords; for smells in syntactic level (e.g. coordination ambiguity and referential ambiguity), the tool will display information about the matched patterns; for readability quality property, the tool will display the readability level of the sentence (easy, hard). If there are no smells in the sentence, the tool will also give the participants a hint (see Figure 7).



Smells Detection

Please add an option to redisplay the "old" error-div if something failed

If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click

I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it

The sentence contains **referential** ambiguity: Mattched Pattern:**NN NN PRP**(PRP: **it**)

The sentence's readability : **easy**

Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it

Sometimes only a quote or brace is missing, which you must find

Best Regards,Thanks

Fig. 5. An example of a sentence with low error level

Figure 5 shows all the detected smells in the third sentence of the input feature request. There is only one smell in this sentence, so the corresponding error level is low.

TABLE II
EXPLANATIONS AND DETECTION METHODS OF SMELLS

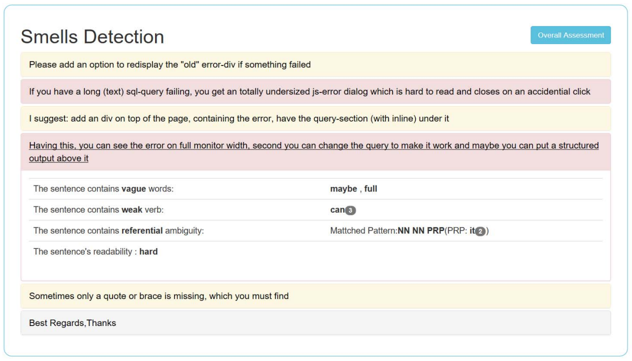
Smell Category	Smell Name	Explanation	Detection method
Ambiguous	Vagueness	Vagueness occurs whenever a statement admits borderline cases, such as appropriate, clear, significant, etc.	Keyword glossary, Lemmatization
	Weakness	Weakness occurs when the feature requests use words with weak semantic content and little emotional color, such as could, may, might, etc.	Keyword glossary, Lemmatization
	Generality	Generality occurs when the sentence contains words that identify a certain type of object, and no modifiers limit its scope, such as flow, access, data, interface, etc.	Keyword glossary, Lemmatization, Dependency parsing
	Coordination ambiguity	Coordination ambiguity occurs when the use of coordinating conjunctions leads to multiple potential interpretations of a sentence.	POS tagging,Regular expression
	Referential ambiguity	Referential ambiguity occurs when an anaphor (e.g. it, that, which, etc.) can take its reference from more than one element, each playing the role of the antecedent.	POS tagging,Regular expression
	Passive voice	Passive voice occurs when the passive voice is used in the feature requests.	Dependency parsing,Regular expression
Incomplete	Missing condition	Missing condition occurs when the sentence contains an if clause expressing the condition, but there is no corresponding else/otherwise clause.	Keyword glossary,Lemmatization
	Missing description	Missing description occurs when the sentence contains omitted-meaning words, such as as defined, to be completed, to be determined, etc.	Regular expression
Unintelligible	Unreadability	Unreadability occurs when the sentences in one feature request are too long or not smooth.	GPT2 LM,Coleman-Liau formula
	Partial Content	Partial Content occurs when the feature requests lack any of the five semantic annotations (except Trivia) mentioned in the content annotation.We assume that feature requests with more different content annotations will deliver more diverse information.	Weighted analysis

TABLE III
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.

TABLE IV
DEFINITION OF ERROR LEVEL

Error Level	The Number of Smells	Background Color
zero	0	white
low	0 to 4	yellow
high	4 or more	red



Please add an option to redisplay the "old" error-div if something failed

If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click

I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it

Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it

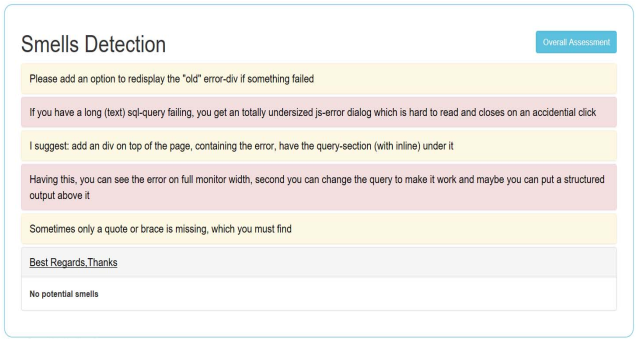
The sentence contains **vague** words:	**maybe** , **full**
The sentence contains **weak** verb:	**can**3
The sentence contains **referential** ambiguity:	Matcthed Pattern:**NN NN PRP**(PRP: **it**2)
The sentence's readability : **hard**	

Sometimes only a quote or brace is missing, which you must find

Best Regards,Thanks

Fig. 6. An example of a sentence with high error level

Figure 6 shows all the detected smells in the fourth sentence of the input feature request. The number of smells in this sentence is more than four, so the error level is high.



Please add an option to redisplay the "old" error-div if something failed

If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click

I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it

Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it

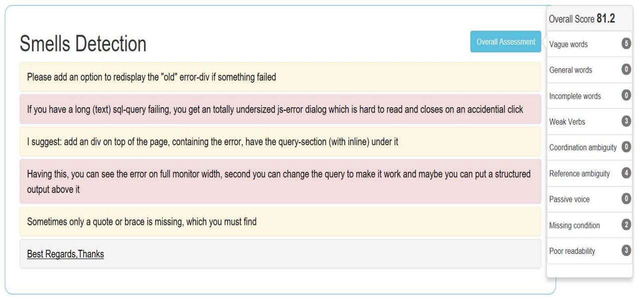
Sometimes only a quote or brace is missing, which you must find

Best Regards,Thanks

No potential smells

Fig. 7. An example of a sentence with zero error

Figure 7 shows the detection result of the last sentence in the input feature request. There is no smell in this sentence, so the corresponding error level is zero error.



Please add an option to redisplay the "old" error-div if something failed

If you have a long (text) sql-query failing, you get an totally undersized js-error dialog which is hard to read and closes on an accidential click

I suggest: add an div on top of the page, containing the error, have the query-section (with inline) under it

Having this, you can see the error on full monitor width, second you can change the query to make it work and maybe you can put a structured output above it

Sometimes only a quote or brace is missing, which you must find

Best Regards,Thanks

Overall Score **81.2**	
Vague words	5
General words	0
Incomplete words	0
Weak Verbs	3
Coordination ambiguity	0
Reference ambiguity	4
Passive voice	0
Missing condition	2
Poor readability	3

Fig. 8. Overall assessment of a feature request

Figure 8 shows the overall assessment of the feature request. The pop-up window on the right sidebar contains the content of the overall assessment of the text, including the overall score and the number of occurrences of various smells. Participants can use the structured information provided in the overall assessment to quickly make a rough judgment about the quality of the input feature request. They can view or hide this pop-up window by clicking the overall assessment button.