

Automated Quality Assessment for Crowdsourced Test Reports of Mobile Applications

Xin Chen^{*†}, He Jiang^{*†}, Xiaochen Li^{*†}, Tieke He[‡], Zhenyu Chen[‡]

^{*}School of Software, Dalian University of Technology, Dalian, China

[†]Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian, China

[‡]School of Software, Nanjing University, Nanjing, China

chenxin4391@mail.dlut.edu.cn, jianghe@dlut.edu.cn, li1989@mail.dlut.edu.cn

dgl232002@smail.nju.edu.cn, zychen@nju.edu.cn

Abstract—In crowdsourced mobile application testing, crowd workers help developers perform testing and submit test reports for unexpected behaviors. These submitted test reports usually provide critical information for developers to understand and reproduce the bugs. However, due to the poor performance of workers and the inconvenience of editing on mobile devices, the quality of test reports may vary sharply. At times developers have to spend a significant portion of their available resources to handle the low-quality test reports, thus heavily decreasing their efficiency. In this paper, to help developers predict whether a test report should be selected for inspection within limited resources, we propose a new framework named TERQAF to automatically model the quality of test reports. TERQAF defines a series of quantifiable indicators to measure the desirable properties of test reports and aggregates the numerical values of all indicators to determine the quality of test reports by using step transformation functions. Experiments conducted over five crowdsourced test report datasets of mobile applications show that TERQAF can correctly predict the quality of test reports with accuracy of up to 88.06% and outperform baselines by up to 23.06%. Meanwhile, the experimental results also demonstrate that the four categories of measurable indicators have positive impacts on TERQAF in evaluating the quality of test reports.

Index Terms—crowdsourced testing, test reports, test report quality, quality indicators, natural language processing

I. INTRODUCTION

Mobile devices grow dramatically and mobile applications evolve rapidly, posing great challenges to the software test activities. However, due to the typical characteristics of mobile devices, such as limited bandwidth, unreliable networks, and diverse operation systems, traditional testing (e.g., laboratory testing) for desktop applications and web applications may be not intrinsically appropriate to a mobile environment [1]. Recently, many companies or organizations tend to crowdsource their software testing tasks for mobile applications to an undefined, geographically dispersed large group of online individuals (namely crowd workers) in a open call form [2], [3]. Therefore, crowdsourced testing has received wide attention from both academia and industry [4]–[7]. In contrast to traditional testing, crowdsourced testing can be performed anytime and anywhere [8], thus tremendously improving the testing productivity. Meanwhile, crowdsourced testing recruits not only professional testers, but also end users for testing [3]. Developers can gain real feedback information, functional requirements, and user experiences.

In crowdsourced testing, crowd workers from open platforms help developers perform testing and submit test reports for abnormal phenomena [4]. A typical test report usually provides some critical field information, such as *environment*, *input*, *description*, and *screenshot* for developers to understand and fix the bug. One of the most important characteristics is that crowdsourced testing is strictly limited in time, such as several days or one week [4]. Thousands of test reports are sent to developers in a short time and the quantity heavily exceeds the available resources to inspect them. Meanwhile, due to the poor performance of workers and the inconvenience of editing on mobile devices, test reports may differ sharply with respect to their quality, which seriously affects the understandability and reproducibility for developers to fix the bugs.

Many studies focus on shortening the total inspection cost by reducing the quantity of inspected test reports [4], [5], [8], [9]. However, these studies neglect the impact of the quality of test reports on the inspection efficiency. High-quality test reports provide overall information and the contained contents can be easily understood, developers can reproduce and fix the bugs within a reasonable amount of time. In contrast, low-quality test reports often lack of important details and consume developers much time and efforts, thus heavily decreasing their efficiency. It is perfect if the quality of test reports can be reliably measured by automated methods so as to developers select the high-quality test reports for inspection. Although no study has been conducted to investigate how to automatically measure the quality of test reports, some studies around quality assessment for bug reports and requirement specifications have thrown light on a practicable direction by defining a set of indicators to quantify the desirable features or properties of bug reports and requirement specifications [10]–[13].

In this paper, to help developers predict whether a test report can be selected to inspect within limited resources, we attempt to resolve the problem of test report quality assessment by classifying test reports as either “Good” or “Bad”. We propose a new framework named TEst Report Quality Assessment Framework (TERQAF) to automatically model the quality of test reports. First, Natural Language Processing (NLP) techniques are applied to preprocess test reports. Then, we define a series of quantifiable indicators to measure the desirable properties of test reports and determine the corresponding

value of each indicator according to the textual content of each test report. Finally, we transform the numeric value of a single indicator into the nominal value (namely Good, Bad) by means of a step transformation function and aggregate the nominal values of all indicators to predict the quality of test reports.

To evaluate the effectiveness of TERQAF, we perform five crowdsourced test tasks for real industrial mobile applications and collect five datasets with 936 test reports from crowd workers. Developers have spent about one week to inspect and evaluate these test reports. With the help of developers, we form the ground truth for experiments. We employ the commonly used accuracy as the metric and investigate three research questions to evaluate the effectiveness of TERQAF in test report quality assessment. Experimental results show that TERQAF can achieve 88.06% of accuracy in predicting the quality of test reports and outperform baselines by up to 23.06%. Meanwhile, the experimental results also demonstrate that the four categories of measurable indicators have positive impacts on TERQAF in test report quality assessment.

In this study, we make the following contributions:

- 1) To the best of our knowledge, this is the first work to investigate the quality of test reports and resolve the problem of test report quality assessment.
- 2) To automatically model the quality of test reports, we propose a new framework named TERQAF by using a taxonomy of quantifiable indicators to measure the desirable properties of test reports.
- 3) We evaluate TERQAF over five real industrial crowdsourced test report datasets of mobile applications. Experimental results show that TERQAF can accurately predict the quality of test reports.

The rest of this paper is structured as follows. Section II details the background and the motivation. In Section III, we systematically summarize some desirable properties that an expected test report should meet. Section IV defines a taxonomy of indicators for the desirable properties. In Section V, we detail TERQAF for test report quality assessment. The experimental setup and the experimental results are presented in Section VI and Section VII, respectively. Section VIII discusses the threats to validity and Section IX reviews some related work. Finally, we conclude this study in Section X.

II. BACKGROUND AND MOTIVATION

In this section, we introduce the background of crowdsourced testing in detail and present several examples as the motivation for resolving the problem of test report quality assessment.

In crowdsourced testing, companies or organizations are responsible for preparing software under test and testing tasks for crowdsourced testing. Workers passing an evaluation select test tasks according to their mobile devices, perform testing, and edit test reports for the observed abnormal behaviors [4], [5]. These test reports are written in natural language together with some screenshots based on the predefined format. A typical test report is usually composed of different fields, such as *environment*, *input*, *description*, and *screenshot*, some of which may vary slightly in different projects from different

crowdsourced platforms, but are generally similar in the content [8], [9]. In our experiments, we perform five crowdsourced test tasks for mobile applications with our industrial partners on the Kikbug crowdsourced testing platform¹.

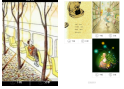

Table I arrays several examples of crowdsourced test reports from the real industrial data. Notably, in our experiments, all test reports are written in Chinese. In order to facilitate understanding, we translate them into English. Field *environment* is the basic configurations of used mobile devices, including phone type, operation system, screen resolution, and system language. Field *input* lists the concrete test steps which are well designed by workers in performing testing based on the actual test requirements. Developers precisely follow these test steps to reproduce the bugs possibly. Field *description* contains the detailed descriptions of bugs and occasionally involves real user experience. By reading the descriptions, developers can understand the content and make an initial decision for fixing the bugs. Field *screenshot* sometimes provides some necessary images to capture the system symptoms when the bugs occur.

However, for crowdsourced mobile application testing, test reports are generally short and uninformative. For example, TR_1 in Table I only contains two words which may make developers confused to understand the bug. Meanwhile, workers do not strictly comply with the given format to write test reports. They may describe their work details or reveal system bugs in Field *input*. For example, the *input* of TR_2 provides the bug description rather than concrete test steps, thus seriously hampering developers to reproduce the bug. At times, for saving time or other motivations, workers may report multiple bugs in the same test report which is called a multi-bug test report. Generally, multi-bug test reports carry more natural language information but relatively marginal for each contained bug. Also, the test steps may be not sufficiently exact to reproduce each bug. For example, TR_3 is a multi-bug test report which reveals two distinct software bugs. Lines 1 to 3 detail that the system does not work well to remind users how to open the downloaded pictures. Line 4 briefs a sharing problem using only two words. Meanwhile, the test steps are not clearly distinguished to reproduce the two bugs.

In aggregate, test report inspection and evaluation are a significant part of mobile application maintenance. However, the widely varied quality of test reports obviously influences the efficiency of developers. In particular, low-quality test reports usually need more time and efforts to understand, thus some test reports are dealt with extremely slowly or not at all constrained by the limited available resources. In practice, test reports usually contain many duplicates. When facing multiple test reports revealing the same bug, developers should select the high-quality one for inspection. In this paper, to help developers predict whether a test report should be selected to inspect, we attempt to resolve the problem of test report quality assessment. Inspired by existing studies around quality assessment for bug reports and requirement specifications, we

¹<http://kikbug.net>

TABLE I: Examples of crowdsourced test report

No.	Environment	Input (test steps)	Description	Screenshot
TR_1	Phone type: Meizu MX 5 Operating System: Android 5.1 Screen Resolution: 5.5 inch System Language: Chinese	Click on the search button at the bottom, select a theme. Horizontally scroll the screen to the end to check the reminder function. Then vertically scroll the screen to the end to check the reminder function.	No prompt function.	
TR_2	Phone type: Gionee GN9000 Operating System: Android 4.4.2 System Language: Chinese Screen Resolution: 4.6 inch	It is failed to share pictures to friends or circle of friends by WeChat.	Sharing operation fails by WeChat	
TR_3	Phone type: Xiaomi MI 4LTE Operating System: Android 4.2.2 System Language: Chinese Screen Resolution: 4.6 inch	1. Click on the category list. 2. Select a category and enter it to view pictures. 3. Horizontally scroll the screen to check the feature of "no more pictures". 4. Download pictures and share pictures.	1. When the pictures are downloaded, the system recommends no applications for opening the downloaded pictures. 2. Sharing problem.	

define a series of quantifiable indicators to measure the quality of test reports based on the textual contents.

In software engineering, requirement engineering is the first stage which captures the expected demands of clients and analyzes what functions the system must achieve [14]–[16]. The requirement engineering processing mainly concentrates on producing and refining the software requirement specifications which play a critical role in determining the overall quality of software. However, manually performed quality evaluation for requirement specifications is a tedious and burdensome task. Therefore, researchers have developed various automated methods and tools to help developers evaluate the quality of requirement specifications. The most generic method is to define a series of indicators to quantify the desirable properties of requirement specifications [11], [12], [17]. For example, size (i.e., text length) directly measures the atomicity of a requirement specification and indirectly reflects the completeness and conciseness. However, some indicators may be not applicable in test reports due to the specific characterizes, such as short text and many images. Therefore, we need to develop new indicators to measure the quality of test reports.

III. DESIRABLE PROPERTIES OF TEST REPORTS

Admittedly, quality is an ambiguous concept which is related to distinct evaluation criteria [11]. Different developers usually have a disagreement over the quality of the same test report. Given a crowdsourced test report, we clearly discriminate qualitative desirable properties based on individually subjective opinions and quantitative quantifiable indicators relying on intrinsically objective characteristics. In this section, we explicitly define what desirable properties we should measure for the quality of test reports.

Since only several studies focus on crowdsourced testing and this study is the first work on investigating the quality of test reports, no referable desirable property is available for test report quality assessment. In practice, test reports can be regarded as special bug reports, the desirable properties of bug reports are also able to adopt to test reports. In the literature, researchers have conducted a series of empirical studies to investigate the quality of bug reports [10], [18], [19]. Some complementary desirable properties are explored, including atomicity, correctness, completeness, conciseness,

understandability, unambiguity, and reproducibility [10], [19]–[22]. We explain these desirable properties as follows in detail:

- Atomicity: A test report reveals only a bug without information of other bugs.
- Correctness: A test report reveals a real bug and each field in a test report provides the specified information.
- Completeness: A test report contains all field information predefined in the format.
- Conciseness: There are not two or more sentences conveying the same information.
- Understandability: The content is correctly understood without difficulty and the described bug can be easily identified from the content.
- Unambiguity: There is only one interpretation for each test report.
- Reproducibility: The provided test steps are effective to reproduce the described bug.

IV. TAXONOMY OF INDICATORS

As mentioned in the literature, quantitative indicators are more or less associated with the qualitative properties [11], thus either directly or indirectly reflecting the quality of textual content. Compared against test reports, requirement specifications contain more contents which are evaluated with more desirable properties, such as traceability, verification, and validation [11]. Existing studies have systematically presented a overall taxonomy of indicators for the quality measurement of requirement specifications. Based on these studies [11], [12] and the typical characteristics of test reports, we define some quantifiable indicators to measure the desirable properties of test reports and classify them into four categories:

- Morphological indicators, such as size and readability, which primarily reveal the surface characteristics of text of a test report without caring the content.
- Lexical indicators, such as the number of imprecise terms, which try to lexically evaluate the textual content of a test report according to the user-defined term lists.
- Analytical indicators, such as the usage of domain terms, which leverage some specific domain terms to semantically analyze the textual contents of test reports.

- Relational indicators, such as itemization, which measure the structured properties of a test report or reveal whether a test report provides each field of information.

In this section, we detail each indicator and clearly clarify what desirable properties it measures. A noticeable point is that test reports consists of four fields of information, namely *environment*, *input*, *description*, and *screenshot*. Since *screenshot* is independent of natural language information meanwhile *environment* can be understood relatively simply, we focus on measuring *description* and *input* which are the most important information for developers to fix the bug. Therefore, the following indicators serve the purpose of evaluating *description* and *input* except for special declarations.

A. Morphological Indicators

Size and readability are the most frequently used and the most intuitive morphological indicators. Besides, punctuation is taken into account.

1) *Size*: In general, size refers to the numbers of characters, words, text lines, sentences, and paragraphs in a document [11], [12]. It directly measures the atomicity and conciseness and indirectly reflects all other properties of test reports. Taking the extremely short text in test reports into consideration, we only adopt the number of characters as the metric. Empirically, long test reports spend much reading time and short test reports lack of necessary details. Therefore, a good test report should keep the size in a moderate length.

2) *Readability*: Readability is an important quality indicator which aims to measure the degree of difficulty of reading a text [11], [12]. In the literature, researchers have conducted extensive studies to investigate the readability of Chinese documents and propose various readability formulas [23]. The first recognized formula for readability measurement is $Y = 14.9596 + 39.07746X_1 + 1.011506X_2 - 2.48X_3$ [23], where X_1 is the ratio of difficult words (excluded by 5,600 commonly used words), X_2 and X_3 denote the number of sentences and the average stroke count, respectively. As for English documents, many readability formulas have been modeled in the literature [24], [25]. For example, the Flesch readability index [24] is calculated by the formula $R_{Flesch} = 206.835 - (1.015 \times WPS) - (84.6 \times SPW)$, where WPS and SPW represent the average number of words within a sentence and the average number of syllables of a word, respectively. Certainly, there exist formulas for measuring the readability of text written in other languages.

3) *Punctuation*: Another representative indicator is punctuation marks [11] which are used to divide a long sentence into relatively short ones to efficiently improve the understandability of test reports. Lacking of punctuation usually leads to the difficulty of understanding a long sentence. Meanwhile, excessive punctuation may cause the missing of semantics. In practice, the number of punctuation marks is directly related to the average length of sentences. When the size of a test report and the expected average length of sentences are given, the number of punctuation marks is consequently determined.

B. Lexical Indicators

Different from morphological indicators, lexical indicators require additional information namely term lists to compute the quality of test reports. Existing studies have summarized overall term lists for the processing of English documents [11], [12]. Aiming at Chinese documents, we reuse the term lists and translate them into Chinese. Meanwhile, we add some synonyms to form the eventual term lists for Chinese. In practice, these terms can be easily defined and collected.

1) *Imprecise Terms*: When describing a bug, due to the uncertainty, workers tend to use imprecise terms which convey ambiguous information, thus introducing risks to the understanding of the bug. Therefore, prohibitively using imprecise or subjective terms can effectively improve the understandability of test reports. Based on the characteristics of imprecise terms, we partition them into six different groups [11]:

- Quality: Good, bad, moderate, medium, efficient, etc.
- Quantity: Enough, abundant, massive, sufficient, etc.
- Frequency: Generally, usually, typically, almost, etc.
- Enumeration: Several, multiple, some, few, little, etc.
- Probability: Possibly, can, may, perhaps, optionally, etc.
- Usability: Experienced, familiar, adaptable, easy, etc.

2) *Anaphoric Terms*: Anaphoric terms refer to the words which are used to replace the frequently used or complicated words or phrases in the same text. Typical anaphoric terms include personal pronouns (e.g., it, them), relative pronouns (e.g., that, which, where), demonstrative pronouns (e.g., this, that, these, those) [11], etc. Although the usage of anaphoric terms is grammatically acceptable, they would impose a threat of imprecisions and ambiguities to the quality of test reports with respect to the understandability. Ideally, a test report should be explicit without an anaphoric term.

3) *Directive Terms*: Differing from imprecise terms and anaphoric terms, directive terms are an positive indicator to help developers understand test reports better [12]. In general, the words following directive terms probably add a concrete example or provide more additional information from a complementary aspect to illustrate and strengthen the bug description. In some cases, a high ratio of the usage of directive terms is conducive to the understanding of bugs. In crowdsourced test reports, the commonly used directive terms include “e.g.”, “i.e.”, “for example”, “note”, etc.

C. Analytical Indicators

Analytical indicators aim to implement the textual analysis based on the contents of test reports. Existing studies for requirement specification quality assessment show that verbal tense and mood terms as well as domain terms are adopted as analytical indicators [11]. However, test reports written in Chinese contain no verbal tense and mood terms, we only consider domain terms in this study.

Domain terms are typically organized either as a simple glossary of terms or in complicated structural form such as thesauri or ontologies [11]. They are usually used to reflect the correctness and the completeness. An obvious problem is that

TABLE II: The corresponding relationship between measurable indicators and desirable properties

Category	Indicator	Desirable property						
		Atomicity	Correctness	Completeness	Conciseness	Understandability	Unambiguity	Reproducibility
Morphological	size	X	•	•	X	•	•	•
	Readability					X	•	
	Punctuation					X	•	
Lexical	Imprecise term		X	•	•	•	X	
	Anaphoric terms		X		•	X	X	•
	Directive terms		•		•	•	X	
Analytical	Negative terms	X	X				•	
	Behavior terms	X	X		•			
	Action terms		X		•		•	X
	Interface elements		X		•	•	•	X
Relational	Itemizations	X	•	•	•	X	•	X
	Environment		•	X		•		X
	Screenshots		•	X		X	•	•

domain terms may differ immensely in different areas, thus it is hard to define and acquire the domain terms for a specific area. Fortunately, some studies have summarized overall lists of domain terms for bug report quality assessment [10], [26]. In this paper, we reuse the term lists defined by Ko et al. [26] and combine negative terms to measure the quality of test reports. When a test report contains a corresponding domain term, the indicator is considered as Good.

1) *Negative Terms*: For mobile applications, an incorrect or missing function will cause a software bug. Therefore, in test reports, workers usually use some negative terms to describe the lacking of system functions, such as no, not, fail, lack, etc.

2) *Behavior Terms*: A software bug is an error, defect, failure, or fault in a computer program or system. Therefore, the descriptions of test reports generally contain the terms which represent the observed behaviors, such as error, bug, defect, problem, failure, etc.

3) *Action Terms*: To be exact, test steps are a series of actions which are triggered by clicking on users interfaces (e.g., button, menu) of mobile applications in perform testing. Hence, test steps should contain some action terms, such as open, select, click, enter, check, etc.

4) *Interface Elements*: If the *input* of a test report is related to test steps, it must contain some action terms followed by corresponding user interface elements, such as button, toolbar, menu, dialog, window, etc.

Notably, negative terms and behavior terms adopt to the *descriptions* of test reports, while action terms and interface elements focus on the *inputs*.

D. Relational Indicators

A test report is a structural document composed of multiple fields of which each provides specified information for developers to understand or reproduce the bug. Ideally, a complete test report contains all field information and the content in each field is correct and relevant. For example, field *input* should provide detailed test steps for reproducing the revealed bug. However, the same as bug reports [10], workers are hard to provide test steps and screenshots for developers in test reports. Therefore, we use the following three indicators to detect the correctness and completeness of test reports.

1) *Itemizations*: Test steps are usually orderly itemized with itemizations or enumerations (such as TR_2 and TR_3 in Table I). Instead, if the *input* in a test report is organized with itemizations or enumerations, it is related to test steps with a very high probability. To distinguish itemizations in test reports, we search the lines starting with an itemization character, e.g., “.”, “*”, or “+”. Similarly, we also identify enumerations by detecting each line whether it starts with numbers or serial numbers in which numbers are enclosed by parentheses or brackets or followed by a single punctuation character [10].

2) *Environment*: In test reports, *environment* is a structured field which describes the basic configuration of a mobile device. By reading the *environment*, developers can clearly realize the adaptations of mobile applications. Therefore, a complete test report should provide environment information. Compared against other fields, the field *environment* can be easily detected by the specific keywords, namely phone type, operation system, screen resolution, and system language.

3) *Screenshots*: A complete test report provides not only natural language information but also screenshots. At times screenshots can effectively help developers realize the system states when the bug occurs. In crowdsourced test reports, screenshots are uniformly enclosed in attachments. Different from bug reports, the attachments for test reports contain no text. Therefore, we easily identify screenshots without any complex tool, i.e., if detecting an attachment, we recognize the test report with screenshots.

In summary, the above defined indicators can measure the desirable properties of test reports to some extent. Based on existing studies [11] and our experience, we summarize the corresponding relationship between quantifiable indicators and desirable properties in Table II, where “X” represents a direct influence and “•” denotes an indirect influence between the indicator and the desirable property. For example, the indicator readability directly influence the understandability of test reports and indirectly reflect the unambiguity.

V. TEST REPORT QUALITY ASSESSMENT FRAMEWORK

In this section, we detail TERQAF shown in Fig.1. TERQAF is composed of three components, namely pre-

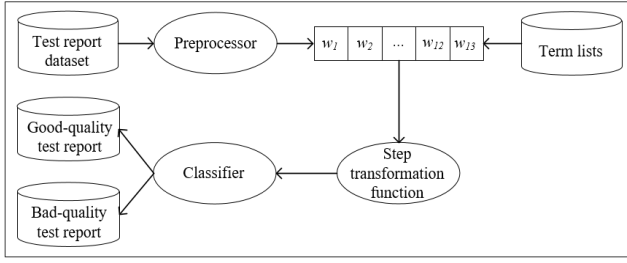


Fig. 1: TERQAF framework.

processor, step transformation function, and classifier which are used to preprocess crowdsourced test reports by NLP techniques, transform the numerical values of indicators into nominal values using step transformation functions, and classify test reports as either “Good” or “Bad”, respectively.

A. Preprocessor

Since test reports are almost written in Chinese and lexical indicators and analytical indicators analyze test reports based on segmented text, we need a Chinese NLP tool to implement the word segmentation for test reports. Fortunately, there are many efficient NLP tools for the processing of Chinese documents, such as Language Technology Platform (LTP)², ICT-CLAS³, and IKAnalyzer⁴. In this study, we adopt IKAnalyzer which can provide efficient services and is widely applied in many studies [27]. Certainly, different natural languages have different characteristics which require different processing. For example, English words are naturally segmented by spaces and presented in different grammatical forms, stemming is the most important step to process English documents.

Next, we proceed to process the test reports and extract the defined indicators based on the segmented text to generate a vector for each test report in which each element is a numerical value of the corresponding indicator. Since the size of *input* varies in a large range, a good *input* is hard to determine. Therefore, size refers to the textual length of a *description*.

B. Step Transformation Function

Once the vector is generated, we need to determine the quality level of each quantifiable indicator by the numerical values in this component. In general, quality measurement is classified into two discrete levels: Good, Bad; High, Low; Cheap, Costly; etc. [11]. To transform the numerical values of indicators into nominal values (namely Good, Bad), we introduce step transformation functions.

Typically, step transformation functions are divided into four major kinds: increasing, decreasing, convex, and concave [11], where the first two step functions involve one parameter while the last two contain two parameters. As shown in Fig. 2, four kinds of step functions can regularize heterogeneous numerical indicators and transform their values into corresponding nominal values by setting suitable bounds. For example, a good

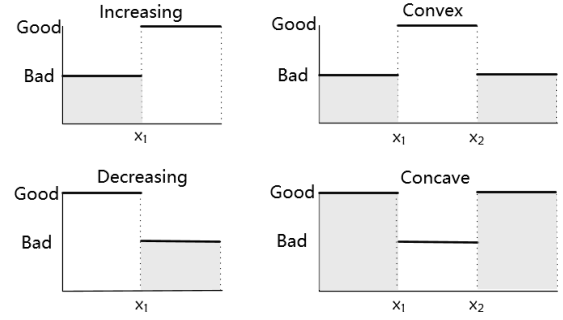


Fig. 2: Four kinds of step transformation functions.

TABLE III: The corresponding step function and interval setting for each measurable indicator

Category	Indicator	Function	Intervals
Morphological	Size	Convex	$0-x_1-x_2$
	Readability	Convex	$0-x_3-x_4$
	Punctuation	Convex	$0-x_5-x_6$
Lexical	Imprecise term	Decreasing	0-1
	Anaphoric terms	Decreasing	0-1
	Directive terms	Increasing	0-1
Analytical	Negative terms	Convex	0-1-2
	Behavior terms	Convex	0-1-2
	Action terms	Increasing	0-1
	Interface elements	Increasing	0-1
Relational	Itemizations	Increasing	0-1
	Environment	Increasing	0-1
	Screenshots	Increasing	0-1

test report should keep the size of text neither too long nor too short. Therefore, the indicator size needs a convex step function to implement the transformation. If the numerical value is between x_1 and x_2 , the size is considered as Good. In contrast, if the value is smaller than x_1 or greater than x_2 , the size is considered as Bad. Again, imprecise terms use an increasing function, if the value is greater than x_1 , the indicator is considered as Good. Otherwise, the indicator is considered as Bad. Similarly, other indicators can also leverage one of four kinds of step functions to implement the transformation. However, the crucial difficulty is how to exactly set the parameter values for each indicator.

Table III lists the corresponding step transformation function for each indicator and presents the parameter settings. In the table, the intervals are defined as closed in the lower bound and open in the upper bound. For example, $0-x_1-x_2$ represents three intervals $[0, x_1)$, $[x_1, x_2)$, and $[x_2, \infty)$. Obviously, the increasing and convex step functions are the most frequently used for the defined indicators. Meanwhile, many indicators use either increasing or decreasing step functions and their intervals are defined by 0-1. Given that multi-bug test reports are usually regarded as low-quality ones which may be identified by multiple negative terms or behavior terms based on their *descriptions*. Therefore, negative terms and behavior terms use convex step functions and their parameter intervals are set to 0-1-2. Moreover, morphological indicators require convex step functions and the parameters can be determined empirically. For example, by an in-depth investigation on test reports, the

²<http://www.ltp-cloud.com/>

³<http://ictclas.nlpir.org/>

⁴<http://www.oschina.net/p/ikalyzer>

TABLE IV: Five crowdsourced test report datasets

Dataset	Version	# R	# B	# R_m	# R_g	# R_b
UBook	2.1.0	201	30	53	89	112
JustForFun	1.8.5	230	25	55	92	138
SE-1800	2.5.1	201	32	35	62	139
iShopping	2.5.1	215	65	28	56	159
CloudMusic	1.3.0	89	21	8	37	52
Totals		936	173	179	336	600

moderate size of a *description* may be 15 to 30 characters. As for punctuation, we replace it with average sentence length since its parameters can be defined empirically. Based on the style of Chinese, a moderate sentence should keep the length between 8 to 15 characters. To gain the appropriate parameter values, we will experimentally tune the parameters for morphological indicators.

C. Classifier

By using step functions, the numerical values of quantifiable indicators are transformed into corresponding nominal values. Next, we leverage the classifier to predict the quality of test reports by aggregating the nominal values of all indicators. In fact, a good test report does not mean all indicators with Good results. Meanwhile, in crowdsourced testing, due to the poor performance of workers, the number of high-quality test reports is obvious less than that of low-quality test reports in the same dataset. Therefore, we consider a test report as Good if there are at least 60% (which is determined by experiments) of indicators with the result Good.

VI. EXPERIMENTAL SETUP

In this section, we detail the experimental setup, including experimental platform, evaluation metrics, and experimental datasets.

A. Experimental Platform

All the experiments are conducted with Java JDK 1.8.0_60, compiled with Eclipse 4.5.1, and run on a PC with 64-bit Windows 8.1, Intel Core i7-4790 CPU, and a 8G memory.

B. Evaluation Metrics

In quality assessment for requirement specifications, accuracy is generally adopted to evaluate the effectiveness of automated methods. In this study, we also employ accuracy as the metric to evaluate the performance of TERQAF. Assuming that m represents the number of test reports which are correctly predicted by TERQAF and n is the number of test reports in the dataset, the calculation formula is as follow:

$$Accuracy = \frac{m}{n} \quad (1)$$

C. Experimental Datasets

From October 2015 to January 2016, crowdsourced test tasks are performed with our industrial partners for the five mobile applications, including UBook, Justforfun, CloudMusic, SE-1800, and iShopping. The brief descriptions for the five applications are presented as follows:

- *UBook*: An online education application developed by New Orientation.
- *JustForFun*: An interesting photo sharing application developed by Dynamic Digit.
- *CloudMusic*: A music playing and sharing application developed by NetEase.
- *SE-1800*: An electrical monitoring application developed by Panneng.
- *iShopping*: An online shopping guideline App developed by Alibaba.

In our experiments, workers are recruited and evaluated by the Kikbug platform, a specialised crowdsourced Android testing platform. The workers are mostly students who have already acquired some knowledge of software testing and have completed some preliminary tasks recorded in our platform, i.e., they are qualified for the testing task. For each testing task, we stipulate the testing time within 2 weeks. Students elaborately design test steps to perform testing according to the test requirements prescribed by developers. When detecting a bug, workers are required to write a test report to describe the abnormal behavior in descriptive natural language together with some necessary screenshots on their mobile devices. These test reports are submitted to the Kikbug platform with a small application installed in the mobile devices.

We collect five datasets and invite 15 developers who develop the applications to evaluate the test reports. Each test report is scored by three developers on a scale of 100 points based on some predefined criteria. If the average score of a test report is greater than 60, it is marked as “Good”, otherwise, it is marked as “Bad”. The workers who submitted good quality test reports would be financially compensated. With the help of developers, we gain the detailed statistical information about the five crowdsourced datasets, as shown in Table IV, where $\#R$ represents the number of test reports, $\#B$ is the number of revealed bugs in the dataset, R_m denotes the number of multi-bug test reports, R_g and R_b are the numbers of good and bad quality test reports, respectively. Five datasets contain 89, 92, 62, 56, and 37 good quality test reports, and 112, 138, 139, 159, and 52 bad quality test reports, respectively.

VII. EXPERIMENTAL RESULTS

In this section, we investigate three research questions to verify the performance of TERQAF.

A. Investigation to RQ1

RQ1. How do the parameters impact the performance of TERQAF?

In TERQAF, we leverage step transformation functions to transform the numerical values of indicators into the corresponding nominal values. However, step functions usually require appropriate parameter settings. Fortunately, only the parameters of morphological indicators are hard to determined. In this RQ, we mainly focus on the parameter tuning of the morphological indicators (namely size, readability, and punctuation) and try to seek suitable parameter values which can be applied to all datasets.

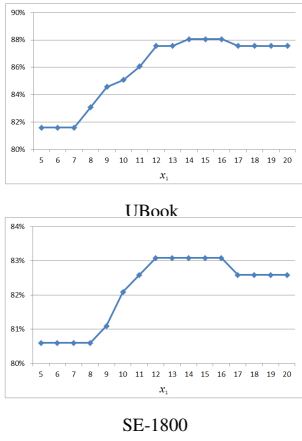


Fig. 3: Results of TERQAF with different x_1 values.

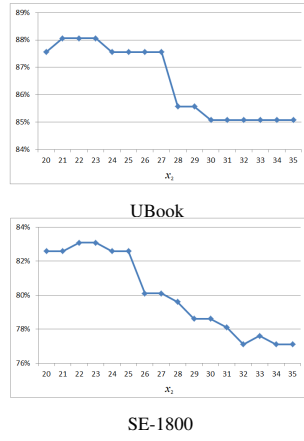


Fig. 4: Results of TERQAF with different x_2 values.

In this experiment, we tune the parameters of one indicator and fix the parameters of other indicators in default values. Given that the details of parameter tuning are the same, we take the indicator size as an example to present the tuning details. Size involves two parameters x_1 and x_2 . Our investigation on test reports over the UBook dataset shows that the minimum, average, maximum size of the *descriptions* are 4, 18.4, and 46. Thus we predefine that x_1 varies from 5 to 20 and x_2 varies from 20 to 35. In this experiment, we set a tuning step to 1 and run TERQAF over UBook and SE-1800 datasets. We gradually change the value of one parameter and keep the other one unchanged.

Fig. 3 uses two sub-figures to present the tuning results with respect to x_1 over the UBook and SE-1800 datasets, respectively. Obviously, TERQAF achieves different accuracy results with the growth of x_1 . When x_1 changes from 5 to 20, TERQAF first increases slowly and then turns to be stable over the UBook dataset. When x_1 is between 14 and 16, TERQAF achieves the best result 88.06% of accuracy. Similarly, TERQAF presents the basically similar trends over the SE-1800 dataset and achieves the best result when x_1 is between 12 and 16. Combining the two sub-figures, when x_1 is set to 14, 15, or 16, TERQAF achieves the best accuracy results over the two dataset, which indicates that $x_1=14$ may be a good choice. Fig. 4 also uses two sub-figures to present the tuning results when x_2 varies from 20 to 35 and x_1 is equal to 14 over the UBook and SE-1800 datasets, respectively. We observe that TERQAF achieves the best result when x_2 falls between 21 and 23 over the UBook dataset. When x_2 is greater than 23, TERQAF shows a strong downward trend. As to the SE-1800 dataset, when x_2 is between 22 and 23, TERQAF achieves the best result 83.08% of accuracy. Therefore, $x_2=22$ may be a feasible choice. In addition, we also observe that TERQAF achieves the same result in some continuous values with respect to x_1 and x_2 , which indicates that TERQAF is not very sensitive to x_1 and x_2 .

In summary, TERQAF achieves different results about different parameter values. We implement the same strategy to

tune the parameters for the indicators readability and punctuation. Based on the results, the appropriate parameter values are $x_1=14$, $x_2=22$, $x_3=-4$, $x_4=10$, $x_5=7$, and $x_6=11$ which approximate the empirical values. In following experiments, we use them as the default parameter values.

B. Investigation to RQ2

RQ2. Can TERQAF outperform some baseline methods in measuring the quality of test reports?

In this study, we explore a new framework named TERQAF to automatically quantify the quality of test reports. Since this work is the first study to investigate test report quality, no state-of-the-art technique is available to validate the effectiveness of TERQAF. Given that test reports can be viewed as special bug reports, we select CUEZILLA [10], an automated tool for measuring the quality of bug reports, as a baseline method. Also, we compare TERQAF with the WORST method, which predicts all test reports as either Good or Bad. In this RQ, we experimentally investigate whether TERQAF can outperform CUEZILLA and WORST.

Similarly, CUEZILLA measures the quality of bug reports based on seven desired features [10]) of the contents. However, test reports provide no code samples, stack traces, and patches. Hence, we only focus on itemizations, keyword completeness (including action items, expected and observed behaviors, steps to reproduce, build-related, and user interface elements [10]) to measure the quality of test reports. We adopt the same method to calculate the numerical values of indicators for CUEZILLA. As for WORST, if the number of good quality test reports is smaller than that of bad quality test reports, all test reports are predicted as Bad. Otherwise, all test reports are predicted as Good.

Table V shows the accuracy results of different methods over all datasets. As shown in the table, we observe that TERQAF outperforms WORST and CUEZILLA over all datasets but SE-1800. For example, TERQAF achieves 88.06% of accuracy over the UBook dataset and improves WORST and CUEZILLA by 32.34% and 9.85%, respectively. The potential reason is that TERQAF uses more indicators to measure the quality of test reports and the results obtained by TERQAF are not influenced by a single indicator. In contrast, CUEZILLA uses only several desired features to predict the quality of test reports and thus a single feature may have great impact on its performance. Nevertheless, CUEZILLA achieves higher accuracy than WORST over all datasets. For example, CUEZILLA achieves 86.07% of accuracy over the SE-1800 and improves WORST by up to 12.48%. Surprisingly, CUEZILLA achieves better result than TERQAF over this dataset. This may be that some of the defined indicators do not work well to measure the quality of test reports over this dataset. This fact also demonstrate that not all indicators adapt to all datasets. That is, the same indicator may be ineffective in this dataset but effective in another dataset.

In summary, TERQAF works well in test report quality assessment and significantly outperforms both WORST and CUEZILLA.

TABLE V: Accuracy results of different methods over all datasets

Dataset	TERQAF	WORST	CUEZILLA
UBook	88.06%	55.72%	78.11%
JustForFun	80.43%	60.00%	75.65%
CloudMusic	86.52%	69.15%	70.79%
SE-1800	83.08%	73.95%	86.07%
iShopping	82.79%	41.57%	82.33%

C. Investigation to RQ3

RQ3. How do the four categories of indicators impact the performance of TERQAF?

In TERQAF, we define a series of indicators and classify them into four categories to measure the quality of test reports based on the textual content from different perspectives. Undoubtedly, different categories of indicators have different impacts on the performance of TERQAF. Also, we are not clear whether each category plays a positive role in evaluating the quality of test reports and which category is the most important. In this RQ, we experimentally investigate the impacts of four categories of indicators on the quality measurement of test reports.

Empirically, to investigate the impact of each category of indicators, we should leverage all indicators of the category to independently evaluate the quality of test reports. However, each single category only includes several indicators which may make a bias. Therefore, based on a complementary ideal, we leverage the other three categories of indicators to perform quality assessment for test reports to validate the effects of each category of indicators. In such a way, four variant methods are produced and named as TERQAF-LAR, TERQAF-MAR, TERQAF-MLR, and TERQAF-MLA for convenience, where M, L, A, and R represent the morphological, lexical, analytical, and relational indicators, respectively.

Table VI presents the results of different methods over the five datasets. Obviously, TERQAF outperforms TERQAF-LAR, TERQAF-MAR, TERQAF-MLR, and TERQAF-MLA over all datasets but JustForFun, which indicates that four categories of indicators have good effects on quality assessment for test reports. For example, TERQAF achieves 88.06% of accuracy over the UBook dataset and improves TERQAF-LAR, TERQAF-MAR, TERQAF-MLR, and TERQAF-MLA by up to 7.96%, 0.50%, 6.47%, and 22.89%, respectively. In particular, TERQAF-MLA achieves the poorest results over all datasets, which demonstrates that the relational indicators play the most important role in determining the quality of test reports. The potential reason is that the relational indicators reveal whether a test report is complete or not. In contrast, TERQAF-MAR outperforms TERQAF-LAR, TERQAF-MLR, and TERQAF-MLA over all datasets but JustForFun. For example, TERQAF-MAR achieves 86.52% of accuracy over the CloudMusic dataset and improves TERQAF-LAR, TERQAF-MLR, and TERQAF-MLA by up to 7.87%, 10.12%, and 11.24%, respectively. The fact shows that the lexical indicators work poorly to evaluate the quality of test reports. The reason may be that most of test reports contain no defined lexical

TABLE VI: Impacts of different categories of indicators on TERQAF

Dataset	TERQAF-LAR	TERQAF-MAR	TERQAF-MLR	TERQAF-MLA	TERQAF
UBook	80.10%	87.56%	81.59%	65.17%	88.06%
JustForFun	66.52%	80.87%	82.61%	56.09%	80.43%
CloudMusic	78.65%	86.52%	76.40%	75.28%	86.52%
SE-1800	81.09%	82.59%	82.59%	67.66%	82.59%
iShopping	80.93%	82.33%	81.40%	73.02%	82.79%

terms. Comparatively, TERQAF-MLR outperforms TERQAF-LAR over all datasets but CloudMusic, which indicates that the morphological indicators are more important than the analytical indicators in test report quality assessment.

In summary, different categories of indicators contribute differently for quality assessment of test reports. The relational indicators are the most important and the lexical indicators work poorly. Moreover, the role of morphological indicators outperforms that of analytical indicators.

VIII. THREATS TO VALIDITY

In this section, we discuss the threats to validity, including parameter settings, natural language selection, and term lists. **Parameter settings.** In TERQAF, step transformation functions are leveraged to transform the numerical values of indicators into nominal values. However, for each indicator, the corresponding step function needs to specify the value of each parameter, which may produce a bias in different projects. However, we have conducted a detailed experiment to tune the parameters over two projects and TERQAF can achieve good results over all datasets with the tuned parameters. In addition, the determined parameter values approximate the empirical values. Therefore, this bias is minimized.

Natural language selection. In our experiments, all test reports are written in Chinese and TERQAF is developed based on Chinese text. As well known, Chinese is very different from English and other Latin languages, which may generate a threat to the generalization of TERQAF to other natural languages. However, TERQAF mainly extracts the defined indicators from the text of test reports by conducting simple string matching. It is still a natural language technique which has been proven to adopt to other natural languages [28]. Therefore, this threats will be greatly reduced.

Term lists. In quality assessment, some indicators (such as lexical indicators) depend on the term lists defined by users, which may impact the effectiveness of TERQAF in test report quality assessment. However, many studies [11], [26] have summarized the overall term lists and they have been reused in some studies [10], [12]. In this paper, we fully use these term lists and transform them into Chinese. Meanwhile, we add some relevant terms by an extensive investigation on test reports. Thus, the impact is negligible.

IX. RELATED WORK

In this section, we review some studies related to our work, including crowdsourced testing and quality assessment for textual documents.

A. Crowdsourced Testing

The concept of crowdsourcing refers to the process of an organization crowdsourcing their work to undefined, geographically dispersed online individuals in an open call form [2], [3]. By combining human and machine power, crowdsourcing achieves a rapid resolution for large-scale tasks.

As a newly emerging technique, researchers have conducted extensively empirical studies to investigate the potentials of crowdsourced testing [29], [30]. For example, Guaiani and Muccini perform an empirical evaluation to demonstrate that crowdsourced testing can complement traditional laboratory testing [29]. Many studies also apply crowdsourced testing to resolve software engineering problems [4]–[7], [31]–[33]. For example, Gomide et al. propose an event detection algorithm for crowdsourcing software usability testing by processing the actions from mouse movements or touch events to identify user emotions [32]. Considering the wide popularity of mobile devices, Sun et al. try to collect as much as possible service QoS data by a mobile crowdsourcing based testing framework which invokes web service from mobile devices [34].

In contrast, some studies concentrate on resolving crowdsourced testing problems [4], [5], [35], [36]. Starov develops a cloud testing of mobile system framework to achieve efficient crowdsourced mobile application testing by providing cloud services [35]. To improve the quality of crowdsourced testing, Chen et al. introduce crowdsourced testing to education platforms and propose Quasi-Crowdsourced Testing (QCT) [36]. Given that developers have no plenty of time to inspect each test report, Feng et al. leverage a text-based technique and image understanding to prioritize test reports to help developers detect more bugs when inspecting a given number of test reports [4], [8]. To reduce unnecessary inspection, Wang et al. attempt to identify the false positives from raw test reports by adopting both a cluster-based classification approach [9] and active learning [5]. Similarly, our study aims to resolve crowdsourced testing problems.

B. Quality Assessment for Textual Documents

Quality is a ambiguous concept which depends on individually subjective judgements. Different users may evaluate differently over the same textual document. Many studies have been conducted to investigate the quality of textual documents, such as bug reports and requirement specifications.

In software maintenance, bug reports are one of the most important resources for developers to improve the software [37]–[42]. However, low-quality bug reports usually need much time to inspect. Zimmermann et al. have conducted an empirical study to investigate what makes a good bug report. They define seven desired features to detect the quality of bug reports [10]. Meanwhile, some researchers have investigated the role of duplicate bug reports. Evidence show that duplicates can provide additional useful information [19]. To determine whether a bug report should be selected to inspect, Hooimeijer and Weimer propose a descriptive model to evaluate the quality of bug reports based on external features [21]. Taking a great amount of text in bug reports, another

body of studies try to summarize bug reports so that developers consult the short and concise summaries instead of the entire bug reports [20].

In requirement engineering, tremendous efforts are devoted to produce and refine the requirement specifications for the quality assurance of a software system [14]. Erroneous requirements that are not detected timely may cause severe problems. Under this motivation, researchers attempt to automate the quality assessment for requirement specifications [1], [17]. Génova et al. define a taxonomy of measurable indicators to quantify the desirable properties of textual requirements [11]. Parra et al. apply rule induction techniques to assess the quality of requirements by leveraging the historical requirements classified by the expert [15]. In contrast, some studies focus on evaluating a single desirable property of requirement specifications, such as ambiguities [43], [44], inconsistencies [45], and conflicts [46]. Similarly, we attempt to define some quantifiable indicators to measure the quality of test reports based on textual contents.

X. CONCLUSION

In this paper, to help developers predict whether a test report can be inspected in a small amount of time, we try to model the quality of test reports by classifying them as either “Good” or “Bad”. We propose a new method towards resolving the problem of test report quality assessment by defining a series of indicators. Given a test report, the numerical value of each indicator is determined according to the textual content. Then, we aggregate numerical values of all indicators to form a vector and use step transformation functions to transform the numerical values into nominal values to determine the quality of test reports. Experiments are conducted over five crowdsourced test report datasets of mobile applications. Experimental results show that the proposed method can predict the quality of test reports with high accuracy.

Although quantifiable indicators can measure the quality of test reports from different aspects to some extent, quality assessment still depends on manual judgments. Nonetheless, this study not only helps developers automatically evaluate the quality of test reports, but also provides some improvement strategies for workers to write a good test report. For example, test reports should contain more domain terms and exclude ambiguous terms. In future, we try to seek more indicators to measure the quality of test reports and investigate the impact of each indicator. Meanwhile, we will develop a tool implementing our approach to measure the quality of test reports and deploy it in a real scenario, such as the National Student Contest of Software Testing in China⁵.

ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China under Grants No. 61370144, 61722202, 61403057, and 61772107, and Jiangsu Prospective Project of Industry-University-Research under Grant No. BY2015069-03.

⁵<http://moocetest.org/>

REFERENCES

- [1] S. Zogaj, U. Bretschneider, and J. M. Leimeister, "Managing crowd-sourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary," *Journal of Business Economics*, vol. 84, no. 3, pp. 375–405, 2014.
- [2] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.
- [3] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *RN*, vol. 15, no. 01, 2015.
- [4] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu, "Test report prioritization to assist crowdsourced testing," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE'15. ACM, 2015, pp. 225–236.
- [5] J. Wang, S. Wang, Q. Cui, and Q. Wang, "Local-based active classification of test report to assist crowdsourced testing," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'16. ACM, 2016, pp. 190–201.
- [6] E. Dolstra, R. Vliegendorst, and J. A. Pouwelse, "Crowdsourcing gui tests," in *Sixth IEEE International Conference on Software Testing, Verification and Validation*, ser. ICST'13. IEEE, 2013, pp. 332–341.
- [7] M. Nebeling, M. Speicher, M. Grossniklaus, and M. C. Norrie, "Crowd-sourced web site evaluation with crowdstudy," in *Proceedings of 12th International Conference on Web Engineering*, ser. ICWE'12. Springer, 2012, pp. 494–497.
- [8] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective test report prioritization using image understanding," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'16. ACM, 2016, pp. 202–213.
- [9] J. Wang, Q. Cui, Q. Wang, and S. Wang, "Towards effectively test report classification to assist crowdsourced testing," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, pp. 6:1–6:10.
- [10] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What makes a good bug report?" *IEEE Trans. Software Eng.*, vol. 36, no. 5, pp. 618–643, 2010.
- [11] G. Génova, J. M. Fuentes, J. L. Morillo, O. Hurtado, and V. Moreno, "A framework to measure and improve the quality of textual requirements," *Requir. Eng.*, vol. 18, no. 1, pp. 25–41, 2013.
- [12] N. Carlson and P. A. Laplante, "The NASA automated requirements measurement tool: a reconstruction," *ISSE*, vol. 10, no. 2, pp. 77–91, 2014.
- [13] W. Wilson, L. Rosenberg, and L. Hyatt, "Automated quality analysis of natural language requirement specifications in proc," in *Fourteenth Annual Pacific Northwest Software Quality Conference, Portland OR*, 1996.
- [14] P. Heck and A. Zaidman, "A systematic literature review on quality criteria for agile requirements specifications," *Software Quality Journal*, pp. 1–34, 2016.
- [15] E. Parra, C. Dimou, J. L. Morillo, V. Moreno, and A. Fraga, "A methodology for the classification of quality of requirements using machine learning techniques," *Information & Software Technology*, vol. 67, pp. 180–195, 2015.
- [16] R. Thakurta, "A framework for prioritization of quality requirements for inclusion in a software project," *Software Quality Journal*, vol. 21, no. 4, pp. 573–597, 2013.
- [17] L. Rosenberg and T. Hammer, "A methodology for writing high quality requirement specifications and for evaluating existing ones," *NASA Goddard Space Flight Center, Software Assurance Technology Center*, 1999.
- [18] 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*, ser. FSE'08. ACM, 2008, pp. 308–318.
- [19] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?" in *24th IEEE International Conference on Software Maintenance*, ser. ICSM'08, 2008, pp. 337–345.
- [20] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Trans. Software Eng.*, vol. 40, no. 4, pp. 366–380, 2014.
- [21] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, ser. ASE'07. ACM, 2007, pp. 34–43.
- [22] M. E. Joorabchi, M. MirzaAghaei, and A. Mesbah, "Works for me! characterizing non-reproducible bug reports," in *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings*, ser. MSE'14, 2014, pp. 62–71.
- [23] S.-j. Yang, *A readability formula for Chinese language*. University of Wisconsin–Madison, 1970.
- [24] R. Flesch, "A new readability yardstick," *Journal of applied psychology*, vol. 32, no. 3, p. 221, 1948.
- [25] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," Naval Technical Training Command Millington TN Research Branch, Tech. Rep., 1975.
- [26] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006)*, ser. VL/HCC'06. IEEE Computer Society, 2006, pp. 127–134.
- [27] R. Zhang, Q. Zeng, and S. Feng, "Data query using short domain question in natural language," in *2010 IEEE 2nd Symposium on Web Society*, ser. SWS'10. Beijing, China: IEEE Computer Society, 2010, pp. 351–354.
- [28] X. Zhou, X. Wan, and J. Xiao, "Cminer: Opinion extraction and summarization for chinese microblogs," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1650–1663, 2016.
- [29] F. Guaiani and H. Muccini, "Crowd and laboratory testing, can they co-exist? an exploratory study," in *2nd IEEE/ACM International Workshop on CrowdSourcing in Software Engineering*, ser. CSI-SE'15. In ACM/IEEE, 2015, pp. 32–37.
- [30] D. Liu, M. Lease, R. Kuipers, and R. G. Bias, "Crowdsourcing for usability testing," *American Society for Information Science and Technology*, vol. 49, no. 1, pp. 332–341, 2012.
- [31] K. Chen, C. Wu, Y. Chang, and C. Lei, "A crowdsorceable qoe evaluation framework for multimedia content," in *Proceedings of the 17th International Conference on Multimedia 2009*. ACM, 2009, pp. 491–500.
- [32] V. H. Gomide, P. A. Valle, J. O. Ferreira, J. R. Barbosa, A. F. Da Rocha, and T. Barbosa, "Affective crowdsourcing applied to usability testing," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 575–579, 2014.
- [33] Y. Tung and S. Tseng, "A novel approach to collaborative testing in a crowdsourcing environment," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2143–2153, 2013.
- [34] H. Sun, W. Zhang, M. Yan, and X. Liu, "Recommending web services using crowdsourced testing data," in *Crowdsourcing*. Springer, 2015, pp. 219–241.
- [35] O. Starov, *Cloud platform for research crowdsourcing in mobile testing*. East Carolina University, 2013.
- [36] Z. Chen and B. Luo, "Quasi-crowdsourcing testing for educational projects," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE'14. ACM, 2014, pp. 272–275.
- [37] N. Nazar, H. Jiang, G. Gao, T. Zhang, X. Li, and Z. Ren, "Source code fragment summarization with small-scale crowdsourcing based features," *Frontiers of Computer Science*, vol. 10, no. 3, pp. 504–517, 2016.
- [38] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering*, ser. SEKE'10. CA, USA: IEEE Computer Society, 2010, pp. 209–214.
- [39] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, 2015.
- [40] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *34th International Conference on Software Engineering*, ser. ICSE'12. Zurich, Switzerland: IEEE Computer Society, 2012, pp. 25–35.
- [41] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone-based multilevel algorithm," *IEEE Trans. Software Eng.*, vol. 38, no. 5, pp. 1195–1212.
- [42] H. Jiang, J. Zhang, H. Ma, N. Nazar, and Z. Ren, "Mining authorship characteristics in bug repositories," *SCIENCE CHINA Information Sciences*, vol. 60, no. 1, p. 12107, 2017.
- [43] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language

- requirements specifications,” *Requir. Eng.*, vol. 13, no. 3, pp. 207–239, 2008.
- [44] D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry, “Reducing ambiguities in requirements specifications via automatically created object-oriented models,” in *Innovations for Requirement Analysis. From Stakeholders’ Needs to Formal Designs*, 2007, pp. 103–124.
- [45] T. C. de Sousa, J. R. A. Jr., S. Viana, and J. Pavón, “Automatic analysis of requirements consistency with the B method,” *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 2, pp. 1–4, 2010.
- [46] A. Sardinha, R. Chitchyan, N. Weston, P. Greenwood, and A. Rashid, “Ea-analyzer: automating conflict detection in a large set of textual aspect-oriented requirements,” *Autom. Softw. Eng.*, vol. 20, no. 1, pp. 111–135, 2013.