Empirical Software Engineering
https://doi.org/10.1007/s10664-019-09793-8

A systemic framework for crowdsourced test report quality assessment

Xin Chen[1] · He Jiang[2] · Xiaochen Li[2] · Liming Nie[3] · Dongjin Yu[1] · Tieke He[4] · Zhenyu Chen[4]

Published online: 27 February 2020

Abstract
In crowdsourced mobile application testing, crowd workers perform test tasks for developers and submit test reports to report the observed abnormal behaviors. These test reports usually provide important information to improve the quality of software. However, due to the poor expertise of workers and the inconvenience of editing on mobile devices, some test reports usually lack necessary information for understanding and reproducing the revealed bugs. Sometimes developers have to spend a significant part of available resources to handle the low-quality test reports, thus severely reducing the inspection efficiency. In this paper, to help developers determine whether a test report should be selected for inspection within limited resources, we issue a new problem of test report quality assessment. Aiming to model the quality of test reports, we propose a new framework named TERQAF. First, we systematically summarize some desirable properties to characterize expected test reports and define a set of measurable indicators to quantify these properties. Then, we determine the numerical values of indicators according to the contained contents of test reports. Finally, we train a classifier by using logistic regression to predict the quality of test reports. To validate the effectiveness of TERQAF, we conduct extensive experiments over five crowdsourced test report datasets. Experimental results show that TERQAF can achieve 85.18% in terms of Macro-average Precision (MacroP), 75.87% in terms of Macro-average Recall (MacroR), and 80.01% in terms of Macro-average F-measure (MacroF) on average in test report quality assessment. Meanwhile, the empirical results also demonstrate that test report quality assessment can help developers handle test reports more efficiently.

Keywords Crowdsourced testing · Test report quality · Desirable properties · Quality indicators · Natural language processing

Communicated by: Massimiliano Di Penta and David D. Shepherd

This article belongs to the Topical Collection: *Software Analysis, Evolution and Reengineering (SANER)*

✉ Xin Chen
chenxin4391@hdu.edu.cn

Extended author information available on the last page of the article.

1 Introduction

Along with the rapid growth of mobile devices, mobile applications have become more and more powerful and complex in modern society. Although users expect mobile applications to be reliable and secure, the continuously increasing complexity makes them error prone (Liu et al. 2010). To ensure the best potential quality of mobile applications, software testing becomes more and more important. However, due to the specific characteristics of mobile devices, such as unreliable networks, widely varied screen sizes, and diverse operation systems, mobile application testing is complex and challenging. Recently, many companies or organizations tend to adopt crowdsourcing to perform testing for mobile applications by recruiting a large group of potentially undefined, geographically dispersed online individuals (namely crowd workers) (Howe 2006; Mao et al. 2015). Therefore, crowdsourced testing has acquired high popularity in the area of software engineering (Feng et al. 2015; Jiang et al. 2018; Dolstra et al. 2013; Nebeling et al. 2012). Compared against traditional testing (e.g., laboratory testing), crowdsourced testing involves diverse platforms, languages, and users. Developers can gain real feedback information, function requirements, and user experiences (Jiang et al. 2018). Meanwhile, crowdsourced testing recruits a large number of workers for testing, thus achieving high parallelization and significantly improving the test efficiency. Moreover, crowdsourced testing can provide a wide variety of test environments, including mobile devices, network environments, and operating systems, which effectively ensures high software and hardware coverage and greatly reduces the testing cost.

In crowdsourced testing, workers perform test tasks and write test reports for observed phenomena to help developers reveal existing bugs in the software. A typical test report is composed of four fields, including *environment*, *input*, *description*, and *screenshot*. Overall, test reports are similar to bug reports in the content. However, compared with bug reports, test reports have their own characteristics. First, test reports involve not only bugs, but also user experience and function requirements from end users. Second, in this study, test reports are generated in crowdsourced testing for mobile applications, thus they must contain specific features of mobile applications, such as environment information and more screenshots. Third, test reports are submitted by crowd workers who may have extremely diverse experience, thus the quality of test reports varies greatly. These test reports are simultaneously submitted to developers in a short time and developers need to manually inspect them and debug the potential bugs. However, developers encounter two kinds of challenges in inspecting these test reports. On the one hand, the quantity of test reports is large, tremendously exceeding the available resources of developers to inspect them. On the other hand, due to the poor expertise of workers and the inconvenience of editing on mobile devices, the quality of test reports may vary sharply (Jiang et al. 2018). Low-quality test reports will greatly decrease the inspection efficiency of developers.

Many studies concentrate on decreasing the inspection time by reducing the quantity of test reports to inspect. To help developers detect more bugs, some researchers attempt to resolve the problem of test report prioritization based on the concept of "the earlier a bug is detected, the cheaper it is to remedy" (Feng et al. 2015). They leverage either the textual information (Feng et al. 2015) or the combination of both the textual information and the image information (Feng et al. 2016) to prioritize test reports. Given that false positive test reports (which actually report correct behaviors or behaviors that are triggered by the third-party software (Wang et al. 2016)) take developers unnecessary inspection time, some studies are conducted to discriminate them from raw test reports by leveraging text-based classification techniques (Wang et al. 2016; 2017) or machine learning techniques (Wang et al. 2016). Similarly, some researchers attempt to partition test reports into clusters by the text-based hierarchical clustering method (Jiang et al. 2018). Thus, developers only need to inspect a representative test report from each cluster. In such a way, the inspection cost can be significantly reduced.

Although the aforementioned studies have achieved promising results in reducing the overall inspection cost of test reports, they do not consider the impact of the quality of a single test report on the inspection efficiency. High-quality test reports contain relatively complete information that can help developers better understand and fix bugs. Developers can accomplish the inspection within a small amount of time. In contrast, low-quality test reports generally lack some necessary details, such as test steps. At times developers have to search and peruse relevant test reports to assist the fixing procedure, thus the efficiency is seriously decreased. Ideally, if the quality of test reports are reliably assessed by automatic methods, developers can select high-quality test reports for inspection, which will effectively improve the inspection efficiency. Although to the best of our knowledge, no study investigates how to model the quality of test reports, some studies about bug reports (generated from open source projects) and requirement specifications have proposed practicable methods for quality prediction by defining a series of quantifiable indicators to measure the desired features or properties (Zimmermann et al. 2010; Génova et al. 2013; Carlson and Laplante 2014; Wilson et al. 1996).

In this paper, to help developers determine whether a test report should be selected for inspection within limited available resources, we attempt to investigate the problem of crowdsourced test report quality assessment by classifying test reports as either “Good” or “Bad”. To effectively tackle this problem, we propose a new TEst Report Quality Assessment Framework (TERQAF) to automatically evaluate the quality of test reports. First, we systematically summarize some desirable properties to specify what an expected test report should meet and define a taxonomy of quality indicators (which can be divided into four categories) to measure these properties. Then, the numerical values of quality indicators are determined according to the contents of test reports. Finally, we apply logistic regression to train a classifier which takes in test reports as the input and outputs the class label (namely Good or Bad) of test reports. In such a way, based on the predicted results, developers can select high-quality test reports for inspection.

To evaluate the effectiveness of TERQAF, we collect five datasets with in total 936 test reports by performing crowdsourced testing for five mobile applications with our industrial partners. With the help of developers, we manually annotate the quality of test reports, thus forming the ground truth for experiments. We select the Random method and CUEZILLA (a tool for bug report quality assessment) as baselines for comparison. We investigate five research questions and employ widely used Macro-average Precision (MacroP), Macro-average Recall (MacroR), and Macro-average F-measure (MacroF) to evaluate the effectiveness of TERQAF. Experimental results show that TERQAF can achieve 85.18% in terms of MacroP, 75.87% in terms of MacroR, and 80.01% in terms of MacroF on average and outperform the comparative algorithms by 35.24%, 26.00%, and 30.11%, respectively. Meanwhile, experimental results also demonstrate that the four categories of indicators play positive roles in predicting the quality of test reports. In addition, we also perform an empirical experiment to validate the effectiveness of test report quality assessment.

In this study, we make the following contributions:

1. To the best of our knowledge, this is the first work which aims to investigate the quality of test reports and attempts to resolve the problem of test report quality assessment.

2. We propose a new framework named TERQAF consisting of three components to automatically evaluate the quality of test reports. TERQAF defines a series of desirable properties and quantifiable indicators for test reports.
3. To evaluate the effectiveness of TERQAF, we conduct extensive experiments over five crowdsourced test report datasets. Experimental results show that TERQAF can effectively predict the quality of test reports and outperform comparative algorithms.

This paper is an extension of our previous work published in the Developer's Collaboration Track at the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'18) (Chen et al. 2018). In this extension, we define new desirable properties, improve the classifier by adopting logistic regression, introduce new evaluation metrics, and add numerous analytical and empirical results.

The remainder of this paper is organized as follows: In Section 2, we introduce the background of crowdsourced testing and outline the motivation. Some desirable properties are summarized to characterize an expected test report in Section 3. Section 4 defines a series of indicators to measure the desirable properties and shows the corresponding relationship between the desirable properties and the measurable indicators. In Section 5, we elaborately describe the details of TERQAF for test report quality assessment. We present the experimental setup and the experimental results in Section 6 and Section 7, respectively. The threats to validity are discussed in Section 8 and the related work is reviewed in Section 9. Finally, Section 10 concludes this paper.

2 Background and Motivation

In this section, we detail the background of crowdsourced testing and introduce several test report examples to explain the motivation of resolving test report quality assessment.

In crowdsourced testing, developers from companies prepare test packages (software under test and test tasks) for crowdsourced testing and release them online. Workers from crowdsourced platforms select test tasks based on their test devices, perform testing, and write test reports in a predefined format for abnormal behaviors (Jiang et al. 2018). Figure 1 presents the generic procedure of crowdsourced testing. In general, test reports mainly involve natural language information, but sometimes also contain some necessary screenshots. For projects from different crowdsourced platforms, the contents of test reports are basically similar (Feng et al. 2016; Wang et al. 2016). In our experiments, we perform crowdsourced testing for five mobile applications with our industrial partners by the Kikbug crowdsourced testing platform[1] in which test reports are defined by four fields, including *environment*, *input*, *description*, and *screenshot*.

Table 1 shows four examples from the real industrial data. A notable point is that test reports are written in Chinese in our experiments. To facilitate understanding, we translate Chinese into English only for the four test reports. The second column is the environment information, including the software and hardware configuration of test devices, such as phone type, operating system, screen resolution, and system language. The following one is the input information which is actually a series of test steps designed by workers based on the test requirements. If the test steps are correct and detailed, developers can follow them to probably reproduce the revealed bug. The fourth column is the description information. When detecting a bug, workers will use some descriptive terms to describe the specific

[1] http://kikbug.net

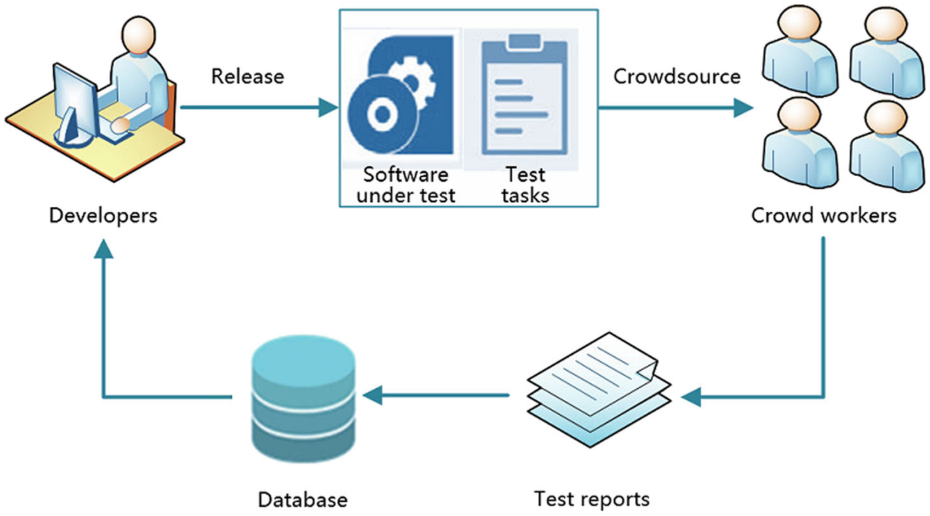


Fig. 1 The generic procedure of crowdsourced testing

phenomenon. Occasionally, workers may report their user experience with the system. The final column is the screenshot information which may provide some necessary images to capture the system states when occurring a bug.

Due to the strict time limitation, developers have to recruit a large number of workers to perform crowdsourced testing and promise some financial compensation for each test task. However, workers are often inexperienced and unfamiliar with software testing. They may perform test tasks for economic income and submit test reports without caring about the quality (Wang et al. 2016). Meanwhile, it is very inconvenient to edit long descriptive natural language on the mobile device soft keyboards. Therefore, test reports are widely different in terms of the quality. For example, as shown in Table 1, TR_1 only contains several words in the *description*, which does not provide enough information for developers to understand what prompt function is missing. In addition, the brief test steps may make it hard for developers to reproduce the reported bug. Similarly, TR_2 simply describes the sharing problem and lacks concrete steps for reproduction. In contrast, TR_3 is a high-quality test report which provides sufficiently detailed test steps for reproducing the bug. It also clearly states that the unexpected behavior causes the sharing problem. TR_4 details two different bugs in the *description*. It involves more details that the sharing operation fails over all interfaces except QQ, but does not distinguish accurately the specific test steps for each bug.

Since test reports are one of the most important resources to reveal potential defects of mobile applications, developers usually understand and fix these defects to ensure the released software as secure as possible. However, due to the impact of the widely varied quality, test report inspection is a tedious and time consuming task. Specially, low-quality test reports may take developers more resources and efforts, thus developers cope very slowly with some test reports or perhaps not at all due to the constraint of limited available resources. In practice, in crowdsourced testing, crowd workers usually submit duplicate test reports to reveal the same bug. Therefore, developers should select a high-quality test report from the duplicates for inspection. In this paper, to help developers predict whether a test report can be selected for inspection, we focus on resolving the problem of test report quality assessment. Based on existing studies about quality prediction for bug reports and

Table 1 Examples of crowdsourced test reports

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: Samsung galaxy s7 Operating System: Android 6.0 Screen Resolution: 5.5 inch System Language: Chinese	1. Open the application. 2. Click on the search button at the bottom. 3. Input an interested topic. 4. Select the first category. 5. Select one picture under this category. 6. Click on the share button. 7. Share the selected picture by WeChat and microblog.	When sharing pictures to circle of friends by WeChat or microblog, an error occured and the system unexpectedly exited.	
TR_4	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. The system recommends what applications to open the downloaded pictures. 2. When sharing pictures by the WeChat, microblog, and QQ interfaces, only the QQ interface succeeds.	

requirement specifications, we define a set of desirable properties and measurable indicators to model the quality of test reports.

In practice, many studies have been conducted for quality assessment of textual documents. In software engineering, requirement engineering is one of the most important stages which aims to acquire the real expected demands of customers and determine what functions should be realized to meet the demands. The requirement engineering process is devoted to iteratively generating and refining the software requirement specifications which play a key role in guaranteeing the quality of projects in software development (Carlson and Laplante 2014). However, manually performed quality assessment for requirement specifications is always burdensome and time consuming (Parra et al. 2015; Thakurta 2013). Therefore, many studies are conducted to explore efficient techniques and tools to help developers automatically model the quality of requirement specifications. In the literature, the most common solution is to define a set of quantifiable indicators based on the textual contents to measure the desirable properties of requirement specifications (Génova et al. 2013; Carlson and Laplante 2014; Rosenberg and Hammer 1999). For example, the indicator *size* (i.e., text length) directly evaluates the conciseness and indirectly measures the understandability and completeness of requirement specifications (Génova et al. 2013).

3 Desirable Properties of Test Reports

As is well known, quality is an ambiguous concept that is associated with different assessment criteria (Génova et al. 2013). In most cases, quality assessment mainly depends on manual judgement. For a given test report, different developers may evaluate it differently with respect to quality due to their respective perspectives. Therefore, based on manual evaluation, we need to uniformly define some desirable criteria or properties which can effectively reflect what test reports we regard as high-quality (namely good) or low-quality (namely bad). In this section, we systematically summarize some desirable properties of test reports and give their explicit definitions.

Although several studies have been conducted for crowdsourced testing, they generally concern how to help developers reduce the inspection cost of test reports. This study is the first work aiming at investigating the quality of test reports. Actually, test reports are similar to bug reports. Compared with test reports, bug reports contain not only bug description and test steps, but also sometimes code examples. However, many studies about bug reports primarily focus on the textual contents since they are the most widely used and the most easily comprehensible (Zimmermann et al. 2010). By an in-depth investigation on existing studies about bug reports, we summarize some desirable properties including atomicity, correctness, completeness, conciseness, understandability, unambiguity, consistency and reproducibility (Rastkar et al. 2014; Zimmermann et al. 2010; Bettenburg et al. 2008; Hooimeijer and Weimer 2007; Joorabchi et al. 2014), which are also adapted to test reports. We present the detailed definition of these desirable properties as follows:

- Atomicity: each test report reveals one bug without mixing information about other bugs.
- Correctness: the revealed bug reflects a real defect existing in the system and each field provides the prescribed information.
- Completeness: each test report contains all fields and all information should be complete.

- Conciseness: each sentence in a test report conveys different information and no sentence conveys irrelevant information about the bug.
- Understandability: the contained content can be clearly understood without difficulty and the described bug can be correctly identified according to the contents.
- Unambiguity: there is only one interpretation for each word and sentence in the test report.
- Consistency: there are no contradictions among all information fields in each test report.
- Reproducibility: the provided test steps are detailed and effective for reproducing the revealed bug.

4 Taxonomy of Indicators

In the previous section, we have clarified what we should measure for a test report. Although these desirable properties mainly rely on individually subjective judgments, it does not mean that they are arbitrary and easy to measure. Therefore, we need to define a series of measurable indicators based on intrinsically objective characteristics of the textual contents to evaluate the desirable properties. In practice, these quantitative indicators are related to the qualitative properties to some extent. They can directly or indirectly measure the quality of test reports (Génova et al. 2013). In literature, researchers have summarized an overall taxonomy of indicators to measure the desirable properties of requirement specifications. Differing from test reports, requirement specifications usually contain more contents and involve more desirable properties, such as modifiability, abstraction, and verification (Génova et al. 2013). Therefore, some indicators may be not intrinsically appropriate to evaluate the quality of test reports. Based on this fact, we define the quality indicators by combining the existing studies (Génova et al. 2013; Carlson and Laplante 2014) and the specific characteristics of test reports. They can be divided into four categories:

- Morphological indicators, e.g., size and readability, which mainly reflect the surface characteristics of the textual contents of a test report without considering the conveyed semantic information.
- Lexical indicators, e.g., the number of imprecise terms or anaphoric terms, which attempt to lexically measure the textual content of a test report according to the term lists defined by users.
- Analytical indicators, e.g., the usage of domain terms, which focus on semantically analyzing the textual contents of test reports by leveraging some domain terms provided by existing studies.
- Relational indicators, e.g., itemization, which try to evaluate the structured properties of test reports or reveal whether each test report contains all the field information.

In this section, we elaborately describe each quality indicator and definitely specify what desirable properties it evaluates. Since test reports are composed of four fields, i.e., *environment*, *input*, *description*, and *screenshot*, we need to evaluate the four fields.

4.1 Morphological Indicators

In quality measurement, size and readability index are the most widely used and the most easily acquired morphological indicators. In addition, we also take the number of punctuation signs as an indicator to measure the quality of test reports.

4.1.1 Size

In general, the size of a document can be calculated by the number of characters, words, phrases, sentences, text lines, and paragraphs (Génova et al. 2013; Carlson and Laplante 2014). It directly evaluates the atomicity and conciseness of test reports and is indirectly associated with all other desirable properties. Due to the inconvenience of editing on mobile devices, test reports are usually short, which only contain one or several sentences. In addition, considering that the number of words or phrases is not suitable for measuring the size of test reports written in Chinese, we only adopt the number of characters within a test report as the metric of size. Intuitively, long test reports may take developers much reading time, while short test reports may miss some important details about the bugs. As a result, a test report should keep the size neither too long nor too short.

4.1.2 Readability

Readability is another representative quality indicator which attempts to evaluate the difficulty level of reading a document (Génova et al. 2013; Carlson and Laplante 2014). In literature, related studies have been extensively conducted to investigate the readability of documents written in Chinese. Diverse formulas have been proposed to compute the readability scores. The first acknowledged formula for Chinese documents readability measurement is $Y = 14.9596 + 39.07746X_1 + 1.011506X_2 - 2.48X_3$ (Yang 1970), where X_1 represents the proportion of difficult words (excluded from 5,600 commonly used words), X_2 is the number of sentences in a document, and X_3 denotes the average stroke count of characters within a sentence. Certainly, there exist other readability measurement formulas, such as $Y = -11.946 + 0.123X_1 + 0.198X_2 + 0.811X_3$ (Guo 2010), where X_1 is the average length of a sentence, X_2 and X_3 denote the numbers of difficult words and difficult characters (words or characters excluded by the common dictionary), respectively. In this study, we adopt the former to calculate the readability scores of test reports.

This indicator is also appropriate to evaluate the readability of documents written in other languages. For example, we can use the Flesch readability index (Flesch 1948), one of the most widely used measurement, to model the readability of English test reports. Its computation formula is $R_{Flesch} = 206.835 - (1.015 \times WPS) - (84.6 \times SPW)$, where WPS and SWP are the average numbers of words within a sentence and syllables of a word, respectively. In general, the larger the value is, the higher the understandability of the text is. In addition, Josè replaces the coefficient 84.6 with 60 and increases WPS by 0.4 times to make the Flesch readability formula effective to calculate the readability of Spanish documents (Férnandez HJ 1959).

4.1.3 Punctuation

Another important indicator is punctuation marks (Génova et al. 2013) which are related to the understandability of test reports. Punctuation marks are used to break down a long text into a series of relatively short sentences. The absence of necessary punctuation marks usually causes difficulty to understand a long text. In contrast, excessively using punctuation may result in the loss of semantics. Although the number of punctuation marks can be easily obtained, it is hard to determine the actual acceptable number for a given text. In practice, users have the expected average sentence length which can be empirically determined. When the size of a test report is determined, the number of punctuation marks can be correspondingly calculated according to the expected average sentence length. In aggregate, the average sentence length is a more easily measurable indicator. Notably, in this study, the used punctuation marks include not only Chinese but also English punctuation marks.

4.2 Lexical Indicators

Compared to morphological indicators, lexical indicators need some reference information, namely term lists defined by users, to measure the quality of test reports. In the existing studies about requirement specification quality assessment, researchers have summarized overall term lists for English documents (Génova et al. 2013; Carlson and Laplante 2014). In this study, we reuse these term lists and translate them into Chinese. Nevertheless, we add some synonyms by leveraging the thesaurus[2] to form the ultimately used term lists for the processing of Chinese documents. In practice, these terms are commonly used in documents, we can easily collect them according to the Chinese dictionaries even though no referenced term list is available.

4.2.1 Imprecise Terms

When detecting a bug, workers need to write a test report to describe the concrete phenomenon with descriptive terms. However, due to the uncertainty and the shortage of vocabulary, they tend to use some imprecise terms which contain ambiguous information. At times, this will introduce ambiguities for understanding and fixing the bug. Therefore, a high-quality test report should not contain imprecise or subjective terms. In literature (Génova et al. 2013), the imprecise terms are collected and divided into six different groups according to their characteristics.

- 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: can, may, possibly, perhaps, optionally, etc.
- Usability: experienced, familiar, adaptable, easy, etc.

4.2.2 Anaphoric Terms

Anaphoric terms are pronoun or other linguistic units to refer back to other terms in former sentences, which are useful to avoid the recurrence. Typically, anaphoric terms contain personal pronouns (such as “it”, “them”), relative pronouns (such as “that”, “which”, “where”), and demonstrative pronouns (such as “this”, “that”, “these”, “those”) (Génova et al. 2013). In practice, using anaphoric terms is grammatically correct and sometimes can effectively improve the conciseness of contents. However, they may introduce threats to the quality of test reports with respect to the understandability due to their inherent imprecisions and ambiguities. Therefore, test reports should exclude all anaphoric terms.

4.2.3 Directive Terms

In contrast to imprecise terms and anaphoric terms, directive terms play a positive role in predicting the quality of test reports (Carlson and Laplante 2014). They are usually used to

[2]https://www.ltp-cloud.com/download/

make test reports more understandable. Generally, the following words or sentences behind directive terms possibly complement some additional details. For example, they may give a specific example to further describe or explain the concrete behaviors of the bug. Therefore, a high proportion of the usage of directive terms may be good to improve the understandability of test reports. In test reports, the widely used directive terms involve "e.g.", "i.e.", "for example", "for instance", "note", etc.

4.3 Analytical Indicators

As mentioned in the previous section, the target of analytical indicators is to semantically analyze the textual contents of test reports. In the studies related to requirement specification quality assessment, researchers have shown that verbal tense, mood terms, and domain terms are the most important analytical indicators for English documents (Génova et al. 2013). However, different from English, Chinese does not involve verbal tense. Meanwhile, our investigation on real industrial data indicates that test reports do not contain or contain few mood terms. Therefore, we only adopt domain terms to measure the quality of test reports in this study.

Typically, domain terms refer to the words or phrases that are organized either as a simple glossary of terms or in a complicated structural form such as thesauri or ontologies (Génova et al. 2013). However, in different areas, domain terms may vary sharply. Consequently, defining and collecting relevant domain terms for a specific area is an important but burdensome task. Fortunately, Ko et al. (2006) have systematically summarized the domain terms for bug reports. In a follow-up study, Zimmermann et al. have used these terms to evaluate the quality of bug reports (Zimmermann et al. 2010). Taking the same nature of test reports and bug reports into consideration, these domain terms also adapt to test reports. In this study, we reuse the existing term lists and introduce negative terms to form the eventual domain terms. Notably, we partition them into four categories, all of which play positive roles in measuring the quality of test reports.

4.3.1 Negative Terms

A software bug actually reveals an incorrect or missing function of an application. In crowdsourced test reports, workers tend to use negative terms to represent the absence of system functions, such as "no", "not", "miss", "fail", and "lack". When identifying a negative term, it potentially indicates that a bug is revealed in the test report.

4.3.2 Behavior Terms

Based on the definition from Wikipedia,[3] a software bug refers to a flaw, defect, failure, or fault that makes the computer program or system occur incorrect or unexpected results. In test reports, worker usually describe the incorrect result by using some behavior terms, such as "error", "bug", "defect", "problem", "failure", "flaw", and "fault".

4.3.3 Action Terms

When performing test task, workers click on some user interfaces (e.g., button, dialog) to detect potential bugs. This procedure will produce a series of test steps which are actually

[3] https://en.wikipedia.org/wiki/Software_bug

regarded as test cases in crowdsourced testing. Therefore, test steps must involve some action terms, such as “open”, “select”, “click”, “enter”, and “check”.

4.3.4 Interface Elements

To be exact, test steps are actually a sequence of events which are triggered by clicking on the user interface elements. If the *input* of a test report is associated with test steps, it may include some action terms followed by corresponding user interface elements, such as “button”, “toolbar”, “menu”, “dialog”, and “window”.

In summary, negative terms and behavior terms are used to evaluate the *descriptions* of test reports while action terms and interface elements are adopted to measure the *inputs*.

4.4 Relational Indicators

In practice, test reports are structured documents consisting of several fields of which each provides some critical information to help developers understand and fix the revealed bugs. Ideally, a good test report involves all field information and the content in each field should be correct and related to the same bug. For instance, Field *description* should present the detailed bug information and Field *input* should list the concrete test steps. However, as same as the empirical investigation on bug reports (Zimmermann et al. 2010), due to the difficulty of providing key information and the poor performance of workers, test reports usually lack important details or contain incorrect information in a certain field. Therefore, we define some relational indicators to measure the correctness and the relevance of test reports.

4.4.1 Itemizations

Undoubtedly, test steps are the most important information to reproduce the bug in test reports. In editing test reports, some experienced workers may leverage itemizations or enumerations to orderly organize test steps in the corresponding field to help developers discriminate each step more efficiently. Generally, if the *input* in a test report is presented in a structured form, such as itemizing with itemizations or enumerations, it indicates that the *input* is associated with test steps with a fairly high probability, as shown TR_2 and TR_3 in Table 1. To identify itemizations in test reports, we detect the lines starting with an itemization character, e.g., “$\cdot$”, “$*$”, or “+”. Similarly, we also distinguish enumerations by examining whether each line starts with numbers or serial numbers in which numbers are enclosed by parentheses, brackets or followed by a single punctuation character (Zimmermann et al. 2010).

4.4.2 Environment

As shown in Table 1, Field *environment* lists some basic configuration of mobile devices to help developers acknowledge the adaptation of mobile applications. As a result, a high-quality test report should contain environment information. Compared with other fields, the *environment* field is presented in a fixed form with some specific keywords, including “phone type”, “operating system”, “screen resolution”, and “system language”. By searching these keywords, we can easily acknowledge whether the test report provides the environment information or not.

4.4.3 Screenshots

In test reports, screenshots are also important information for developers to understand the revealed bug by capturing the abnormal phenomenon of the system. Thus, a complete test report should contain some necessary screenshots. In our experiments, screenshots are uniformly packaged in attachments which are simultaneously delivered to the database together with test reports. In general, these attachments contain only screenshots. Even though the attachments contain text, we can use the *file* tool in UNIX to identify screenshots (Zimmermann et al. 2010). In this paper, if an attachment is detected, the test report must carry screenshots.

In aggregate, these quality indicators are defined based on the textual contents and the compositions of test reports. They are more or less associated with one or more desirable properties of test reports. To facilitate understanding, we show the corresponding relationship between measurable indicators and desirable properties in Table 2, where "X" indicates a direct relationship between the measurable indicators and the desirable property while "●" represents an indirect relationship. For example, the indicator *size* directly measures the atomicity and conciseness and is indirectly related to all other properties. In the table, the relationships of the morphological and lexical indicators are defined based on the existing studies (Génova et al. 2013; Zimmermann et al. 2010). The relationships of the analytical and relational indicators are summarized based on our experience.

5 Test Report Quality Assessment Framework

In this section, we describe TERQAF consisting of three components, as shown in Fig. 2. First, we preprocess crowdsourced test reports by word segmentation. Then, we determine the numerical values of measurable indicators according to the contents of test reports. Finally, we adopt logistic regression to train a classifier and leverage the trained classifier to predict the quality of test reports as either "Good" or "Bad".

5.1 Preprocessor

The preprocessor aims to implement word segmentation for test reports. In our experiments, test reports are written in Chinese with extremely few English terms. To evaluate the quality of test reports, we define a series of indicators according to the contents of test reports. Many indicators, such as lexical indicators and analytical indicators, leverage term lists to evaluate test reports. Therefore, we need to perform word segmentation for test reports by using Chinese Natural Language Processing (NLP) tools. In literature, Language Technology Platform (LTP),[4] ICTCLAS,[5] and IKAnalyzer[6] are the widely used NLP tools for processing Chinese documents (Jiang et al. 2018; Feng et al. 2015; Feng et al. 2016). In this study, we adopt IKAnalyzer since it has good effectiveness in word segmentation and can implement simple processing for English (Jiang et al. 2018).

As for term lists, we first collect the initial term lists by referring to the studies conducted by Génova et al. (2013) and Zimmermann et al. (2010), respectively. Then, we translate the

[4]http://www.ltp-cloud.com/

[5]http://ictclas.nlpir.org/

[6]http://www.oschina.net/p/ikanalyzer

Table 2 The corresponding relationship between measurable indicators and desirable properties

Category	Indicator	Desirable property							
		Atomicity	Correctness	Completeness	Conciseness	Understandability	Unambiguity	Consistency	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	●		●

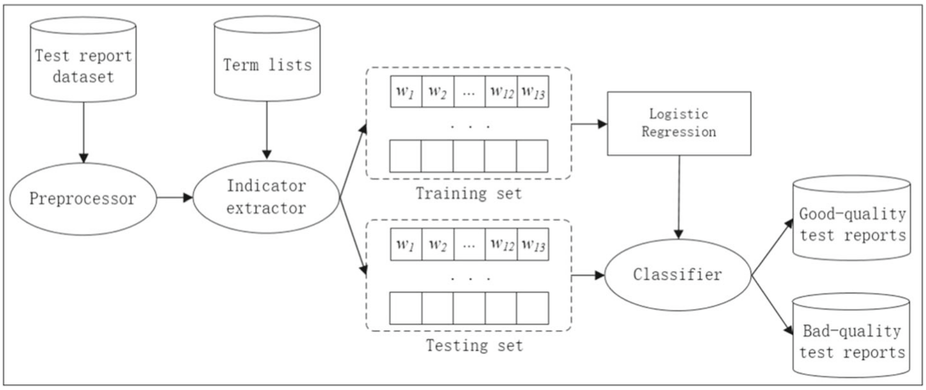


Fig. 2 TERQAF framework

term lists to form Chinese term lists, and leverage the thesaurus[7] to add some relevant terms for each list. Finally, we collect and add new term lists, including the negative term list and the environment term list. Thus, the ultimately used term lists are formed.

5.2 Indicator Extractor

The indicator extractor is designed to build the quality vector for each test report. As mentioned above, we define 13 indicators to measure the quality of test reports. As a result, each test report can be represented by a 13-dimensional quality vector in which each dimension corresponds to a single indicator. Its numerical value is determined based on the contents of the test report. Next, we explain how to determine the numerical value of each dimension in detail.

For the indicator *size*, the numerical value is the number of Chinese characters (excluding punctuation marks) of text contained in the test report. The value of the indicator *readability* can be calculated by the Chinese readability measurement formula. For the indicator *punctuation*, we replace it with the average sentence length that can be determined by the size of text and the number of punctuation marks. In terms of morphological indicators and lexical indicators, the value of each indicator is the number of occurrences of related terms in the text. Notably, if a term occurs repeatedly in the text, we repeatedly calculate the number of occurrences of the term. For example, assuming a term list only contains two terms, one occurs 2 times and the other occurs 3 times. The value of the indicator is 5. Similarly, we calculate the value of the indicator *itemizations* according to the number of occurrences of the same type of itemizations which refers to itemizations with the same structure. For example, "1)" and "2)" are the same type of itemization, but "1" and "1)" as well as "+" and "-" are different types of itemizations. For the indicator *screenshot*, we define the number of screenshots as the value of the corresponding dimension. Different from the above indicators, the value of the indicator *environment* is 0 or 1. We determine the value by identifying whether the environment information contains all the four keywords, including "phone type", "operating system", "screen resolution", and "system language". If yes, the value is 1; otherwise, the value is 0.

[7]https://download.csdn.net/download/u010721054/9494800

5.3 Classifier

To predict the quality of test reports, we need to construct a classifier. Considering that the collected data is small-scale, we adopt logistic regression to train the classifier.

5.3.1 Logistic Regression

In this study, we employ logistic regression to train the classifier. Logistic regression is a generally used machine learning algorithm (Hsu et al. 2019). It is usually used to analyze the association between a categorical dependent variable and a set of independent variables. It requires that the dependent variable has only two values, such as “0” and “1” or “Yes” and “No”. Since logistic regression does not assume that the independent variables meet the normal distribution, it is widely applied to resolve various binary classification problems (Denoeux 2018).

In logistic regression, each sample is represented as $[x_1, x_2, \cdots, x_k; y]$, where x_j $(j = 1, 2, \cdots, k)$ represents a feature value, and y is the label of the sample. The classification model of logistic regression is defined:

$$h_\theta(\boldsymbol{x}) = \frac{1}{1 + e^{\theta^T \boldsymbol{x}}} \tag{1}$$

where $\theta = \{\theta_1, \theta_2, \cdots, \theta_k\}$ represents the set of regression coefficients, $h_\theta(\boldsymbol{x})$ is the prediction result. The logistic regression model leverages the maximum likelihood estimate to calculate the regression coefficients:

$$P(y|\boldsymbol{x}) = (h_\theta(\boldsymbol{x}))^y (1 - h_\theta(\boldsymbol{x}))^{(1-y)},\ y = 1 \text{ or } 0 \tag{2}$$

The maximum likelihood estimate is as follows:

$$L(\theta) = \prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{x}^{(i)}) = \prod_{i=1}^{n} (h_\theta(\boldsymbol{x}^{(i)}))^{y^{(i)}} (1 - h_\theta(\boldsymbol{x}^{(i)}))^{1-y^{(i)}} \tag{3}$$

where i is the index of a sample, and n represents the number of samples.

5.3.2 Training Classifier

In this study, each test report can be regarded as a sample. By the defined indicators, we represent each test report as a vector $[x_1, x_2, \cdots, x_{13}, y]$, where $x_j (j = 1, 2, \cdots, 13)$ represents the numerical value of an indicator, y denotes the class label of a sample and its value is 1 (Good) or 0 (Bad).

Classifier training follows a supervised learning paradigm. Based on the objective of the task, we need sufficient samples including good quality test reports and bad quality test reports. In this study, we collect five datasets. The detailed information is presented in Section 6. We employ leave-one-out cross validation (Petrosyan et al. 2015) to train the classifier. The main reason for employing leave-one-out cross validation is that the results are more reliable and reproducible (Jiang et al. 2016). Meanwhile, leave-one-out cross validation is suitable for training the classifier with small-scale data. More specifically, in each run, one dataset is chosen as the testing set and all the other datasets are treated as the training set. Then, all the test reports from the training set are fed into the classifier to perform training. The training objective is to minimize the logarithm function of the maximum likelihood estimate:

$$J(\theta) = log(L(\theta)) = \prod_{i=1}^{n}(y^{(i)}log(h_\theta(\boldsymbol{x}^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(\boldsymbol{x}^{(i)}))) \tag{4}$$

6 Experimental Setup

In this section, we detail the experiment setup. First, we present our Research Questions (RQs) and explain why setting these RQs. Then, the data acquisition and data validation are described. Third, the evaluation metrics are introduced.

6.1 Research Questions

In this study, we propose a new problem of test report quality assessment and present our attempts towards resolving the problem. We define a series of measurable indicators and develop a new framework named TERQAF consisting of three components to predict the quality of test reports. The experimental goal includes two aspects: validating the effectiveness of TERQAF and evaluating the role of the defined indicators. Therefore, we design the following four research questions (RQs).

RQ1. Can TERQAF outperform some baseline methods in measuring the quality of test reports?
RQ2. Can the classifier trained by logistic regression outperform the classifiers built by other methods?
RQ3. How do the four categories of indicators impact the performance of TERQAF?
RQ4. What is the importance degree of each indicator?

Both *RQ1* and *RQ2* are designed to evaluate the effectiveness of TERQAF. *RQ1* aims to evaluate how effective TERQAF is in predicting the quality of test reports. RQ2 attempts to validate the efficacy of prediction method in the critical component (namely the classifier), i.e., whether logistic regression has better performance than other comparative prediction algorithms. Both *RQ3* and *RQ4* are designed to explore the role of the defined indicators. *RQ3* is to validate the impacts of the four categories of indicators in predicting the quality of test reports by quantitative analysis. *RQ4* tries to investigate the degree of importance of each indicator by qualitative analysis.

6.2 Data Acquisition and Validation

In this subsection, we detail how to collect the data and how to form the ground truth for experiments.

6.2.1 Data Acquisition

From October 2015 to January 2016, we conduct crowdsourced testing for five mobile applications with our industrial partners, including UBook, Justforfun, CloudMusic, SE-1800, and iShopping. We briefly describe the five mobile applications as follows:

- *UBook:* an online education application developed by New Orientation.[8]
- *JustForFun:* a photo sharing application developed by Dynamic Digit.
- *CloudMusic:* a music playing and sharing application developed by NetEase[9].
- *SE-1800:* an electrical monitoring application developed by Panneng.
- *iShopping:* an online shopping guideline App developed by Alibaba[10].

We recruit 352 workers which are composed of students from different universities through the Kikbug crowdsourced platform to perform crowdsourced testing for the five applications. These students all have experience in using the Kikbug platform. They have once completed at least one historical crowdsourced test task through the platform and submitted valid test reports validated by developers. In our experiments, only 321 students submit test reports for the five applications.

For each application, we prescribe the test time to be 2 weeks. Workers need to install a small application on their Android devices, which is used to write and deliver test reports. Workers design appropriate test steps and perform test tasks according to the test requirements defined by developers. Once a bug is detected, workers are required to fill in some information for forming a test report in the small application. They need to use some descriptive text information to describe the observed abnormal behavior and record the detailed test steps for reproducing the bug. Sometimes, workers can add some screenshots to display the concrete phenomenon when the bug occurs. Meanwhile, the application can automatically capture the configuration of the mobile device. Eventually, a complete test report is generated and delivered to the platform through the small application.

6.2.2 Data Validation

When all the test reports are delivered to the platform, developers need to manually evaluate these test reports and give some economic compensation based on their quality. In our experiments, 15 developers take charge of the evaluation of test reports for the five applications. Each application involves 3 developers. According to the demands from companies, the evaluation concerns not only the revealed bugs, but also the writing quality of test reports since the developers think the writing quality greatly influences the inspection efficiency. Therefore, to ensure the fairness and effectiveness of the evaluation, the developers design a series of criteria which are presented in Table 3. First, each test report is scored by three developers on a scale of 100 based on the criteria. Then, the average score is calculated. If the average score of a test report exceeds 60 (which is an empirical value determined by those developers), the developers believe that the test report is well-written and the worker should be given some economic compensation.

In this study, we also select 60 as the threshold value to label test reports to form the ground truth. That is, if the average score of a test report is greater than 60, it is marked as

[8]http://www.pgyer.com/y44v
[9]http://music.163.com
[10]https://guang.taobao.com

Table 3 The scoring criteria for evaluating test reports

No.	Item	Score
1	Whether a test report reveals a true bug, or whether it is related to user experience or function requirements.	If yes, +10 points (bug), +5 points (user experience or function requirements); otherwise, +0 points. If a test report does not reveal a true bug or is not related to user experience or function requirements, the developers do not need to continue the evaluation.
2	Whether the revealed bug is related to the corresponding test requirement.	If yes, +5 points; otherwise, +0 points.
3	Test description: (1) the level of detail; (2) the level of expertise; (3) readability.	+1-10 points; +1-10 points; +1-5 points;
4	Test input: (1) whether the input is related to test steps; (2) the level of detail; (3) the level of expertise; (4) readability.	If yes, +10 points; otherwise, +0 points; +1-10 points; +1-10 points; +1-5 points.
5	Test output: the number of screenshots	=0, +0 points; =1, +5 points; >=2, +10 points.
6	Test environment: whether a test report contains environment information.	If yes, +10 points; otherwise, +0 points.
7	Other information: whether a test report contains other information such as video, audio.	If yes, +5 points; otherwise, +0 points.

Note: The level of expertise evaluates whether test reports contain some technical terms or domain terms

"Good"; otherwise, it is marked as "Bad". There are two main reasons. First, 60 may be a good threshold value to distinguish the quality of test reports based on both the experience of developers and the actual demand of companies. Second, the number of low-quality test reports outperforms that of high-quality test reports by an observation on the five datasets. Based on the evaluated results of test reports, we find that the average scores of many test reports are 0. These test reports are invalid test reports which actually describe correct behaviors or behaviors that are incurred by third-party software applications (Feng et al. 2015). However, invalid test reports may contain some information (such as test steps, environment, screenshot) but lack bug description information, which may influence the effectiveness of TERQAF. Fortunately, some researchers have conducted extensive studies to identify invalid test reports and developed state-of-the-art techniques with up to 99% in terms of identification accuracy (Wang et al. 2016). Therefore, in this study, we remove these invalid test reports from the five datasets straightforwardly.

We collect five datasets including 443, 291, 348, 408, and 238 test reports. 792 invalid test reports are removed and thus the numbers of the retained test reports in the five datasets are 201, 230, 201, 215, and 89, respectively. The detailed statistical information of the used datasets is presented in Table 4. 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. As shown in

Table 4 Five crowdsourced test report datasets

Dataset	Version	$\#R$	$\#B$	R_m	$\#R_g$	$\#R_b$
UBook	2.1.0	201	30	53	89	122
JustForFun	1.8.5	230	25	55	92	138
SE-1800	2.5.1	201	32	35	43	158
iShopping	2.5.1	215	65	28	35	180
CloudMusic	1.3.0	89	21	8	47	42
Totals		936	173	179	306	630

the table, the five datasets include 89, 92, 43, 35, and 47 good quality test reports, and 122, 138, 158, 180, and 42 bad quality test reports, respectively.

6.3 Evaluation Metrics

In practice, test report quality assessment can be regarded as a binary classification problem. Precision, recall, and F-measure are the most frequently used metrics for evaluating the performance of automated classification techniques in binary classification problems. Precision measures the correct degree of the predicted results by comparison with the ground truth, recall evaluates the level of consistency between the predicted results and the ground truth, and F-measure is the tradeoff between precision and recall.

However, different from generic binary classification problems, this task distinguishes not only good test reports, but also bad test reports. That is, both good test reports and bad test reports can be viewed as positive samples. Only calculating the precision, recall, and F-measure for one classification (e.g., good test reports) cannot reflect the effectiveness of TERQAF on the other classification. Therefore, we adopt the widely used evaluation metrics for multi-classification problems, namely Macro-average Precision (MacroP), Macro-average Recall (MacroR), and Macro-average F-measure (MacroF) (Aceto et al. 2018), to evaluate the effectiveness of TERQAF. We first calculate the local result of each classification and then achieve the global result by averaging the results of all the classifications. Assuming that $G = \{G_1, G_2\}$ and $P = \{P_1, P_2\}$ represent the predicted results and the ground truth, respectively, and $G_i(i = 1, 2)$ corresponds to $P_j(j = 1, 2)$. The formulas for MacroP, MacroR, and MacroF are presented as follows:

$$MacroP = \frac{1}{c}\sum_{i=1}^{c}\frac{TP_i}{TP_i + FP_i} \tag{5}$$

$$MacroR = \frac{1}{c}\sum_{i=1}^{c}\frac{TP_i}{TP_i + FN_i} \tag{6}$$

$$MacroF = \frac{2 * MacroP * MacroR}{MacroP + MacroR} \tag{7}$$

where c is the number of classifications, TP_i is the number of test reports belonging to both G_i and P_j, FP_i is the number of test reports belonging to both non-G_i and P_j, and FN_i is the number of test reports belonging to both G_i and non-P_j.

7 Experimental Results

In this section, we investigate four research questions to validate the performance of TERQAF.

7.1 Investigation into RQ1

Motivation For this framework (namely TERQAF) to be useful, one of the important questions is how effective the framework is in performing its prediction and whether it can outperform baseline methods. Since to the best of our knowledge this study is the first work to investigate the quality of test reports, no state-of-the-art technique is available to validate the effectiveness of TERQAF. As mentioned in the previous section, test reports are similar to bug reports, techniques for bug report quality assessment may be suitable to test reports. Hence, we select CUEZILLA (Zimmermann et al. 2010) as a baseline for comparison. In addition, we also select one preliminary method, the Random method, as a comparative method to validate the effectiveness of TERQAF.

CUEZILLA is an effective method for predicting the quality of bug reports. It measures the quality of bug reports based on the contained field information. CUEZILLA defines seven desired features to characterize good-quality bug reports and scores each feature with either binary or continuous values to form a feature vector.

Approach In our study, given that test reports contain no code samples, stack traces, and patches, we only leverage itemizations, keyword completeness, readability, and screenshots to measure the quality of test reports. Notably, keyword completeness includes five indicators, i.e., action items, expected and observed behaviors, steps to reproduce, build-related, and user interface elements. Thus, each feature vector consists of eight components. We also adopt leave-one-out cross validation to predict the results. In this experiment, we employ the same classifier (used by TERQAF) to predict the quality of test reports for CUEZILLA. The Random method randomly predicts a test report as either "Good" or "Bad" with respect to the quality.

Results Taking the nature of the Random method, we independently run it 20 times and calculate the average results. Table 5 presents the experimental results of different methods over the five datasets. As shown in the table, TERQAF outperforms CUEZILLA and Random in terms of MacroR and MacroF over all the datasets, and in terms of MacroP over the UBook, iShopping, and CloudMusic datasets. For example, TERQAF achieves 84.54% in terms of MacroP, 77.82% in terms of MacroR, and 81.04% in terms of MacroF, and

Table 5 Experimental results of different methods over all datasets

Dataset	TERQAF			CUEZILLA			Random		
	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF
UBook	84.54 %	77.82 %	81.04 %	72.94 %	68.94 %	70.88 %	49.60%	49.52%	49.55%
JustForFun	78.01 %	82.49 %	80.19 %	79.60 %	78.93 %	79.26 %	50.08%	50.10%	50.09%
SE-1800	87.10 %	75.90 %	81.12 %	87.78 %	74.67 %	80.70 %	50.08%	50.09%	50.08%
iShopping	89.59 %	66.75 %	76.50 %	84.31 %	66.17 %	74.14 %	49.35%	48.99%	49.17%
CloudMusic	86.65 %	76.38 %	81.19 %	84.36 %	74.24 %	78.98 %	50.56%	50.63%	50.60%

improves CUEZILLA by 11.61%, 8.89%, and 10.16%, and Random by 34.94%, 28.30%, and 31.49% over the UBook dataset, respectively. The potential reason is that CUEZILLA only defines several indicators to evaluate the quality of test reports, thus its performance is easily influenced by one or more indicators. For example, if test reports do not (or a few of test reports) contain the terms related to these indicators defined by CUEZILLA, the performance of CUEZILLA tends to be poor. In addition, CUEZILLA does not define an indicator to measure the environment information that is also important. Comparatively, CUEZILLA has high MacroP but relatively low MacroR over all datasets. For example, CUEZILLA achieves 84.31% in terms of MacroP and 66.17% in terms of MacroR. Similarly, TERQAF achieves better results in terms of MacroP and relatively low results in terms of MacroR over all the datasets but JustForFun. Surprisingly, CUEZILLA achieves better results in terms of MacroP than that of TERQAF over the JustForFun and SE-1800 datasets. The reason may be that test reports in these two datasets contain no terms related to some indicators, thus some indicators may be ineffective. For example, we observe that a few test reports in these two datasets contain relevant terms of the lexical indicators. In addition, Random works poorly in test report quality assessment. It achieves approximate results in terms of MacroP, MacroR, and MacroF over the five datasets.

Conclusion TERQAF achieves good results in terms of MacroP, MacroR and MacroF over all datasets. TERQAF significantly outperforms both CUEZILLA and Random in predicting the quality of test reports.

7.2 Investigation into RQ2

Motivation For the task of test report quality assessment, the classifier plays an important role to decide the performance of TERQAF. In this study, we apply logistic regression to train the classifier. Besides, Support Vector Machine (SVM) and Naive Bayes (NB) are also classic efficient machine learning algorithms in precessing small-scale data. In this RQ, we attempt to investigate whether logistic regression can outperform SVM and NB. In addition, we also validate whether logistic regression can improve the unsupervised method proposed in the previous study (Chen et al. 2018).

In the study (Chen et al. 2018), we apply step transformation functions to transform the nominal values of indicators into numerical values and adopt 60% of indicators with good results to determine the quality of test reports.

Approach In this experiment, we fully follow the unsupervised method (Chen et al. 2018) to build the classifier and adopt the same parameters. In order to facilitate representation, we represent the previous proposed method as TERQAF-UN. In addition, we replace logistic regression with SVM and NB to train the classifier in the third component, respectively, and keep other components unchanged. Similarly, we represent the two methods as TERQAF-SVM and TERQAF-NB. We also adopt MacroP, MacroR, and MacroF as the evaluation metrics.

Results Table 6 presents the experimental results of different methods in terms of MacroP, MacroR, and MacroF on the five datasets. As shown in the table, for the average results, we can observe that TERQAF outperforms other methods in terms MacroP, MacroR, and MacroF. In addtion, TERQAF can obtain better results than TERQAF-UN, TERQAF-SVM, and TERQAF-NB in terms of MacroF on the JustForFun, SE-1800, and iShopping datasets. For example, TERQAF achieves 80.19% in terms of MacroP on the Justforfun dataset and

Table 6 Impact of different categories of indicators on experimental results

Dataset	TERQAF			TERQAF-UN			TERQAF-SVM			TERQAF-NB		
	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF
UBook	84.54%	77.82%	81.04%	85.19%	83.41%	84.29%	87.41%	83.48%	85.40%	86.39%	77.56%	81.73%
Justforfun	78.01%	82.49%	80.19%	76.18%	80.77%	78.41%	74.44%	79.86%	77.06%	74.36%	77.67%	75.98%
SE-1800	87.10%	75.90%	81.12%	84.22%	74.38%	78.99%	78.82%	70.12%	74.22%	84.24%	72.36%	77.85%
iShopping	89.59%	66.75%	76.50%	72.73%	70.32%	71.51%	84.79%	67.42%	75.12%	84.90%	63.05%	72.36%
CloudMusic	86.65%	76.38%	81.19%	86.44%	84.31%	85.36%	77.77%	72.81%	75.21%	75.51%	67.64%	71.36%
Average	85.18%	75.87%	80.01%	80.65%	74.74%	77.40%	81.08%	71.66%	75.86%	80.95%	78.64%	79.71%

improves TERQAF-UN, TERQAF-SVM, and TERQAF-NB by 1.78%, 3.13%, and 4.21%, respectively. The corresponding MacroP and MacroR are 78.01% and 82.49%, respectively. Similarly, in terms of MacroP and MacroR, TERQAF outperforms other methods on most of the datasets.

Compared with TERQAF, TERQAF-SVM, and TERQAF-NB, the differences between MacroP and MacroR achieved by TERQAF-UN are very small on all the datasets. For example, the difference between MacroP and MacroR achieved by TERQAF-UN on the UBook dataset is 1.78%, but the differences between MacroP and MacroR achieved by TERQAF, TERQAF-SVM, and TERQAF-NB are 6.72%, 3.93%, and 8.83%, respectively. Especially, there are big differences between MacroP and MacroR achieved by TERQAF, TERQAF-SVM, and TERQAF-NB on the iShopping dataset. From this point of view, TERQAF-UN may be more suitable for test report quality assessment without considering the additional effort for parameter tuning. In addition, TERQAF-SVM ahieves better results than TERQAF-NB on most of the datasets.

Conclusion Logistic regression is more suitable for training the classifier than SVM and NB in predicting the quality of test reports. Meanwhile, logistic regression outperforms the original method proposed in our previous study.

7.3 Investigation into RQ3

Motivation In this study, we define a taxonomy of indicators based on the contained contents for test report quality measurement and classify them into four categories. These indicators can evaluate test reports from different but complementary aspects. They may have different impacts on the effectiveness of TERQAF in predicting the quality of test reports. Meanwhile, we are unclear whether each category has a positive impact on TERQAF in test report quality assessment and what is the most important category. In this RQ, we mainly focus on investigating the effect of each category of indicators in test report quality assessment and analyzing the degree of importance of each category of indicators.

Approach In general, to validate the role of one category of indicators, we should directly adopt all the relevant indicators belonging to this category to independently measure test reports. However, each category only contains three or four indicators. The previous experiment has demonstrated that the predicted results are easily influenced by one or several indicators when using a small number of indicators for test report quality assessment. Consequently, based on a reversed ideal, we can adopt the other three categories of indicators to indirectly reflect the effect of one category in evaluating the quality of test reports. In this experiment, we adopt the same parameter settings. In order to facilitate representation, we produce four variants of TERQAF, namely TERQAF-LAR, TERQAF-MAR, TERQAF-MLR, and TERQAF-MLA, where M, L, A, and R denote the morphological, lexical, analytical, and relational indicators, respectively.

Results We present the experimental results of different categories of indicators over the five datasets in Table 7. By an observation on the results in Tables 5 and 7, we find that TERQAF achieves better results than TERQAF-MAR, TERQAF-MLR, and TERQAF-MLA in terms of MacroP, MacroR, and MacroF over all datasets. For example, compared with TERQAF-MAR, TERQAF-MLR, and TERQAF-MLA, TERQAF achieves 1.40%, 4.18%, and 16.27% improvement in terms of MacroP, 3.03%, 3.92%, and 14.80% improvement in terms of MacroR, and 2.34%, 4.05%, and 15.50% improvement in terms of MacroF

Table 7 Impact of different categories of indicators on experimental results

Dataset	TERQAF-LAR			TERQAF-MAR			TERQAF-MLR			TERQAF-MLA		
	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF	MacroP	MacroR	MacroF
UBook	84.70%	76.27%	80.27%	85.04%	78.77%	81.78%	84.21%	67.85%	75.15%	80.78%	68.78%	74.30%
Justforfun	77.81%	77.50%	77.65%	78.01%	82.49%	80.19%	76.88%	72.18%	74.45%	66.58%	69.10%	67.82%
SE-1800	86.74%	75.10%	80.50%	87.10%	75.90%	81.12%	83.39%	70.77%	76.57%	72.27%	52.02%	60.50%
iShopping	82.82%	67.01%	74.08%	88.54%	65.91%	75.57%	80.62%	67.13%	73.26%	78.88%	59.35%	67.74%
CloudMusic	85.25%	73.35%	78.85%	84.94%	77.00%	80.78%	82.47%	72.46%	77.14%	70.38%	61.58%	65.69%

over the CloudMusic dataset, respectively. In addition, TERQAF outperforms TERQAF-LAR in terms MacroF over the iShopping and CloudMusic datasets, but TERQAF-LAR outperforms TERQAF in terms of MacroF over the UBook dataset. To explain the potential reason, we perform an investigation on the ground truth. We discover that 21, 36, 29, 31, and 11 test reports in the five datsets contain the related terms of the lexical indicators, and 13 out of 30 good-quality test reports contain imprecise terms and anaphoric terms over the UBook dataset, this may introduce a local bias that the indicators *imprecise terms* and *anaphoric terms* are considered as good over this dataset in training the classifier.

Comparatively, TERQAF-MAR outperforms TERQAF-LAR, TERQAF-MLR, and TERQAF-MLA in terms of MacroF over all datasets, in terms of MacroP but the CloudMusic dataset, and in terms of MacroR but the iShopping dataset. For example, TERQAF-MAR improves TERQAF-LAR, TERQAF-MLR, and TERQAF-MLA by 0.36%, 3.71%, and 14.83% in terms of MacroP, 0.80%, 5.13%, and 23.88% in terms of MacroR, and 0.62%, 4.55%, and 20.62% in terms of MacroF over the SE-1800 dataset, respectively. The results directly reflect that the lexical indicators work poorly in test report quality assessment. In contrast, TERQAF-MLA achieves the poorest results in terms of MacroP, MacroR, and MacroF over most of the datasets, which demonstrates that the relational indicators are the most important. In addition, although the performance of TERQAF-MLR is approximate to that of TERQAF-LAR, TERQAF-LAR achieves better results than TERQAF-LAR in terms of MacroF, MacroR, and MacroF over all the datasets but iShopping, which indicates that the analytical indicators are more important than the morphological indicators.

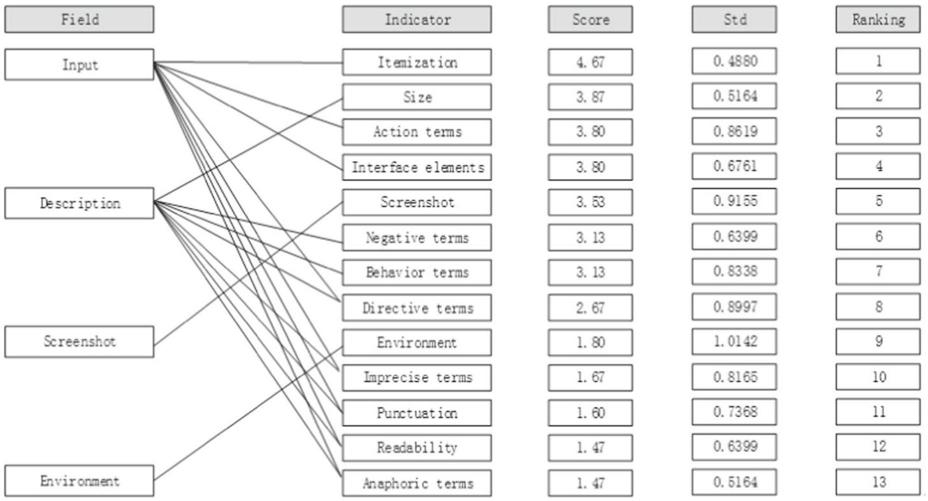
Conclusion Different categories of indicators have different effects on TERQAF in evaluating the quality of test reports. The most important category is the relational indicators and the poorest category is the lexical indicators. Comparatively, the role of the analytical indicators is slightly better than that of the morphological indicators.

7.4 Investigation into RQ4

Motivation Although a series of indicators are defined to measure the quality of test reports, we are unclear about which indicators are mainly concerned by developers when evaluating test reports. In this RQ, we attempt to investigate the degree of importance of each indicator and validate whether the indicators are effective in a real scenario by manual evaluation.

Approach In this experiment, we invite 15 developers from the five companies (each company includes 3 developers). They are very familiar with these mobile applications and acknowledge test reports. Due to the cooperative relationship, the developers are willing to conduct a manual evaluation. For each indicator, we give the concrete definition and explain how it measures the quality of test reports in detail. First, each developer scores each indicator on a scale of 5 points according to the importance degree in evaluating test reports. Then, the average score of each indicator is calculated by the obtained results. Finally, we rank the indicators in a descending order based on the average scores. In practice, this evaluation procedure is simple, thus developers only take about 10 minutes to complete the whole evaluation.

Results The results of manual evaluation are presented in Fig. 3, where the connect lines connects the fields an indicator can evaluate, “score” is the average score of the 15 developers, and “std” represents the standard deviation of the scores. As shown in the figure,



Indicator	Score	Std	Ranking
Itemization	4.67	0.4880	1
Size	3.87	0.5164	2
Action terms	3.80	0.8619	3
Interface elements	3.80	0.6761	4
Screenshot	3.53	0.9155	5
Negative terms	3.13	0.6399	6
Behavior terms	3.13	0.8338	7
Directive terms	2.67	0.8997	8
Environment	1.80	1.0142	9
Imprecise terms	1.67	0.8165	10
Punctuation	1.60	0.7368	11
Readability	1.47	0.6399	12
Anaphoric terms	1.47	0.5164	13

Fig. 3 The results from developers for each indicator

8 indicators are used to evaluate the *input* and the *description*. In contrast, only one indicator is used to measure the *screenshot* and the *environment*. According to the results, we observe that the most important indicator is *itemization* which obtains 4.67 scores. The potential reason is that this indicator is directly related to test steps which are the most important information for reproducing the bugs. Therefore, the indicators *action terms* and *interface elements* also gain high scores since they directly reflect whether the *input* is associated with test steps. The indicators *size*, *negative terms*, and *behavior terms* mainly evaluate the *description* and obtain relatively high scores. In contrast, the indicators *imprecise terms*, *punctuation*, *readability*, and *anaphoric terms* receive low scores. This may be because test reports contain a small number of imprecise and anaphoric terms. Meanwhile, the readability of test reports is generally good and developers are familiar with Chinese grammars. Therefore, the indicator *readability* is not concerned by developers. Also, the sentence length is often acceptable in test reports, thus developers do not care about the indicator *punctuation*. In addition, we observe that the indicator *environment* has the maximum standard deviation 1.0142, followed by the indicator *screenshot*. The main reason is that some developers think the environment and screenshots are important because these information can help them acknowledge the adaptability of the system and the abnormal behaviors, but other developers think these information is negligible in fixing bugs. Similarly, the indicators *directive terms*, *action terms*, *behavior terms*, and *imprecise terms* obtain high standard deviation.

Conclusion. The most important indicators are *itemization*, *size*, *action terms*, and *interface elements*. Other indicators, such as *imprecise terms*, *punctuation*, *readability*, and *anaphoric terms*, play weak roles in test report quality assessment.

8 Threats to Validity

In this section, we discuss the threats to validity from three aspects, including internal validity, external validity, and construct validity.

8.1 Internal validity

Term lists. In TERQAF, some indicators (e.g., imprecise terms and directive terms) rely on term lists to measure the quality of test reports. However, it is hard to obtain the overall term lists, especially domain terms, which may have an impact on the effectiveness of TERQAF in our experiments. Fortunately, many studies about quality assessment have summarized overall term lists which are reused in other studies. We also reuse these term lists by translating them into Chinese and add some relevant terms by leveraging the thesaurus. Thus, the ultimately used term lists are relatively complete. In such a way, the impact can be greatly reduced.

8.2 External validity

Natural Language Selection In our experiments, test reports are written in Chinese and most of the indicators are defined based on the Chinese text. Admittedly, there is a big gap between Chinese and English as well as other Latin languages, which may produce a bias to TERQAF in different natural languages. However, NLP techniques have been widely adopted to process documents written in different languages, we only need to choose appropriate processing steps and NLP tools. Meanwhile, even though some indicators are designed based on Chinese text, they derive from existing related studies and have been proven to adapt to English documents. Therefore, this bias is negligible.

Manual Evaluation In the experiments, we invite 15 developers to conduct manual validation for the defined indicators and 5 developers to complete the reproduction. Due to the experience and subjectivity, threats may be introduced to evaluation results. However, these developers have rich experience of software development and are familiar with the five applications and test reports. Also, reproduction tasks are simple and the developers only need exactly follow test steps to reproduce bugs. In addition, we list detailed criteria for the manual evaluation and bug reproduction. Therefore, this threat can be minimized.

8.3 Construct validity

Property Selection To measure the quality of test reports, we select some desirable properties to characterize expected test reports. However, no study is conducted to investigate what are the real desirable properties for developers, which may threaten the validity of this study. Actually, given that test reports are similar to bug reports, the desirable properties of bug reports are also adaptive to test reports. By investigating many studies related to bug reports, we summarize some desirable properties to characterize test reports. Although there may be some other properties, to the best of our knowledge, the selected properties in this study are more important. Therefore, this threat can be significantly reduced.

Indicator Selection In TERQAF, we define a series of quantifiable indicators which are classified into four categories. There may be some other indicators that can be used to measure the quality of test reports, this may produce a bias to the effectiveness of TERQAF. In practice, Génova et al. (2013) defined more indicators to measure the quality of requirement specifications and validated their effectiveness. We select some indicators which are adaptive to evaluate the quality of test reports and are related to the criteria which the developers adopted to score test reports. In addition, we also add some new indicators according to the characteristics of test reports. Therefore, the bias will be minimized.

Besides, we construct the relationship between the desirable properties and the measurable indicators based on existing studies (Génova et al. 2013) and our experience. Due to the experience level, this may be a threat to the validity of TERQAF. In practice, we have conducted a detailed discussion about this problem and consulted the developers. Meanwhile, the relationship establishment is based on the scoring criteria of test report evaluation. Therefore, this threat is subtle.

9 Related Work

In this section, we discuss two areas of studies related to our work, namely crowdsouced testing and quality assessment for textual documents.

9.1 Crowdsourced Testing

The concept of crowdsourcing was first proposed by Howe in 2006, which refers to the act of an company crowdsourcing their work to a large potentially undefined group of online individuals in open call (Howe 2006). In general, it combines human and machine power to resolve problems (Mao et al. 2015). Due to the high efficiency and low cost (Guaiani and Muccini 2015), crowdsourcing has been widely applied to software testing activities.

As a newly emergent technique, many studies have been conducted to investigate the potential of crowdsourced testing. For example, an empirical investigation showed that crowdsourced testing can compensate traditional laboratory testing (Guaiani and Muccini 2015). Leicht et al. conducted two case studies to reveal the advantages of crowdsourced testing (2016). Compared with traditional in-house testing, crowdsourced testing can bring many advantages in terms of speed, cost, and user feedback. In contrast, some studies focus on resolving software engineering problems by applying crowdsourced testing (Dolstra et al. 2013; Liu et al. 2012). For example, Wu et al. presented a new framework which adopted crowdsourcing to quantify the QoE of multimedia content. The framework can support cheat detection and simplify the rating procedure (Wu et al. 2013). Gomide et al. proposed an events detection algorithm to support crowdsourcing software usability testing. It leverages human-computer interfaces to identify users emotions by processing user's actions from mouse movements or touch events (Gomide et al. 2014). Given that the GUI testing is costly and time-consuming, Vliegendhart et al. recruited hundreds of workers to perform A/B testing for a multimedia application through the Amazon's crowdsourcing platform Mechanical Turk (2012). In addition, some studies introduce crowdsourced testing to mobile applications (Gao et al. 2018). Zhang et al. provided a systematical tutorial for mobile application testing and detailed the obvious difference between crowdsourced testing and traditional lab-based mobile testing (Zhang et al. 2017).

Some studies attempt to resolve the problems existing in crowdsourced testing (Chen and Luo 2014; Starov 2013; Chen et al. 2019). To select appropriate workers for testing, Cui et al. presented a multi-objective crowd worker selection approach which aims to maximize the coverage of test requirement and bug-detection experience, and minimize the cost (Cui et al. 2017). Taking the poor performance of workers, Zhang et al. proposed an approach to help workers to acquire domain knowledge and guide them to complete test tasks (Zhang et al. 2016). To help developers identify more bugs when detecting a given number of test reports, Feng et al. attempted to resolve test report prioritization by leveraging textual information and screenshots (Feng et al. 2015; Feng et al. 2016). Given that false positive test reports will take developers unnecessary time, Wang et al. adopted text-based classification approaches (Wang et al. 2016; 2017) and machine learning techniques (Wang et al. 2016) to identify them from a large number of test reports. Under the motivation of reducing the inspection cost, Jiang et al. developed a new framework to partition test reports into clusters in which test reports in one cluster detail the same bug (Jiang et al. 2018). In addition, to help developers determine the severity lever of test reports, Guo et al. proposed a knowledge transfer classification technique which leverages similar information contained in bug reports (Guo et al. 2017). Different from the above studies, this study aims to evaluate the quality of test reports to help developers select high-quality test reports for inspection, thus accelerating inspection efficiency.

9.2 Quality Assessment for Textual Documents

Quality is an ambiguous concept which is related to individually subjective judgements. Given a textual document, different users may have different opinions about the quality. In literature, researchers have conducted many studies to investigate the quality of textual documents, such as bug reports and requirement specifications.

In software maintenance, bug report resolution is one of the most important tasks (Nazar et al. 2016). However, the quality has a great impact on the inspection efficiency of bug reports. Motivated by this, researchers have conducted extensive studies to investigate the quality of bug reports. Zimmermann et al. (2010) performed an investigation about what makes a good bug report. They revealed that there exists a serious mismatch between what developers need and what users provide. An effective tool named CUEZILLA was developed to quantify the quality of bug reports by extracting domain features to detect the contained information (Zimmermann et al. 2010; Bettenburg et al. 2008). To help developers determine which bug reports can be selected for inspection, Hooimeijer and Weimer defined some external features based on the surface characteristics of text to build a descriptive model to measure the quality of bug reports (2007). Given the lengthy text contained in bug reports, some studies attempted to extract important and informative sentences to generate short summary information, thus improving the inspection efficiency of bug reports for developers (Rastkar et al. 2014). In addition, some studies focus on the effect of duplicate bug reports. For example, Bettenburg et al. conducted an empirical investigation on duplicate bug reports (2008). They pointed out that duplicate bug reports usually provided additionally useful information for developers to understand and fix the bug more efficiently. Thus developers should aggregate them to produce an extended bug report rather than discarding them straightforwardly. Perry pointed out what testers should concern when delivering good test reports and presented the detailed workbench for reporting test results (2006). Another study was conducted to characterize quality standards about reporting of test results, thus ensuring that the reported results will be properly understood (None 2014).

In requirement engineering, requirement specifications play an important role in guaranteeing the overall quality of software. A great amount of effort is devoted to generate and refine requirement specifications (Heck and Zaidman 2016). Therefore, timely detecting the erroneous requirements can avoid serious problems, such as spending significant additional costs or failing to satisfy the expected demands. However, manual quality assessment for requirement specifications is tedious and time-consuming. Under this motivation, some researchers try to explore automated techniques and tools to help developers perform the quality assessment for requirement specifications (Zogaj et al. 2014; Rosenberg and Hammer 1999). The generic method is to define some quantifiable indicators to measure the desirable properties of requirement specifications. For example, Génova et al. (2013) summarized 15 indicators based on the textual contents and document structure to evaluate the 13 desirable properties. Carlson and Laplante (2014) also explored some new indicators to quantify the quality of specifications and conducted analytical experiments to validate the distribution of indicators in requirement specifications. Another body of studies focuses on machine learning. For example, Parra et al. (2015) adopted rule induction techniques to evaluate the quality of requirement specifications. The experimental results demonstrated that the proposed method works well in quality assessment. In addition, some studies are conducted to investigate a single property, such as ambiguities (Kiyavitskaya et al. 2008; Popescu et al. 2007), inconsistencies (de Sousa et al. 2010), and conflicts (Sardinha et al. 2013).

10 Conclusion

To help developers determine whether a test report can be selected for inspection, this paper issues a new problem of test report quality assessment and present our attempts towards resolving this problem by classifying test reports as either "Good" or "Bad". First, we define some desirable properties and a taxonomy of indicators for modeling the quality of test reports. Then, we adopt an efficient Chinese NLP tool to process crowdsourced test reports and compute the numerical value of each indicator based on the contained contentst. Finally, a classifier based on logistic regression is trained to predict the quality of test reports. We conduct extensive experiments over five crowdsourced test report datasets of mobile applications. The experimental results demonstrate that the proposed method can predict the quality of test reports with high accuracy.

Besides, this study also provides some suggestions on writing a good test report. For example, workers should use some domain terms to describe the bug and avoid ambiguous terms as far as possible. The size of test reports should be kept in an acceptable length. However, the quality assessment of test reports stills depend on the manual judgement of developers. Nonetheless, this study also plays an auxiliary role in manually performed quality assessment. In future, we will define more desirable properties and measurable indicators to quantify the quality of test reports and investigate the concrete effect of each indicator. Meanwhile, we should collect more test reports to validate the generalizability of TERQAF in other projects and languages. In addition, we try to design a practicable tool to implement our method and deploy it in a real scenario, such as the National Student Contest of Software Testing in China.[11]

Acknowledgments We greatly thank the developers who devote their precious time on evaluating and inspecting the quality of test reports. We would thank José M. Fuentes who provides help for us to conduct this work. This work is partially supported by the National Key Research and Development Program of China under grant no. 2018YF-B1003900, and the National Natural Science Foundation of China under Grants No. 61902096, 61972359, 61370144, 61722202, 61403057, and 61772107.

References

Aceto G, Ciuonzo D, Montieri A, Pescapè A (2018) Multi-classification approaches for classifying mobile app traffic. J Netw Comput Appl 103:131–145

[11] http://mooctest.org/

Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T (2008) What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, ser. FSE'08. ACM, pp 308–318

Bettenburg N, Premraj R, Zimmermann T, Kim S (2008) Duplicate bug reports considered harmful... really? In: 24th IEEE international conference on software maintenance, ser. ICSM'08, pp 337–345

Carlson N, Laplante PA (2014) The NASA automated requirements measurement tool: a reconstruction. ISSE 10(2):77–91

Chen Z, Luo B (2014) Quasi-crowdsourcing testing for educational projects. In: Companion proceedings of the 36th international conference on software engineering, ser ICSE'14. ACM, pp 272–275

Chen X, Jiang H, Li X, He T, Chen Z (2018) Automated quality assessment for crowdsourced test reports of mobile applications. In: 25th international conference on software analysis, evolution and reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018. IEEE, pp 368–379

Chen X, Jiang H, Chen Z, He T, Nie L (2019) Automatic test report augmentation to assist crowdsourced testing. Frontiers of Computer Science (print)(5)

Cui Q, Wang S, Wang J, Hu Y, Wang Q, Li M (2017) Multi-objective crowd worker selection in crowdsourced testing. In: The 29th international conference on software engineering and knowledge engineering, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 5-7, 2017, pp 218–223

de Sousa TC, Almeida JR Jr, Viana S, Pavón J (2010) Automatic analysis of requirements consistency with the B method. ACM SIGSOFT Software Engineering Notes 35(2):1–4

Denoeux T (2018) Logistic regression revisited: Belief function analysis. In: Belief functions: theory and applications - 5th international conference, BELIEF 2018, Compiégne, France, September 17-21, 2018, Proceedings, pp 57–64

Dolstra E, Vliegendhart R, Pouwelse JA (2013) Crowdsourcing gui tests. In: Sixth IEEE international conference on software testing, verification and validation, ser. ICST'13. IEEE, pp 332–341

Feng Y, Chen Z, Jones JA, Fang C, Xu B (2015) Test report prioritization to assist crowdsourced testing. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE'15. ACM, pp 225–236

Feng Y, Jones JA, Chen Z, Fang C (2016) 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, pp 202–213

Flesch R (1948) A new readability yardstick. Journal of Applied Psychology 32(3):221

Gao R, Wang Y, Feng Y, Chen Z, Wong WE (2018) Successes, challenges, and rethinking – an industrial investigation on crowdsourced mobile application testing. Empir Softw Eng 2:1–25

Génova G, Fuentes JM, Morillo JL, Hurtado O, Moreno V (2013) A framework to measure and improve the quality of textual requirements. Requir Eng 18(1):25–41

Gomide VH, Valle PA, Ferreira JO, Barbosa JR, Da Rocha AF, Barbosa T (2014) Affective crowdsourcing applied to usability testing. Int J Comput Sci Inf Technol 5(1):575–579

Guaiani F, Muccini H (2015) 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. ACM/IEEE, pp 32–37

Guo S, Chen R, Li H (2017) Using knowledge transfer and rough set to predict the severity of android test reports via text mining. Symmetry 9(8):161

Guo W (2010) Research on readability formula of chinese text for foreign students. Ph.D. dissertation, Shanghai Jiao Tong University

Heck P, Zaidman A (2016) A systematic literature review on quality criteria for agile requirements specifications. Softw Qual J: 1–34

Férnandez HJ (1959) Medidas sencillas de lecturabilidad. Consigna (214):29–32

Hooimeijer P, Weimer W (2007) Modeling bug report quality. In: 22nd IEEE/ACM international conference on automated software engineering (ASE 2007), ser ASE'07. ACM, pp 34–43

Howe J (2006) The rise of crowdsourcing. Wired Magazine 14(6):1–4

Hsu H, Chang YI, Chen R (2019) Greedy active learning algorithm for logistic regression models. Computational Statistics & Data Analysis 129:119–134

Jiang H, Chen X, He T, Chen Z, Li X (2018) Fuzzy clustering of crowdsourced test reports for apps. ACM Trans Internet Techn 18(2):18:1?18:28

Jiang H, Zhang J, Li X, Ren Z, Lo D (2016) A more accurate model for finding tutorial segments explaining apis. In: IEEE 23rd international conference on software analysis, evolution, and reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016, vol 1, pp 157–167

Joorabchi ME, MirzaAghaei M, Mesbah A (2014) Works for me! characterizing non-reproducible bug reports. In: 11th working conference on mining software repositories, MSR 2014, Proceedings, ser. MSE?14, pp 62–71

Kiyavitskaya N, Zeni N, Mich L, Berry DM (2008) Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requir Eng 13(3):207–239

Ko AJ, Myers BA, Chau DH (2006) 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, pp 127–134

Leicht N, Knop N, Müller-Bloch C, Leimeister JM (2016) When is crowdsourcing advantageous? the case of crowdsourced software testing. In: 24th European conference on information systems, ECIS 2016, Istanbul, Turkey, June 12-15, 2016, p Research Paper 60

Liu D, Lease M, Kuipers R, Bias RG (2012) Crowdsourcing for usability testing. American Society for Information Science and Technology 49(1):332–341

Liu Z, Gao X, Long X (2010) Adaptive random testing of mobile application. In: International conference on computer engineering and technology, pp V2–297 – V2–301

Mao K, Capra L, Harman M, Jia Y (2015) A survey of the use of crowdsourcing in software engineering. RN 15(01)

Nazar N, Jiang H, Gao G, Zhang T, Li X, Ren Z (2016) Source code fragment summarization with small-scale crowdsourcing based features. Frontiers of Computer Science 10(3):504–517

Nebeling M, Speicher M, Grossniklaus M, Norrie MC (2012) Crowdsourced web site evaluation with crowdstudy. In: Proceedings of 12th International Conference on Web Engineering, ser ICWE'12. Springer, pp 494–497

None (2014) Itc guidelines on quality control in scoring, test analysis, and reporting of test scores. Int J Test 14(3):195–217

Parra E, Dimou C, Morillo JL, Moreno V, Fraga A (2015) A methodology for the classification of quality of requirements using machine learning techniques. Information & Software Technology 67:180–195

Perry WE (2006) Effective methods for software testing, 3rd edn. Wiley, Hoboken

Petrosyan G, Robillard MP, Mori RD (2015) Discovering information explaining API types using text classification. In: 37th IEEE/ACM international conference on software engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, vol 1, pp 869–879

Popescu D, Rugaber S, Medvidovic N, Berry DM (2007) Reducing ambiguities in requirements specifications via automatically created object-oriented models. In: Innovations for requirement analysis. From stakeholders? needs to formal designs, pp 103–124

Rastkar S, Murphy GC, Murray G (2014) Automatic summarization of bug reports. IEEE Trans Software Eng 40(4):366–380

Rosenberg L, Hammer T (1999) A methodology for writing high quality requirement specifications and for evaluating existing ones. NASA Goddard space flight center software assurance technology center

Sardinha A, Chitchyan R, Weston N, Greenwood P, Rashid A (2013) Ea-analyzer: automating conflict detection in a large set of textual aspect-oriented requirements. Autom Softw Eng 20(1): 111–135

Starov O (2013) Cloud platform for research crowdsourcing in mobile testing. East Carolina University

Thakurta R (2013) A framework for prioritization of quality requirements for inclusion in a software project. Softw Qual J 21(4):573–597

Vliegendhart R, Dolstra E, Pouwelse J (2012) Crowdsourced user interface testing for multimedia applications. In: ACM multimedia 2012 workshop on crowdsourcing for multimedia, pp 21–22

Wang J, Cui Q, Wang Q, Wang S (2016) 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, pp 6:1–6:10

Wang J, Cui Q, Wang S, Wang Q (2017) Domain adaptation for test report classification in crowdsourced testing. In: 39th IEEE/ACM international conference on software engineering: software engineering in practice track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017. IEEE, pp 83–92

Wang J, Wang S, Cui Q, Wang Q (2016) 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, pp 190–201

Wilson W, Rosenberg L, Hyatt L (1996) Automated quality analysis of natural language requirement specifications in proc. In: Fourteenth annual pacific northwest software quality conference Portland OR

Wu C, Chen K, Chang Y, Lei C (2013) Crowdsourcing multimedia qoe evaluation: a trusted framework. IEEE Trans Multimed 15(5):1121–1137

Yang S-j (1970) A readability formula for Chinese language. University of Wisconsin–Madison

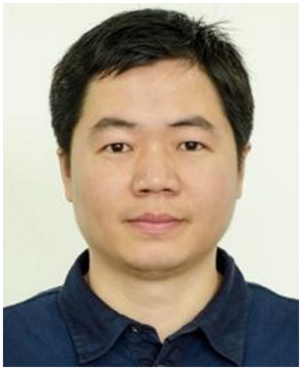
Zhang T, Gao JZ, Cheng J (2017) Crowdsourced testing services for mobile apps. In: in 2017 IEEE symposium on service-oriented system engineering, SOSE 2017, San Francisco, CA, USA, April 6-9, 2017, pp 75–80

Zhang X, Chen Z, Fang C, Liu Z (2016) Guiding the crowds for android testing. In: Proceedings of the 38th international conference on software engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume, pp 752–753

Zimmermann T, Premraj R, Bettenburg N, Just S, Schröter A., Weiss C (2010) What makes a good bug report. IEEE Trans Software Eng 36(5):618–643

Zogaj S, Bretschneider U, Leimeister JM (2014) Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. Journal of Business Economics 84(3):375–405

Xin Chen is currently a Lecturer in School of Computer Science, Hangzhou Dianzi University. He has received his Ph.D. degree from Dalian University of Technology in June 2018. His main research interests include mining software repositories and search based software engineering. He is also a member of the ACM and the CCF (China Computer Federation).



He Jiang received the Ph.D. degree in computer science from the University of Science and Technology of China, China. He is currently a Professor in Dalian University of Technology, China. He is also a member of the ACM and the CCF (China Computer Federation). He is one of the ten supervisors for the Outstanding Doctoral Dissertation of the CCF in 2014. His current research interests include Search-Based Software Engineering (SBSE) and Mining Software Repositories (MSR). His work has been published at premier venues like ICSE, SANER, and GECCO, as well as in major IEEE transactions like TSE, TKDE, TSMCB, TCYB, and TSC.

Xiaochen Li received the doctoral degree in software engineering from the Dalian University of Technology, China in 2019 under supervision with Prof. He Jiang. He is currently a research associate at Software Verification and Validation research group in University of Luxembourg, headed by Prof. Lionel Briand. His current research interests are intelligent software engineering and software semantic analysis.

Liming Nie received the Ph.D. degree in Computer Application Technology from the Dalian University of Technology, Dalian, China, in 2017. He is currently a lecturer with Zhejiang Sci-Tech University, Hangzhou, China. His current research interests include Intelligent Software Development, and Intelligent Cyber Systems. Dr. Nie is a member of the ACM and the CCF.

Dongjin Yu is currently a professor at Hangzhou Dianzi University, China. His research efforts include intelligence software engineering, big data, service computing and business process management. He is the director of Institute of Big Data (IBD) and Institute of Computer Software (ICS) of Hangzhou Dianzi University. He is a member of ACM and IEEE, and a senior member of China Computer Federation (CCF). He is also a member of Technical Committee of Software Engineering CCF (TCSE CCF) and a member of Technical Committee of Service Computing CCF (TCSC CCF).

Tieke He is currently a research assistant at Software Institute, Nanjing University, Nanjing. He got his B.S., M.S. and Ph.D. degrees in software engineering from Nanjing University, Nanjing, in 2010, 2012, and 2017, respectively. His research interests include recommender systems and knowledge graph.

Zhenyu Chen is a professor at Software Institute, Nanjing University, Nanjing. He got his B.S. and Ph.D. degrees in mathematics from Nanjing University, Nanjing, in 2001 and 2006, respectively. His research interests include intelligent software engineering and mining software repositories. He is a member of the CCF and the ACM.

Affiliations

Xin Chen[1] · He Jiang[2] · Xiaochen Li[2] · Liming Nie[3] · Dongjin Yu[1] · Tieke He[4] · Zhenyu Chen[4]

He Jiang
jianghe@dlut.edu.cn

Xiaochen Li
li1989@mail.dlut.edu.cn

Liming Nie
Lmn@zstu.edu.cn

Dongjin Yu
yudj@hdu.edu.cn

Tieke He
dg1232002@smail.nju.edu.cn

Zhenyu Chen
zychen@nju.edu.cn

[1] College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China

[2] School of Software, Dalian University of Technology, Dalian, China

[3] School of Information Science and Technology, Zhejiang Sci-tech University, Hangzhou, China

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