Generating Descriptions for Screenshots to Assist Crowdsourced Testing

Di Liu, Xiaofang Zhang
School of Computer Science and Technology
Soochow University
Suzhou, China
dliu0721@stu.suda.edu.cn
xfzhang@suda.edu.cn

Yang Feng, James A. Jones
Department of Informatics
University of California, Irvine
Irvine, California, USA
{yang.feng, jajones}@uci.edu

***Abstract*—Crowdsourced software testing has been shown to be capable of detecting many bugs and simulating real usage scenarios. As such, it is popular in mobile-application testing. However in mobile testing, test reports often consist of only some screenshots and short text descriptions. Inspecting and understanding the overwhelming number of mobile crowdsourced test reports becomes a time-consuming but inevitable task. The paucity and potential inaccuracy of textual information and the well-defined screenshots of activity views within mobile applications motivate us to propose a novel technique to assist developers in understanding crowdsourced test reports by automatically describing the screenshots. To reach this goal, in this paper, we propose a fully automatic technique to generate descriptive words for the well-defined screenshots. We employ the test reports written by professional testers to build up language models. We use the computer-vision technique, namely Spatial Pyramid Matching (SPM), to measure similarities and extract features from the screenshot images. The experimental results, based on more than 1000 test reports from 4 industrial crowdsourced projects, show that our proposed technique is promising for developers to better understand the mobile crowdsourced test reports.**

I. Introduction

Crowdsourced techniques have been widely used to resolve various software engineering tasks, *e.g.,* testing [7], development [12] and design [11] Especially, for software testing, compared with conventional methods, crowdsourced testing is able to simulate real usage scenarios and provide real users' feedbacks. Thus, it is widely adopted to test mobile applications, which need rapid development-and-deployment iterations and support diverse mobile platforms.

Typically, when conducting mobile testing, crowd workers are required to submit their feedback in the form of *test reports*, which may contain various kinds of information, such as text descriptions, screenshots, voice record, and operation videos. To understand these test reports, developers need to diagnose these test reports. However, because crowdsourced testing often involves a large number of crowd workers, and as such, results in an overwhelming amount of test reports, diagnosing and understanding these reports becomes a time-consuming but inevitable task [6].

To improve the efficiency of diagnosing test reports, software-engineering researchers have proposed many techniques. In almost all such techniques, the test reports are captured and analyzed based on their textual similarity (*e.g.,* [6], [8], [10], [18], [20]–[22]) or their execution traces (*e.g.,* [4], [5], [16], [24]). However, these two categories of information are often insufficient or unavailable in crowdsourced mobile testing. While execution traces are hard to collect, Zhang *et al.* find that the text description of mobile application bug reports are often shorter than the desktop software [26]. This fact fundamentally disables or impedes the applications of these conventional test-report management techniques on crowdsourced mobile testing.

In this paper, we present a method to make the test reports with rich screenshots and insufficient text descriptions easier to understand and also support textual analysis among multiple test reports to understand trends in the bugs being reported. The goal of this approach to generate descriptive keywords for the screenshots of crowdsourced mobile test reports. This approach is a fully automatic learning-based technique to assist developers to understand crowdsourced mobile test reports which are often lacking sufficient accurate text descriptions but are rich in screenshots. For the screenshots analysis, we employed the Spatial Pyramid Matching (SPM) [13] technique to measure the similarity of screenshots and apply natural-language processing techniques to analyze the text descriptions of the well-written test reports to get the weighted keywords. Further, with weighted keywords and similarity matrix, we cluster the screenshots and further build language models for each of these image clusters. Based on these language models, we can generate descriptive keywords for the unseen screenshots to assist developers in understanding the test reports.

The main contributions of this paper are as follows:

- We propose an image-understanding-based technique to generate descriptive keywords for the screenshots in the crowdsourced test reports. To the best of our knowledge, this is the first work to generate descriptive keywords for screenshots to assist developers to diagnose and understand crowdsourced mobile test reports.
- To validate our technique, we collaborated with the largest crowdsourced testing platform of China, Baidu

978-1-5386-4969-5/18/$31.00 © 2018 IEEE

SANER 2018, Campobasso, Italy
ERA Track

Crowd Test[1]. We conducted a preliminary experiment using our technique on four industrial crowdsourced projects. The experimental result shows that our technique can reach a high precision, as well as recall, for the generated descriptive keywords.

II. Motivation

With the increasing popularity of mobile devices, ensuring the quality of mobile applications has become a crucial software engineering task. On mobile devices, images have played a crucial role in sharing, expressing and exchanging information. Regarding software testing, Zhang *et al.* [26] found that the test reports of mobile applications contain much shorter text descriptions and more screenshots in comparison with the test reports of desktop software. Reporters prefer taking screenshots of the well-defined activity views of these applications rather than inputing long paragraphs to describe bugs.

For the prevalent mobile testing method, crowdsourced testing, even though taking screenshots simplifies the procedure of submitting test reports, the shortage of text description brings difficulties to developers in comprehending these reports. When diagnosing these mobile crowdsourced test reports, to understand the bugs and reproduce them, developers often need to read and analyze the contents manually, including the bug description, operations, screenshots, and environment information so on. In this procedure, these screenshots are helpful for developers to locate the bug on the activity level rapidly, but it is hard to reproduce the faults based on insufficient the descriptive words. Further, one major goal of crowdsourced software testing is to get the feedbacks from diverse usage settings. Thus, the crowd workers may have different knowledge, background, use habit, and software and hardware configurations. This fact makes the quality of descriptions of crowdsourced test reports varying widely. The poorly-written and inaccurate descriptions may be difficult to understand or misleading for triagers. The cognitive gaps between developers and bug reporters bring difficulties to fixing software bugs. Thus, automatically retrieving information from these screenshots and generating descriptive words to present high-level understanding of the bugs are helpful for assisting developers in processing these test reports and reducing the cost.

III. Approach

A. Preliminary

For crowdsourced software testing, various forms of multimedia information exist in test reports, such as voice messages and operation videos. Among these kinds of information, text description and screenshots are the two most widely used. In this paper, we focus on analyzing the content of screenshots to generate descriptions for them to assist software developers to comprehend the test reports. We use and analyze only those two parts of the test reports, the textual description and the set of screenshots. In other words, the test report set $TR(r) = \{tr(S_i, T_i)|i = 0 \dots n\}$, in which, S denotes the screenshots (*i.e.,* images) and T denotes the text describing the bug and the operation steps. Note that each *test report* often contains more than one screenshot, *i.e.,* for the screenshot set S_i of test report tr_i, we have $S_i = s_{i1}, s_{i2}, \dots, s_{im}$, in which, s_{ij} denotes the jth screenshot in test report tr_i. In this paper, we aim at training language models from well-written test reports to generate descriptive keywords for screenshots.

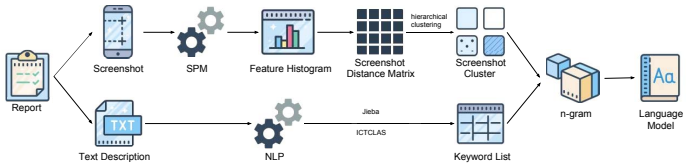


Fig. 1: Model-Building Phase

B. Technique Framework

The technique framework consists of two major phases: a model building phase and an application phase. In the model building phase, we aim at employing the text description of training test reports and building the language models for the training screenshots, and in the application phase, the major goal is to employ these language models to generate proper descriptive words for the new screenshots. In this section, we introduce the details of our technique framework.

1) Model-Building Phase: The model-building phase is composed of the following sub-steps: (1) analyzing text descriptions of well-written test reports; (2) extracting features from screenshots and clustering similar screenshots; and (3) building language models for each of the images clusters. Fig. 1 depicts this model-building phase.

Text processing. In software-engineering research, natural-language-processing techniques have been employed to assist various tasks. In our technique, we leverage the text description of the well-written test reports to build a corpus of descriptive words. The text processing consists of the following steps: (1) tokenization, (2) filtering, (3) keyword-vector building.

In our technique, we employ the Jieba[2] tool, a Python-based segmentation system, to tokenize the text description. We input the text information, which contains the description of the bugs and operations, of each mobile crowdsourced test report into Jieba and obtain a token list and corresponding part-of-speech tagging. However, this preliminary raw outputs also contain noise, such as the typos and uncommon words. To build up the language models that are capable of generating keywords to describe operations and situations, we need to identify the keyword from the raw outputs. Prior studies show that verbs and nouns are most important to reflect the content of a document [17], [25]. Hence, we retained only verbs and nouns as candidate keywords of test reports and filtered out other words. To reduce the bias induced by stop words, we

[1]http://test.baidu.com

[2]https://github.com/fxsjy/jieba

also filter them out based on the ICTCLAS stop word list[3]. After these two steps, we can obtain a word sequence for each of the training test reports.

Screenshot processing. For mobile applications, each activity view is designed to meet some functional requirement, and almost all of the screenshots come from the activity view of the mobile applications. This fact motivates us to group all of these screenshots into several clusters and further build a language model for each of these image clusters. To reach this goal, the screenshot processing procedure consists of three fundamental steps: (1) building feature histograms, (2) building a distance matrix, (3) clustering similar screenshots.

First, because the screenshots provide not only the buggy symptoms but also app-specific visual appearances, variable resolution, and complex backgrounds, modeling the features merely based on naive RGB values is not a proper approach for our task. Thus, to address these challenges, we employ the Spatial Pyramid Matching (SPM) algorithm [13] to extract the scale-invariant feature transform (SIFT), which is known to be highly accurate in matching image features [23], from the screenshots. Based on the SIFT, our technique identifies similar layouts and widgets from the screenshots. After we obtained the feature histogram, we build a distance matrix over the image set. In our technique, we employ the Chi-Square distance metric to measure the dissimilarity between each pair of images.

To group the screenshots into clusters, we conduct agglomerative hierarchical clustering on the image set. There are two fundamental parameters influencing the clustering result of this algorithm: the linkage type, which defines the method of calculating the distance between clusters, and the threshold for determining the stop point of clustering. We refer the threshold value as α. One noteworthy point is that agglomerative hierarchical clustering algorithm can automatically stop the clustering when the distance between all of these clusters is larger than α. Thus, we can obtain the cluster set meeting distance criteria instead of pre-specifying the number of clusters.

Building Language Models. We present the process of building a language model for one screenshot cluster in Fig. 2. For each cluster, we identify the test reports that contain the screenshots in the cluster, and then we take the keyword lists generated from the text descriptions of these test reports as the inputs for the n-gram model. The n-gram model is a probabilistic language model for predicting the next item in a contiguous sequence, in the form of a $(N-1)$-order Markov model. Given the keyword lists, which come from the test reports containing the similar screenshots, the n-gram can build up the probabilistic language model for generating the words with various context.

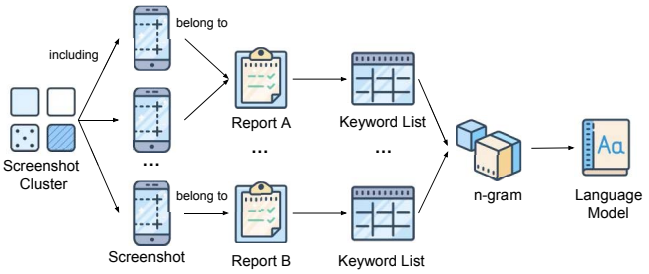


Fig. 2: The Process of Building Language Model

Additionally, all language models must address out-of-vocabulary (OOV) words, which refer the words that had a very low appearance frequency in the dictionary or the database during the model-building phase. In our technique, we set a threshold value X for identifying the OOV words. We replace the words that occur fewer than X times in the training set with the special symbol *unk*, and thereafter treat *unk* like a regular word.

2) Application Phase: For each report of the testing data set, we aim to generate words from the language models based on only the screenshots. First, for each screenshot in the test reports, we extract the SIFT feature for them and compute its K nearest neighbors in the training screenshot set. Applying the voting strategy on clusters of these K nearest neighbors, we can identify the cluster that the testing screenshot belongs to. Thus, for each test report we can identify the cluster sequence and corresponding language model sequence based on the screenshot list.

Because the n-gram language model is a kind of Markov model in which the previous states determine the current state, we employ the output of the previous one as the input for each language model in the sequence; we input the start symbol to initiate the generation. From each language model, we sequentially generate at most 20 words.

IV. Evaluation

We collaborated with the largest crowdsourced testing platform in China, Baidu Crowd Test, to validate our technique. The platform hires a large number of crowd workers to test applications, and collects mobile crowdsourced test reports. It provides us the crowdsourced test reports of four real mobile applications. These reports have been manually inspected and the professional testers have written operation steps and descriptions for them based on the original description and screenshots. It is worth noting that, in practice, *professional testers* invest efforts to diagnose each test report because they need to obtain a full understanding of the crowdsourced testing result. The major goal of our technique is to automatically generate a keyword list for the screenshots in all of these mobile crowdsourced test reports based on some well-written test reports and further improve the efficiency of diagnosing individual reports. Thus, to evaluate our technique framework, we conducted a ten-fold cross-validation over the test reports that have the text descriptions written by professional testers.

Table I shows the detailed information of these applications. All of these subjects are popular applications on the Android market. $\#TR$ denotes the number of test reports, $\#S$ denotes

[3]https://github.com/chdd/weibo/tree/master/stopwords

the number of screenshots, and $\#TR_m$ denotes the number of test reports that have more than one screenshot.

TABLE I: EXPERIMENTAL APPLICATION SUBJECTS

	Name	$\#TR$	$\#S$	$\#TR_m$
P1	Follower Wiki	184	331	103
P2	Jiajia Sport	273	336	221
P3	Cloud Driving School	288	366	279
P4	Hanan	311	452	295
	Total	1056	1465	898

We employ *precision*, referred as P and *recall*, referred as R, to evaluate our technique. In the test report of testing data set $TR' = \{tr_i\}$, y_i denotes the predicted keyword list for tr_i, and g_i denotes the ground-truth keyword list. Then, the precision and recall can be computed based on the following equations.

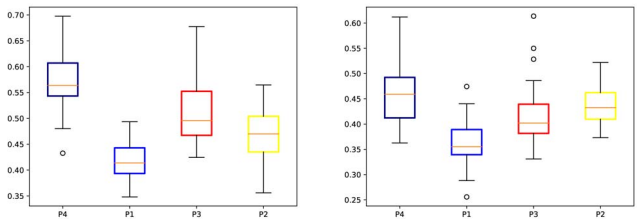
$$P = \frac{1}{|TR'|}\sum_{i=1}^{|TR'|}\frac{|y_i \bigcap g_i|}{y_i}; R = \frac{1}{|TR'|}\sum_{i=1}^{|TR'|}\frac{|y_i \bigcap g_i|}{g_i} \quad (1)$$

In our experiment, we employ the trigram and apply the average linkage that defines the distance as the average distance between each instance in one cluster to every instance in the other cluster. Further, we set the distance threshold value $\alpha = 0.1$, the lowest frequency threshold $X = 2$ and the number of nearest neighbors $K = 3$. We conduct the experiment 30 times and present the experiment results in the Figures 3a and 3b. These two boxplots show that, for all four projects, we obtain an average precision value more than 0.41 and an average recall value more than 0.35.

When analyzing these data, an important phenomenon should be taken into consideration, there is a very low possibility that people use the same words to refer to the same concepts. This phenomenon has been recognized and well-studied by both the linguistic and natural language processing researchers [9], [15]. For description generation techniques, experiment result of the word-comparison based metrics may vary given the ground truth that comes from different people [19]. Thus, even though the average precision and recall values are not so high in comparison with the reported results of the classic classification-based techniques, in which these values are commonly higher than 0.8, our technique that can automatically generate 40% keywords of the manually written descriptions is promising in improving the comprehensibility of crowdsourced test reports.

V. RELATED WORK

Compared with natural language processing (NLP), which aim at analyzing textual information and have been widely applied to assist various software engineering tasks, image analysis and understanding techniques are rarely investigated and studied in software engineering domain. Cai *et al.* propose the VIPS algorithm [1], which segments a web page's screenshot into visual blocks to infer the hierarchy from the visual layout, rather than from the DOM. Choudhary *et al.* propose a tool called WEBDIFF to compare the page's appearance



(a) Precision (b) Recall

Fig. 3: Experiment Results (averaged over 30 runs)

on screenshots of the web pages to identify cross-browser issues automatically [3]. In [14], Michail *et al.* proposed a static approach, GUISearch, to guide search and browsing of its source code by using the GUI of an application. They further proposed a dynamic approach to obtain an explicit mapping from high-level actions to low-level implementation by identifying execution triggered by user actions and visually describing actions from a fragment of the application displayed [2]. While all of these works employ image-understanding technique to detect software bugs, our approach aims at improving the understandability of screenshots of test reports. Feng *et al.* [7] proposed an image-understanding-based prioritization technique to provide an efficient diagnosis order for developers to process test reports. However, even though this method enables the developers to identify bugs earlier, it does not assist developers in improving the efficiency of diagnosing individual test reports that contain only short-text descriptions and screenshots.

VI. CONCLUSION

In this paper, we propose a framework to analyze the content of screenshots to generate descriptions for them to further assist software developer comprehension of crowdsourced test reports. To our knowledge, this is the first work to propose using image-understanding and n-gram techniques to assist improving the efficiency of understanding crowdsourced test reports.We present the preliminary experiment result based on four real industrial mobile crowdsourced projects. The experiement results show that our technique can reach a relative high precision and recall regarding generating descriptions for the images.

To further improve the performance and make this technique mature, we will refine the language model and design the novel ones. Additionally, further user studies and empirical experiments are therefore necessary to investigate the improvement for developers in comprehending the crowdsourced test reports.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation under award CAREER CCF-1350837 and National Natural Science Foundation of China (61772263, 61772014, 61572375).

REFERENCES

[1] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma. Vips: a visionbased page segmentation algorithm. Technical report, Microsoft technical report, MSR-TR-2003-79, 2003.

[2] K. Chan, Z. C. L. Liang, and A. Michail. Design recovery of interactive graphical applications. In *Proceedings of the 25th international conference on Software engineering*, pages 114–124. IEEE Computer Society, 2003.

[3] S. R. Choudhary, H. Versee, and A. Orso. A cross-browser web application testing tool. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–6. IEEE, 2010.

[4] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel. Rebucket: a method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1084–1093. IEEE Press, 2012.

[5] Y. Feng and Z. Chen. Multi-label software behavior learning. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1305–1308. IEEE Press, 2012.

[6] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu. Test report prioritization to assist crowdsourced testing. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. New York: ACM*, pages 225–236, 2015.

[7] Y. Feng, J. A. Jones, Z. Chen, and C. Fang. Multi-objective test report prioritization using image understanding. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 202–213. ACM, 2016.

[8] Y. Feng, Q. Liu, M. Dou, J. Liu, and Z. Chen. Mubug: a mobile service for rapid bug tracking. *Science China Information Sciences*, 59(1):1–5, 2016.

[9] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.

[10] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 52–61. IEEE, 2008.

[11] T. D. LaToza, M. Chen, L. Jiang, M. Zhao, and A. Van Der Hoek. Borrowing from the crowd: A study of recombination in software design competitions. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 551–562. IEEE Press, 2015.

[12] T. D. LaToza, W. B. Towne, C. M. Adriano, and A. Van Der Hoek. Microtask programming: Building software with a crowd. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 43–54. ACM, 2014.

[13] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[14] A. Michail. Browsing and searching source code of applications written using a gui framework. In *Proceedings of the 24th International Conference on Software Engineering*, pages 327–337. ACM, 2002.

[15] V. Ordonez, G. Kulkarni, and T. L. Berg. Im2text: Describing images using 1 million captioned photographs. In *Advances in Neural Information Processing Systems*, pages 1143–1151, 2011.

[16] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 465–475. IEEE, 2003.

[17] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Natural language processing and text mining*, pages 9–28. Springer, 2007.

[18] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 499–510. IEEE, 2007.

[19] H. Schütze. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, 2008.

[20] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 253–262. IEEE Computer Society, 2011.

[21] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 45–54. ACM, 2010.

[22] Y. Tian, C. Sun, and D. Lo. Improved duplicate bug report identification. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 385–390. IEEE, 2012.

[23] J. Wu, Z. Cui, V. S. Sheng, P. Zhao, D. Su, and S. Gong. A comparative study of sift and its variants. *Measurement Science Review*, 13(3):122–131, 2013.

[24] X. Xia, Y. Feng, D. Lo, Z. Chen, and X. Wang. Towards more accurate multi-label software behavior learning. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 134–143. IEEE, 2014.

[25] K. Zhang, H. Xu, J. Tang, and J. Li. Keyword extraction using support vector machine. In *Advances in Web-Age Information Management*, pages 85–96. Springer, 2006.

[26] T. Zhang, J. Chen, X. Luo, and T. Li. Bug reports for desktop software and mobile apps in github: What is the difference? *IEEE Software*, 2017.