UIChecker:An Automatic Detection Platform for Android GUI Errors

Meichen Ji
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, Shanghai 200240, China
jimc5512@sjtu.edu.cn

Abstract—**At** present, And..oid automated GUI testing has been widely used in mobile application testing. Automated GUI test input generation technology and tools are hot topics for practitione..s, but errors in some test sc..eenshots generated by automated test Input tools still need to be reviewed manuaUy. In this paper, **we** creatively proposed an automatic detection platfo..m for GUI errOI·s, detecting the GUI errOI·s of mobile related and image-related widget error classification model th..ough machine learning, which detects the error of widgets. On all experimental App test sets, the **accuracy** of the text-related widget error classification model reached an average of 98.06%, and the **accuracy** of image-related widgets error classification model achieved an average of 95.44%, which greatly reduced the time cost of reviewing GUI errors manually. In addition, **we** analyze the relative positional relationship between the widgets, and use the Wilson score **sorting** algorithm to analyze the symbiosis and interdependence between the widgets, and finally generate the **assertion** tables, thus mo..e complex GUI erro..s can be detected.

Keywords-automated GUI testing; automated traversal tool; machine learning; assertions; GUI error detection

I. INTRODUCTION (HEADING 1)

Due to the current uneven level of technology for mobile application developers, coupled with the short development cycle of mobile applications and tight deadlines, the app's robustness in the application market is seriously insufficient. This will pose a challenge for mobile application testers. The most widely used method for mobile application testing is Gill testing [3]. This method analyzes the test results of the user interface to determine whether the program has errors, such as clicking a button on the Gill, the test will check if it is running correctly by the health of the application [7]. In the traditional Gill test method, manual Gill test is generally used, but the labor cost of manual Gill test is large and time-consuming. Some errors are not necessarily discovered by testers in time, so in order to carry out low-cost effective testing, the technology and tools for mobile application automation Gill testing have been extensively studied. Currently Android has occupied most of the market share in the mobile operating system, so the Android App automated Gill testing has attracted more attention from researchers and practitioners [9,10,12]. At present, the automated Gill testing input generation technology has been extensively explored, but the screenshots generated by the test input generation tool still need to be manually reviewed, so we have innovatively proposed an intelligent Gill error detection method based on machine learning and statistics to help testers review automatically, and also, we develop an automated Gill error detection platform UlChecker, which reducing the time cost of manual review. We mainly use the screenshots captured by the automated traversal tool and the information in the corresponding layout xml file to establish text-related and image-related widgets (hereinafter, "related widgets") error classification model through machine learning, in order to find the errors of the related widgets in the application. At the same time, we give some assertions by analyzing the relationship between the brother widget pairs and father-son widget pairs, mcludmg. relative positional relationship, symbiotic relationship and mterdependence relationship, which help the testers to analyze some complex Gill errors.

. The main structure of this paper is as follows: Chapter 2 introduces design architecture of illChecker, Chapter 3 descnbes the specific implementation. Chapter 4 describes the experimental methods and results. Chapter 5 describes the conclusion.

II. DESIGN

A. Design Architecture

. This article divides the GUI error types of Android App mto two categones. The first type of error can be directly found from the display status of a single widget on the screen, such as garbled text, image loading failure, etc. The second type of error can be discovered from the display status between two widgets on the screen, for example, the text and images may overlap each other. There is another error related to the context of the running application, such as the disappearance of some title bar in the application, we also classify it as a second type of error. We use the machine learning method to establish the classification model of text and image-related widgets for the first type of error, and uses the mathematical statistics method for the second type of error. By extracting the attribute information of the brothers and father-son widget pairs, the assertion table is generated, which define the relative positional relationship, symbiotic relationship and interdependence relationship, and help testers judge the second type of error. The following figure is the design architecture ofUIChecker:

Supported by the National Natural Science Foundation of China

978-1-5386-6565-7iI8/S31.00 ©2018 IEEE

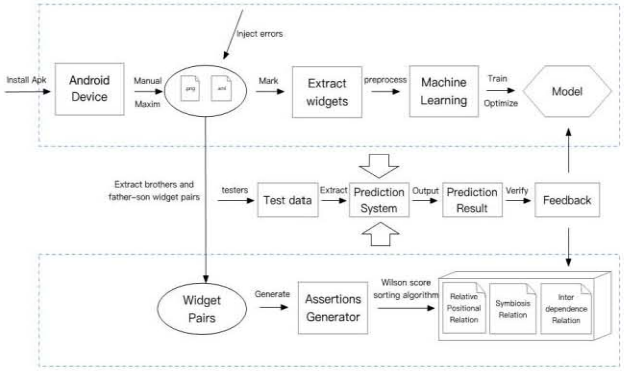

Figure 1. Design architecture diagram of UIChecker

The UIChecker design architecture is divided into three parts.

- Establish a single widget error classification model. The classification model includes a text-related error classification model and a picture-related error classification model. Text-related classification model include a common error classification model and a proprietary model for a particular App. However, as the image-related error classification model has a large relevance to the App content, only the error classification model for a specific App is established.
- Generate assertion tables. For the pair of widgets with brothers and father-son relationship, we extract the relative positional relationship, symbiotic relationship and interdependence relationship between the widget pairs, finally generate the assertion tables about the widget pairs.
- Establish a prediction system. The tester re-acquires the automated test screenshots of a particular App and uses the model and the assertion tables to predict the new test data, and generates a prediction report. For the added function modules in the new App, if the prediction system which is based on the previous model and assertion tables will produce some erroneous predictions. The system provides a feedback entry for the error predictions. We will add the correct related classification model and correct assertion tables to the error classification model generation module and the assertion table generation module.

III. IMPLEMENTATION

A. *Data collection*

We first need to install the Apk file on the Android phone, and then collect the screenshots of the App when it runs on the phone. The data collection is mainly divided into two steps:

- Automated collection. Automated GUI testing is performed by the automated traversal tool Maxim [13] based on Android Monkey, Maxim improves the coverage of the test through a depth-first traversal strategy. It calls the UIAutomator2 API provided by Google can simulate five action events per second and dump the current UI screenshot and its corresponding layout file. The layout file is a standard xml file that stores the details of each widget node in the screenshot. By setting Maxim's running time, you can get a large number of the screenshots and layout files of the app at runtime. After removing the redundancy of the collecting data, we will get the first part of the original data set.
- Manual collection. The UI screenshot of the App at runtime and its corresponding layout file are collected through manual GUI testing. Since the App does not have enough errors collected by Maxim, in order to build the error classification model, we need to manually produce some error data, such as simulating cutting off the network and so on, then use UIAutomator2 to capture the screenshot and the layout file of the running app more quickly. In this way, we get the second part of the original data set.

B. *Error classification model for single widget*

We surveyed the App GUI errors exposed in many test community sites, such as TesterHome, the largest test community site in China, collecting some error widget types about text and image. We use the machine learning method to establish the error classification model of the related widgets, and apply it to automatically detect the new GUI test data.

1) *Error classification model for text-related widgets*

- We randomly inject the text errors that may occur in the survey into the nodes of the text-related widgets in the layout file, and increases the error classification data in the training set.
- We will extract the widget attribute information that is not directly displayed in the layout file as the features, mainly from the attribute values of text and content-desc, such as whether the user interface returns “null”, whether there is unresolved Unicode code, html characters, and other garbled characters, etc. In addition, the attribute features directly displayed in the layout file, such as resource-id, clickable, and focus, will help us to increase the accuracy of classifying specific app text.
- For a widget attribute value such as resource-id, we normalize it to a numeric form. For a Boolean variable, we normalize it to a value of 0 and 1.
- We choose the decision tree algorithm. Firstly, it does not require much training data. Secondly, the process of training is faster. The most important thing is that the model can be visualized, which can help us to optimize the model quickly.
- We use the 10-fold cross-validation method to verify the generated model, and improve the classification of the error classification model by continuously adjusting the parameters in the decision tree model, including the maximum depth of the decision tree, the maximum number of features, and the maximum number of minimum leaf nodes.

- We ven fy the data on the test set to see If the model is too fitted In addition, we need to persist the opbmized error classification model of the text-related widgets.

2) Error classification model for image-related widgets

- For the screenshct file in the first and second part of the original dataset, we use the bounds of the widget in the layout file to intercept each urage widget in the screenshot, and then manually mark the correctness of the captured snage
- We use the erage recognition algorithm to extract the main COCIS presented in the picture, each color selects Its cccrespondmg R, 0, B attributes as the cheacte-encs of the picture In addition, we also extract the length and width and area of the picture widget as important features
- The R, 0, and B properties of the rram color of the image are represented by 0 to 255, and the length, Width, and area are calculated based on the attribute values in the widget node
- We still choose the decision tree algorithm. Because sc:rne of the extracted features can accurately classify the wrong picture widget, the decision tree algorithm is the best choice
- The process of pararreter-tm ing, model-verification and persistent stoage are similar to the process of creating text-related *ercr* classification rnodel

C. *Analysis for widget pairs*

UIChed:er ccntaersthe assernco geneauon module, where we extract all attribute information of the widget pairs on the ccerea part of the first part of the original data set and create the essence tables through the mathematical stausccs method, that describe the relative positional relationship and symblOhc relationship and interdependence relationship between the widgets to solve the secoodtype of e-ce

1J Relative positional relationship

There are many widget pairs in the App, and we select the two most important posaional relationship widget pairs

- The father-son reletic nship widget pairs We mainly focus on the strong relatrce relationship of the duld widget relative to the psree widget, thai is, whether the child widget is at the most top, left, bottom or right of the boundary of the peect widget.
- Brother relaticosbip widget pairs. We mainly cce cem abc:ot the relative positional relationship between the two child widgets under the parent Widget, that is, whether cqe widget is up, left, down, right, aligned *cr* overlapped in sc:rne direction relative to another sibling widget.

2) Symbiotic and interdependence relationship

A symbrctrc relationship means that two widgets either exist at the same time or do net exist at the same time An interdependence relationship means that ere widget can enly exist when another one widget exists We use the Wilson score sotmg algonthm to determine the symbiotic and jnterdependence relationship The equation (1) is shown below

$$\begin{cases} n = u + v \\ p = \frac{u}{n} \\ S = (p + \frac{z_\alpha}{2n} - \frac{a_\alpha}{2n}\sqrt{4n(1 \quad p)p + z_\alpha^2}) \\ \qquad (1 + \frac{z_\alpha^2}{n}) \end{cases} \tag{1}$$

For widgets with perent-duld relationship and Slblllg relationship <W1, W2>, we calculate three values (1) The number of occurrences that W1 and WZ appear simultaneously wtnch is represeued by wtn WZ. (2) The number- of ccce-ences thai widget W2 does net appear while widget WI appears which IS represented by wI n ..,w2, 0) The number of cccerences that the widget WI does net appear and the widget W2 appears alene which is represented by w 2 0 - ovt. $z_{..}$ is a normal distribution of the quantile, which generally takes 2, Next, we use the Wilson sccee soetng algcrithm in two stages

- Determine the syrm iotic relationship of the widget pair In the above Icerrula u = w1 0 w2 , v = (w10 ..,w2)+(w2 0 ..,w1), p represents the probability that the widget W 1, W2 appear together, S represents the final score of the symbietic relatrceshrp between the widgets Wl and WZ Because S performs different in different Apps, we establish the symbiotic relationship assertion table by empirically setting the minimum threshold of S
- peemoe the interdependence relationship of the widget pairs Firstly, we describe the dependency of the widget WI reletwe to !he widget WZ In the above formula u=wt 0 WZ, v = WI' 0 ..,w2, P represents the probability that Wt can appear only if W2 appears, S represents the final sce-e of the dependency of the widget WI relative to widget WZ By empirically setting the minimum threshold of S, a dependency essence table of W1 to W2 is gene-eted, Secondly, the dependency of the widget W2 reletrse to the wrdget W1 15 described, u = w1 ∩ w2, l· =w2n ..,w1, p represents the probability that W2 can appear only If W1 appears, S represents the final sccee of the dependency of Widget W2 relative to WI, and also we enpmcally set !he minirrum threshold of S, a dependency assertion table of WZ to WI is generated

IV, EXPERIMENTATION

A. *Environment Setup*

We have built a test environment on desktop PC and Android device. The detailed experimental envircnment is described in Table

TABLEL ENVIRONMENT OF TIIE DESKTOP PC AND TIIE ANDROID DEVICE

Desktop PC	Operating System	MacOS High Sierra 10.13.4
	CPU	2.7GHz Intel Core i5
	Memory	8GB 1867 MHz DDRJ
Andro id device	Device name	Huawei honor 10
	Andro id System	Android8.1.0
	processor	HiSilicon Kirin 970
	Memory	6GB
	Screen pixel	2280*1080

UIChecker IS a GUI testmg platform based on black-box. It does not need to obtain the detailed information of the widgets on the screenshot through the source code. It can apply to both open source apps and closed source apps. The experiment selects 5 closed source apps in the third-party application market, namely: Taobao, Sohu news, Today's headlines, JD Finance, Zhihu,

B. Impirical Results

The experimental results are introduced in two aspects:

- The accuracy of the related widget error classification model for classifying the error widget on the training set and the test set;
- The number of widget pairs extracted from the original data set and the detail information of assertion tables.

1) Detection oftext-related error widgets

By analyzing the data intercepted by the automated traversal tool Maxim, we cannot find too many text-related widgets errors, in order to increase the error classification data in the training data, we manually inject 30% widget errors of the total amount of text-related widgets. The probability of each type of error injection is equaL Table below describes the prediction classification aimed at the injected text error by using the text-related widget error classification model on the training set and test set:

TABLE II. PREDICTION RESULT OF TIIE TEXT-RELATED WIDGET ERROR CLASSIFICATION MODEL ON TIIE TRAINING SET AND TEST SET

App name	training set	test set			
	Accuracy	Accuracy	Recall	Precision	Fl-score
Taobao	98.31%	98.31%	92.02%	100%	95.84%
Sohu news	98.49%	98.79%	96.81%	100%	98.38%
Today's headlines	98.59%	97.98%	92.31%	100%	96.00%
JD finance	99.61%	99.90%	99.86%	100%	99.93%
Zhihu	96.97%	95.34%	89.20%	100%	94.29%

It can be seen from the figure that each App's error classification model has a higher classification accuracy on the training set. The average prediction accuracy through the 1.0-fold cross-validation is basically over 98%. The prediction results on the test set are also close to the prediction results on the training set, and the accuracy is generally higher. On the JD Finance test set, the accuracy of the error widget classification is close to L00%. On the Zhihu test set, the accuracy of the error widget classification is only 95.34%, because the cardinality problem of the first part of the original data set, the accuracy rate has not reached a high leveL In addition, the precision of the test set has reached L00%, except for JD Finance, Sohu News. The recall rate of other App is not very high, indicating that some errors have not been fuLly predicted, but through F L-score, the recall rate and precision are considered comprehensively, and the predicted results are generally good.

2) Detection ofimage-related error widgets

Table shows the prediction resuLts of the image-related widget error classification model on the training set and test set:

TABLE III. PREDICTION RESULT OF TIIE IMAGE-RELATED WIDGET ERROR CLASSIFICATION MODEL ON TIIE TRAINING SET AND TEST SET

App name	Training set	Test set			
	Accuracy	Accuracy	Recall	Precision	Fl-score
Taobao	92.09%	93.03%	82.69%	75.88%	79.14 %
Sohu news	97.53%	98.41%	86.67%	83.87%	85.25 %
Today's headlines	95.36%	97.52%	89.39%	89.39%	89.39%
JD finance	93.19%	94.46%	81.25%	76.47%	78.79%
Zhihu	94.25%	93.79%	66.67%	82.35%	73.69%

It can be seen from the table that the classification accuracy of the model on the training set is basically above 92%, most of the App's prediction results on the test set are better than those on the training set. In addition, the prediction classification accuracy on the test set is above 93%. However, the recall rate and precision do not reach at 90% in all the closed source apps. On the one hand, the error dataset collected is not enough, on the other hand, it depends on the limitation of the decision tree classification modeL If the unloaded or errored image widgets contained in the App screenshot have very similar color to some correct image widgets, and the size of the image widgets is also similar to the correct widget size, the error image-related widget may not correctly classified.

3) error detection for widget pairs

By extracting the details of the two widgets in the brothers and father-son relationship widget pairs, we can generate relative position, symbiotic, interdependency relationship assertion table of the widget pairs. Table below is the quantity statistics of assertions in the three kinds of assertion tables.

TABLE IV. NUMBER OF RELATIVE POSITION RELATION, SYMBIOSIS AND INTERDEPENDENCE ASSERTIONS EXTRACTED FROM SIBLING AND FATIIER-SON WIDGET PAIRS INFORMATION

App name	Relative Positional relationship		Symbiotic relationship		Interdependency relationship	
	bros	Father so	bros	Father so	bros	Father so
Taobao	20732	65541	707	554	141	269
Sohu news	21401	33786	223	244	113	70
Today's headlines	5499	22896	180	126	192	156
JD finance	14800	36109	602	205	123	35
Zhihu	33928	20605	4593	174	175	97

For the relative positional relationship assertion, we mainly analyze the relative positional relationship between each widget and other widgets based on mathematical statistical methods. If the relative position relationship between a certain widget and other widgets in the newly intercepted GUI test screenshot cannot be found in the assertion tables, then we highlight the widget, prompting the tester to have potential error widgets in the screenshot. In addition, if a widget in the newly captured GUI test screenshot can be queried in the symbiotic relationship assertion table, but does not appear with another widget, we will highlight the widget. Similarly, if a widget is queried in an interdependency relationship assertion table, while does not depend on another widget and appears separately, the widget will also be highlighted. At the same time, we also found that many assertions missed in the symbiotic relationship assertion table and the interdependency relationship assertion table, however, by receiving feedback on the actual prediction results, we can constantly improve the assertion tables.

CONCLUSION

This paper creatively proposes a GUI error detection method based on machine learning, and introduces an automatic GUI errors detection platform UIChecker, which help testers detect GUI errors intelligently. For the error of a single widget, we have separately created the text-related and image-related error classification model. For the error between the widget pairs, we extract the relation between widget pairs, including relative positional, symbiotic and interdependence relationship, and generate three assertion tables. The experimental results show that the accuracy of recognition of injected errors is very high, and the assertion tables also help us find more complex GUI errors.

ACKNOWLEDGMENT (HEADING 5)

I would like to thank Zhenyu Cai and developer of Maxim for their support.

REFERENCES

In terms of test generation technology, Google officially provides Android Monkey, which can simulate the user behavior and generate clicks, touches, swipe gestures and some system-level pseudo-random event streams, but it is completely random, some domestic industry researchers make some improvements to it, such as the automated GUI test generation tool Maxim, which uses a depth-first traversal algorithm to improve the completely random of Monkey.

In addition, the model-based GUI test generation technology is also a hot spot for researchers [1,3,4,5,6,8]. Young-Min Baek [4] et al. proposed a multi-level GUI comparison standard GUICC, which provides one more abstract level selection for GUI model generation, which is more efficient than activity-based GUI model generation. Peng Liu [1] et al. provide an approach based on deep learning that automatically generates the most appropriate text as input based on the context of the App.

In the mobile application automation GUI error detection, there are many tools like AppDoctor [11] that are developed to detect App errors, but all the errors need to be manually reviewed. GTV designed by Boyang Li [2] et al., which uses computer vision, can automatically detect, classify and generate reports on Gill errors in the app, but due to the limitations of CV, the tool cannot detect the error correctly if there is overlap of widget boundaries, while UIChecker can still detect these errors.

[1] Peng Liu , Xiangyu Zhang, Marco Pistoia, Yunhui Zheng, Manoel Marques and Lingfei Zeng. Automatic text input generation for mobile testing. In Proceedings of the 39th International Conference on Software Engineering. 2017. 643-653

[2] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, Denys Poshyvanyk Kevin Moran. Automated reporting of GUI design violations for mobile apps. In Proceedings of the 40th International Conference on Software Engineering. 2018.1 65-175

[3] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon. Using GUI ripping for automated testing of android Applications. In Proceedings of the 27th IEEE!ACM International Conference on Automated Software Engineering. 2012. 258-261.

[4] Young-Min Baek, Doo-Hwan Bae. Automated model-based Android GUI testing using multi-level GUI comparison criteria. In Proceedings of the 31st IEEE!ACM International Conference on Automated Software Engineering. 2016. 238-249.

[5] Q. Xie and A. M. Memon. Using a pilot study to derive a gui model for automated testing. ACM Trans. on Software Engineering and Methodology, 2018 18(2):7:1- 7:35.

[6] D. Amalfitano, A. R. Fasolino, and P. Tramontana. A GUI crawling-based technique for android mobile application testing. In Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshops. 2011. 252-261.

[7] G. Bae, G. Rothermel, and D.-H. Bae. Comparing model-based and dynamic event-extraction based GUI testing techniques: An empirical study. Journal of Systems and Software, 2014,97:15-46.

[8] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. D. Carmine, and G. Imparato. A toolset for GUI testing of android applications. In Proceedings of the 28th IEEE International Conference on Software Maintenance. 2012. 650-653.

[9] S. R. Choudhary, A. Gorla, and A. Orso. Automated test input generation for android: are we there yet? (e). In Proceedings of the 30th IEEE!ACM International Conference on Automated Software Engineering. 2015. 429-440.

[10] GoogleDevelopers. UI!Application exerciser monkey (http://developer.android.com/intl/ko/tools/help!monkey.htrnl)

[11] G. Hu, X. Yuan, Y. Tang, and J. Yang. Efficiently, effectively detecting mobile app bugs with appdoctor. In Proceedings of the 9th European Conference on Computer Systems. 2014

[12] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan. Puma: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services. 2014. 204-217.

[13] https:l!github.com!zhangzha04444/Maxim