© Telkom University

INTERNATIONAL JOURNAL OF INNOVATION IN ENTERPRISE SYSTEM

Journal homepage: https://ijies.sie.telkomuniversity.ac.id/index.php/IJIES/index
e-ISSN: 2580-3050

Automated Event-Triggered GUI Testing and Bug Reproduction for Non-Android Applications

Anit Thomas M[1]*, Chinchu Krishna S[2]

[1] PG Student
Rajagiri School of Engineering and Technology, Kochi, 682039, INDIA

[2] Assistant Professor
Rajagiri School of Engineering and Technology, Kochi, 682039, INDIA

*anitthomasm@gmail.com

ARTICLE INFO

Article history:
Received 14 October 2020
Accepted 10 March 2021
Published 31 July 2021

Keywords:
Android Applications; Automated Bug Reproduction; Bug Reports; LogFile; Use case.

ABSTRACT

Today's mobile gadgets are seamlessly incorporating innovative features demanded by the users. Most often, the applications contain bugs or functionality issues reported by the customers. The developers are responsible for reproducing such reported bugs, which are written in natural languages. Reproducing bugs from bug reports make the bug resolution inefficient. Nowadays, various methods are adopted to reproduce crash reports for android applications. But bug reproduction for non-android applications is still a challenging task. This paper proposes a novel approach that is capable of doing bug reproduction from bug reports to help the developers to solve the functionality issues of non-android applications in an automated manner. This approach uses a UI tester called the GUI engine, which is an excel sheet. The use cases which are to be tested can be filled in the GUI engine. All the use cases are to be filled based on a particular syntax. For that purpose, the developer can make use of a set of yaml files containing all the GUI information for all the screens of the application under test. The use cases are executed in the GUI engine and conclusions are made based on the test result. The GUI engine displays two colors green and red showing the working and failing of GUI components of the application under test. So that the developer can easily identify the failing components and take actions accordingly. The test has been done with more than 1000 test cases for one region and the result shows that almost all GUI components work for this method except for animation. It has tested for software developed for US, Europe, and German regions. Also, the proposed method is found to be much faster and efficient than the existing as well as manual testing methods.

1. Introduction

The fourth industrial revolution known as Industry-4.0 is predominantly proceeding through the seamless semantic data connectivity and applications of Artificial intelligence. The technology-driven Industry-4.0 is supported by mobile gadgets and their subsequent applications to empower society to thrive beyond the limitations of time and space. The world became practically a single village in terms of the knowledge dissemination and service networks fostered through social media applications. The vast majority of the population around the globe depends on such social media mobile

applications to ease their life activities [1],[2]. Multitudes of applications are available in the virtual marketplace to suit the demands of the customers, which in turn naturally hiked the mobile gadgets market also.

The spread of handheld gadgets initiated the ancillary development of the Omni platform applications, and an array of multiple apps is available with almost similar features to select. The user has the freedom to opt for those reliable applications that are bug-free and having useful functionalities. The decisive features of the applications were rated in the competitive virtual market, and those with high utility with no performance issues were most sought after. These user preferences naturally cautioned the developers to identify and correct errors as early as possible on a real-time basis [3].

Mobile applications are incorporating multiple functionalities to cater to the demands of the users. These applications, which seamlessly incorporating innovative features, are readily demanded by the users. As developers add more features to the applications, users are readily updating those and thrive for further experience with them [3]. One of the most critical features of every application is its Graphical User Interface known as GUI that helps the users to interact with the applications. GUI is composed of buttons, texts, status bars, sliding bars, and icons. After the development of each software or application, it must undergo a multileveled thorough testing process. It is quite expected that software or an application might have specific errors after its development. All the preidentified errors will be solved by the developers immediately on the testing process [4].

However, bugs are inevitably expected in any application even after its deployment. So, the bug tracking and rectification system became essential for the smooth functioning of the application. The inbuilt bug tracking systems (as Bugzilla, Apple Crash Reporter, Google Code Tracker, Github Issue Tracker etc.) [5], [6], [7] of the applications help testers and users to report the bugs they identified while using a particular software or application. All the errors intimated by the users will be given to the developer team for reproduction and resolution. Bug reports help the developers to know the nature of the unexpected bugs [8]. However, bug reports written in natural languages with four or five-step descriptions may be often incomplete, and it is rather difficult for the developers to reproduce those bugs reported by the user [9], [10]. Also, actual reproduction can be challenging as the applications can have complex event-related and GUI-related behaviors, and many GUI-based actions are also required for the bug reproduction process [3]. Manually reproducing all such bugs will consume a significant amount of time and is non-practical as per the nature of the system [11]. This necessitated the application of automated systems for bug recreation from bug reports.

The existing methods either deals with android applications or improve the quality of bug reports. But there exists no method that deals with UI testing as well as bug reproduction for non-android applications. Also, the existing methods are not capable of testing if the number of GUI components on the screen increases. This paper is intended to unveil a novel approach to help the developers to reproduce the functional issues from bug reports that are written in natural languages for non-android applications in an automated way. Also, the proposed approach is capable of testing all types of GUI components even if the amount of components on a screen increases.

2. Theoretical Framework

Nowadays several methods exist for the bug reproduction of android applications:

ReCDroid [3] is one such approach that can automatically reproduce crashes from bug reports. This method uses a combination of natural language processing (NLP) and GUI exploration to extract event sequences that lead to the reproduction of reported crashes. The main limitations of this approach are, it is mainly dealing with the reproduction of android applications crashes and it is not dealing with any other functionality issue reproduction.

CRASHSCOPE [12] is another method that deals with the reproductions of android application crashes. This method was also invented to help developers to reproduce android application crashes in an automated manner. It uses Abstract Syntax Tree (AST) based analysis to extract the GUI events. The main limitations of this method are, it is not dealing with swipe gestures also this tool relates to only the window detection in Android applications and is mainly concentrating on crash reproduction of android applications.

A similar purposed method is a MoTiF [13] that is also concentrated on supporting app developers in automatically reproducing context-sensitive crashes for android applications. It analyses the patterns in crash reports and identifies the shortest sequence of events that are capable of reproducing a crash and those sequences are turned into a test suite. MoTiF learns the contexts where the crash test suites truly reproduce the observed. This test suite is validated and MoTiF notifies the app developers. The limitations of this approach are, this method is also concentrating on android crash reproduction and is not dealing with any non-android applications or solve the other functionality issues of the applications also it is not dealing with how to deal with incomplete bugs reports.

CRASHDROID [14] is another approach that is capable of translating the incomplete steps from bug reports into steps that are able to reproduce android application crashes in an automated manner. It has a database that contains links between scenarios, descriptions, and kernel input events and reproducible/replayable bug reports for target crashes so that CRASHDROID can link scenarios, natural language descriptions, replay scripts, and app execution profiles and automatically transforms GUI events from incoming crash reports into sets of expressive bug reports and replay scripts

© Copyright by Telkom University

for a software maintainer. Bug reproduction can be inefficient due to the unexpected behaviors in the bug reports reported by the customers.

DeMIBuD [15 is an automated approach that detects missing information from bug reports such as expected behavior and steps to reproduce to solve the ambiguities in the bug reports. This approach developed patterns that describe Observed behavior, expected behavior, and steps to reproduce by analyzing a subset of bug reports. DeMIBuD has been developed in three versions based on regular expressions, heuristics and Natural Language Processing (NLP), and Machine Learning (ML). DeMIBuD can alert customers while submitting bug reports in order to assess the quality of the report, so that they can contact developers to improve their reports to enhance bug reproduction.

A similar method to improve the quality of bug reporting is FUSION [12]. It helps users of mobile applications to auto-complete the reproduction steps in the bug reports. Static and dynamic analysis is used to extract program artifacts and FUSION links this with the user-provided information. The main limitations of this method include, it is not dealing with all types of gestures and is only dealing with tap or click gestures. Also, it is capable of auto-completing those bug reports that can be uncovered using only GUI-Gestures such as tap, long-touch, swipe, and type.

The proposed method differs from prior works that deal with bug reproduction because most of the existing methods were solely concentrated on crash reproduction for android applications [16]. And they were concentrated predominantly on improving the quality of bug reports as well [2], [15], [8], [17], [18], [19]. None of them have developed an effective bug reproduction system that could retrieve necessary information from the available bug reports. This inadequacy of existing methods to deal with the bug reproduction of non-android applications is addressed in this novel approach. It is a robust method that could extract all the information from the existing bug reports and all these bug reports are taken into consideration for bug reproduction regardless of their quality. This method is considered a powerful tool as it is capable of reproducing bugs regardless of the count of GUI components in any application.

3. Methodology

From the study about the various bug reproduction methods, we came to the following conclusions:

- The current methods are solely addressing to reproduce crashes of android applications only and are not intended in solving the functionality issues of non-android applications [3], [8], [13], [14], [15], [20], [16], [21].
- The bug reports that are written in natural languages may be incomplete, or they may not convey the actual issues faced by the customers so, the existing bug reproduction process became eventually inefficient [3], [9], [10], [22].
- Existing methods are not efficient enough to deal with the bug reproduction process where the quantum of GUI components are exponentially increased and the screens which consume a large amount of time on manual bug reproduction [23], [24].
- It is observed that the same functionality issues were reported by the customers multiple times. Manually reproducing those same issues multiple times during the testing process may be detrimental to the entire project.

This paper introduces a novel and effective solution to confront these critical lacunas of the existing bug reproduction systems. It is capable of addressing all the issues mentioned above and compact enough to help developers to reproduce all reported bugs that are written in natural languages for any non-android application in an automated manner.

The architecture of the bug reproduction system is shown in Fig. 1. It consists of two major phases – the bug report analysis phase and the bug reproduction phase.

Phase 1: The bug report analysis phase

In this phase, the bug reports from the customers that are written in natural languages are tracked by the bug tracking system. Every application has a large number of screens and each such screen contain many clusters of GUI components such as buttons, texts, icons, status bars, etc. The bug reports have a five-step description of how to reproduce the bugs. These steps are executed on the application under test and the logs are stored.

Phase 2: Bug reproduction phase

In the second phase, the log file which contains all the user actions such as hard key events, soft key touch coordinates, view names, etc. is used. The logged events are fed to a sequencer that generates input commands, generate view check and optimize the sequences. The output of the sequencer is given to a repeater module that sets the required pre-conditions and activates the traces and is capable of doing repeated bug reproduction process any number of times. Fig 1 shows the overview of the bug reproduction framework.

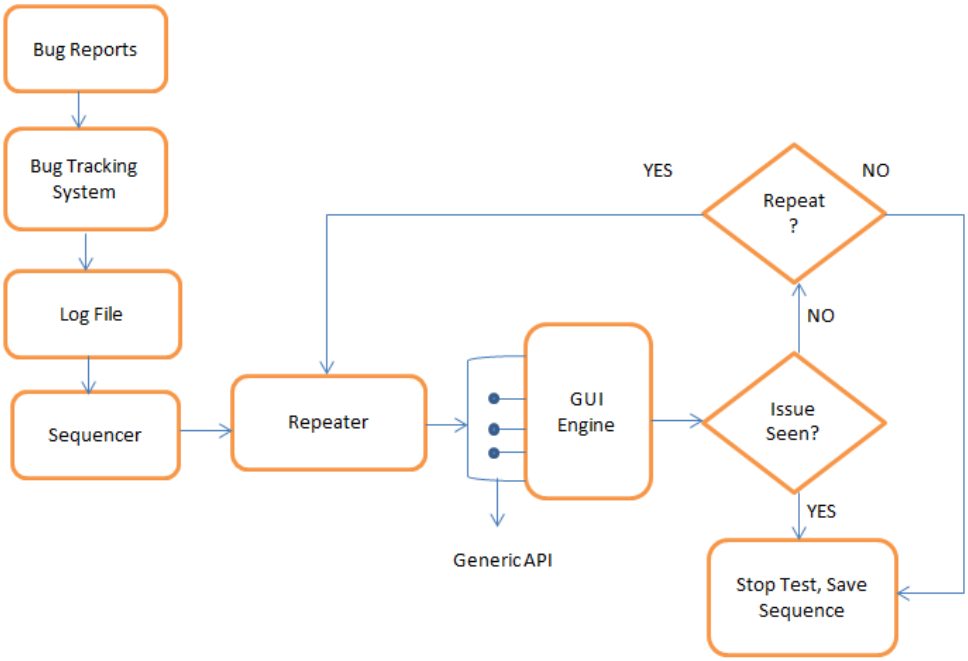

Fig. 1 - Overview of Bug Reproduction Framework

The bug reproduction system consists of the following modules:

3.1 Bug Report

Every application has an inbuilt bug reporting system that allows reporting, document, store, manage, assign, close, and achieve the reports. During the execution of any application, the person who observes any bug or crashes can report that bug to people in charge of fixing the bugs or failures through the bug reporting systems. Such bug reports will be written in natural languages containing a five-step description of how to reproduce those bugs. A good bug report should have the following information:

- It should be an efficient form of communication for both bug reporters and bug receivers.
- It should be filed in a defined way and should not be too lengthy.
- It should not contain unstructured or ungrammatical text.
- It should not contain non-technical language and spelling errors [22].

3.2 Bug Tracking System

It is essential as every application or software may have many bugs reported by the customers. Each of such bugs should be monitored, evaluated, and prioritized for debugging. Many bug tracking systems exist nowadays (e.g., Bugzilla, Jira, Mantis etc.) [25] to track the bugs reported by the customers. The main benefit of a bug-tracking system is to provide a clear centralized overview of the development and its state. A bug-tracking system is mainly developed with the purpose of helping programmers at fixing bugs. However, this may sometimes yield inaccurate results because different bugs may have different levels of severity and complexity.

3.3 Log File

Every application has a set of screens and a large number of GUI components in each such screen. Touching the GUI components can trigger events. These events occur when the description given by the customer is executed. The logfile logs all the events whenever the use cases are executed. The log file contains the user actions such as hard key events, soft key touch coordinates, visible view name, transition info, etc., and the GUI component with specific values.

© Copyright by Telkom University

3.4 Sequencer

The main aim of this step is to process the log file information and to identify the event sequences that are capable of reproducing the bugs. It generates input commands, generates view checks, and optimizes sequences.

3.5 Repeater

The repeater is used to improve the quality of bug reproduction. The large log file contents are converted into the sequence as the long sequences of actions may lead to inefficient results. It also sets required pre-conditions and activates required traces. The repeater will repeat the same process N number of times until the issue is seen.

3.6 GUI Engine

Next and one of the most critical components of this architecture is the GUI Engine that takes input from the sequencer that is capable of reproducing the bugs. The GUI Engine will have a UI test setup providing a generic API to simulate input events to target and deduct output states. The generic APIs are a set of functions that are understandable by the GUI Engine. The outputs of the repeater module are also in a form similar to the generic APIs of the GUI Engine. Then only the GUI Engine can understand the reproduction steps that were written in natural languages. Whenever a use case is executed based on the description given by the customer, the GUI Engine checks whether the issue is seen. If no issues are seen during the initial execution, it checks automatically whether the test is to be repeated; if yes, it gets redirected to the repeater module and repeats the same steps until the issue is seen. Whenever the issue is seen, it stops the test and saves the sequence. This saved sequence can be given to the developer to understand at what stage of the execution the issues are occurring. Likewise, this process can be repeated for any number of bug reports reported by the customer in an automated manner, thereby reducing the manual reproduction efforts.

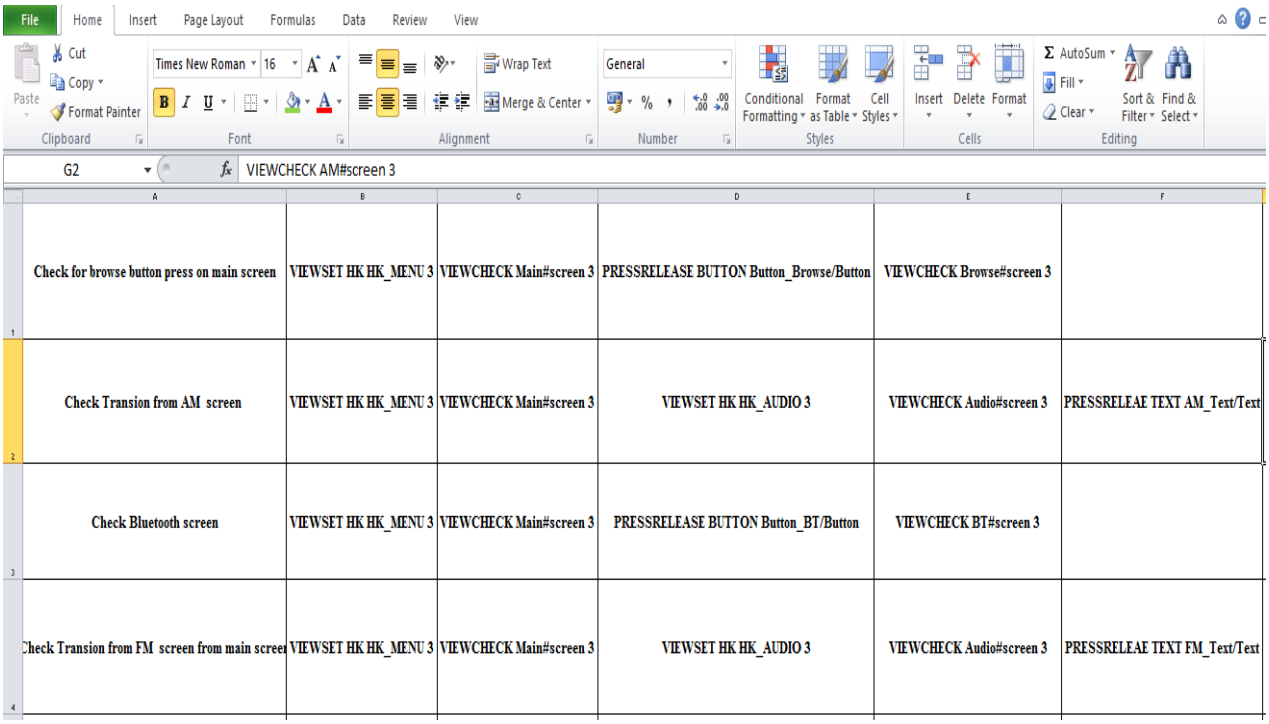

Fig.2 - GUI Engine

Fig.2 shows the structure of the GGUI Engine. Any sequence-based test could be automated using exposed APIs of this GUI engine.

3.6.1 GUI engine working

This is an excel sheet that is working as the GUI engine. Here, each cell of the excel is used to enter the use cases based on the data flow. The First cells of each row will contain the action or the use case that is to be executed. The sequence of flow is entered from the second cell of each row based on the use case. Here the length of each row will vary depending

on the use case. The cells after every use case will keep blank. So that during the execution time, use cases that were added row by row can be tested easily. Each use case will check the blank cell after the execution of each use case. If a blank cell comes, the count will increment and execution will start from the next row and these processes continue until the execution reaches a blank row and a blank column. In this case, we can enter any number of use cases based on the requirements. The design of the GUI engine is done with the help of a file called YAML file.

3.6.2 Role of YAML file in GUI engine design

YAML files are a set of files that are created by parsing the XML files corresponding to each and every screen of the application under test. The use cases are filled in the excel cells based on a particular syntax. The names of GUI components are required. To get information about the GUI components on a particular screen of the application, some commands need to be run and to search the resulting logfile. But it is a time taking task as the log file will contain much information about all the components. So searching the name of a particular name is a very time-consuming task. So, YAML files are used to get rid of parsing the large log files to get information about buttons and texts on the screen under test. The YAML files are created separately for each screen by parsing each screen of the test application. It will help to see the names of the required component by opening the file corresponding to the respective screen. Each YAML file is saved with the file name as the screen name. So that each component on the required screen can be identified easily. The YAML files are created by parsing the XML files corresponding to each screen of the application using an element tree parser. YAML files will automatically store in a particular folder location after parsing. That folder will contain all YAML files corresponding to all screens of the application with the .yaml extension. Fig.3 shows the architecture of the creation of YAML file.

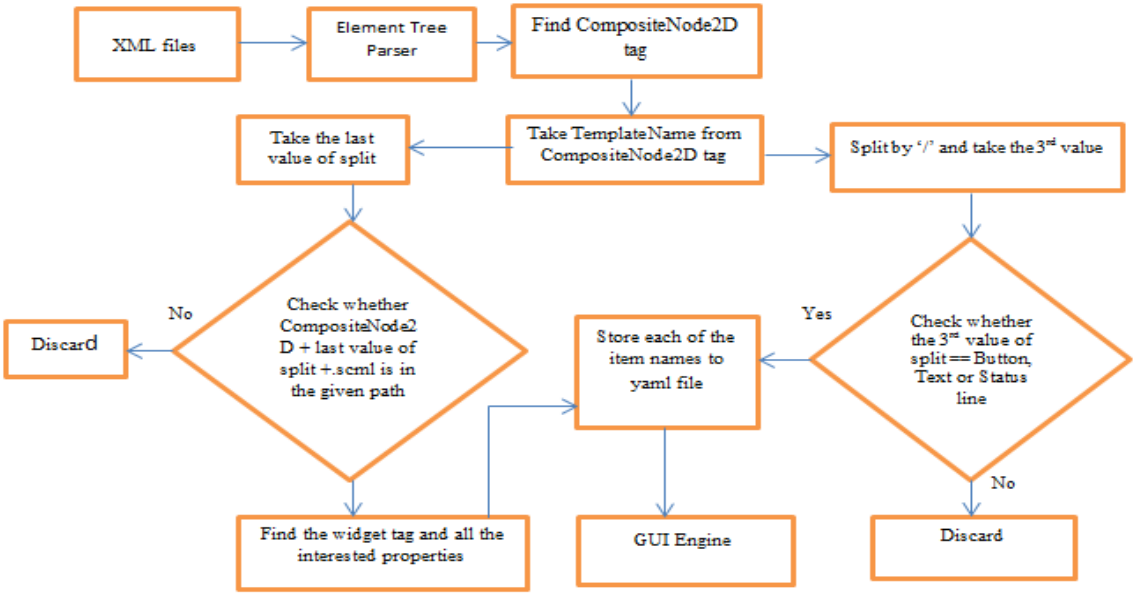

Fig.3 - Yaml File Creation Architecture

Fig. 3 shows the yaml file corresponding to a particular screen of the application under test. The number of yaml files depends on the number of screens of the test application. The name of each yaml file will be the same as the respective screens of the application. The informations in the yaml files are used to fill the uses cases in the GUI engine.

The currently available methods deal with either manual or automated bug reproduction for android applications only. The proposed approach has many advantages over the existing systems, as it could reproduce bugs even from manually written bug reports for non-android applications. This proposed method is different from all other existing approaches due to its increased test coverage and it can test every part of an application for every new release also. It doesn't require any human interventions, can do multiple parallel tasks while the test is running. One of the advantages of this approach is its faster delivery, as it takes only very little time compared to manual testing. Manual test results depend on the skills of a manual tester, and if he skips any test, it may affect the entire project. But automated testing helps to cover all the areas that are to be tested. Also, there is no need to worry about the ambiguity in the bug reports. One of the most

© Copyright by Telkom University

surpassing advantages of this approach is that once the actions corresponding to a use case are stored, it can be used N number of times to reproduce the same bugs, thereby reducing manual effort.

```
- Screen Name: AUDIO_DVD__MAIN
- AUDIO_DVD__MAIN
- Parent Path: /Modes
  -- Composite Type: ButtontnAudioControls
  -- Name: Button_Repeat
  -- Widget: Button_Repeat/[{widget_name:
                                Button, ButtonImage, ButtonImageIcon }]

- ParentPath: /Modes
  -- CompositeType: ButtontnAudioControls
  -- Name: Button_FR
  -- Widget: Button_FR/[{widget_name:
                               Button, ButtonImage,  ButtonImageIcon }]

- ParentPath: /Content/Buttons
  -- CompositeType: ButtonSource
  -- Name: Button_Browse
  -- Widget: Button_Browse/[{widget_name: Button,Text}]

- ParentPath: /Content/Texts
  -- CompositeType: TextStyle_Text
  -- Name: DisplayModeText
  -- Widget: DisplayModeText/[{widget_name: Text}]

- ParentPath: /Backgroung_BG
  --CompositeType: StatusLine
  --Name: StatusLine
  --Widget: StatusLine/[{widget_name: HeaderText, HeaderText_Long}]
```

Fig.4 - Input Yaml File

In summary, our paper makes the following outstanding contributions to the existing bug reproduction systems:

- The proposed method is mainly concentrated on helping developers to solve the functionality issues of applications.
- It is much efficient in dealing with the swipe gestures which were not properly working in the existing methods for android applications.
- It is capable of reproducing bugs for non-android applications only.
- It is an effective approach to deal with the ambiguity in the bug reports that are written in natural languages.
- It works well for all applications that have complex GUI behaviors.
- It is also concentrated on reproducing the same bugs any number of times in an automated manner.
- It is much faster and efficient than manual testing of applications.
- One of the limitations of this approach is that it is not efficient enough to deal with animation.

4. Result and Discussion

Fig.5 shows the result after executing the GUI Engine. Here the use cases are entered row by row in the excel cells and any number of use cases can be added in this excel. The flow of execution of each and every use case is centered horizontally. The excel cells are filled based on a particular syntax. The function name will be the first parameter followed by the GUI component name, press time, coordinates, etc. The green color in the excel cells shows that the GUI component entered in that particular cell is working fine without any failure and the red color shows that the GUI component in that particular cell is not working fine.

From this result itself, the tester can understand what all GUI components are working fine and what all GUI components are not working fine. A test case is working fine if and only if all the cells corresponding to that test case turned green. If any of the cells turned red, then that GUI component failed and thereby making the use case failed. All the logs are stored to the given path after executing all use cases entered in the excel cells. From the log file, the developers can easily

understand the reasons for the failure of GUI components. For example, if the view name entered in the excel cell is view1 and in the log file it is showing as view3, then the developer can easily understand that instead of view1 a button press or text press is going to some other page i.e. View2. With these results, the developer can make changes in their script to solve the GUI failures. The proposed method is found to be helpful for bug reproduction and for GUI testing in an automated manner for non-android applications.

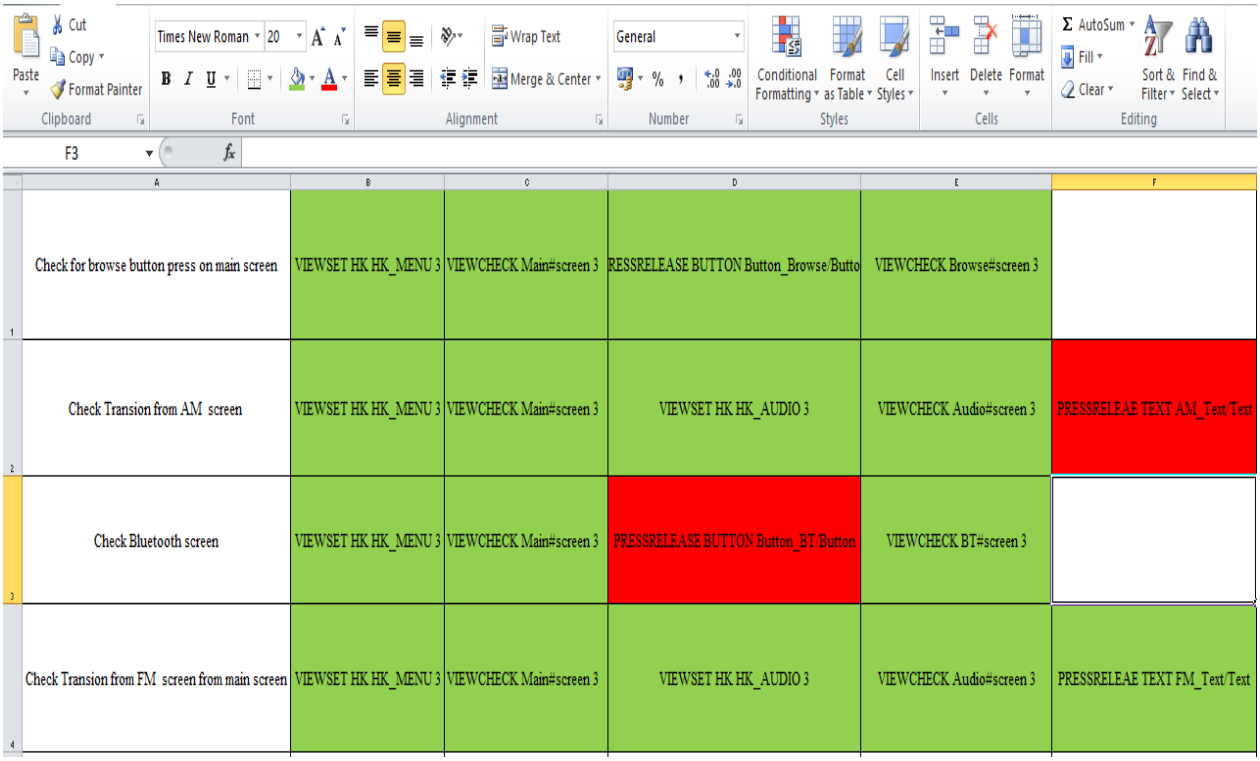

Fig.5 - GUI Engine Execution Summarized Result

The results showed that this method worked well with any number of GUI components. The efficiency of capturing the GUI events that lead to bug reproduction has considerably improved than any other existing method. Also, the newly introduced methods helped to reproduce the same issue any number of times based on the requirements. This method is found to be capable of testing almost all types of GUI actions. The application of this method could produce better results in both GUI testing and bug reproduction and it saves time and cost to a great extent when compared to manual testing. It also implies the real-time improvement of the application and thereby customer satisfaction. Table 1 shows the GUI components and their working status. So that the developer can easily understand which all GUI components have failed. So, the manual efforts in checking all the GUI components separately get reduced to a great extent.

Table.1 - GUI Engine Execution Summarized Result

Use case	GUI components used	Result	Failed GUI component
Check for a browse button press on the main screen	Hard key main, Browse button, View check	Passed	
Check transition from AM screen from the main screen	Hard key main, Audio button, View check, AM soft key	Passed	
Check Bluetooth screen	Hard key main, Audio button, View check, Bluetooth soft key	Failed	Bluetooth soft key
Check transition from FM screen from the main screen	Hard key main, Audio button, View check, FM soft key	Passed	
The transition from AM to FM screen	Hard key main, Audio button, View check, FM soft key, AM soft key	Passed	

© Copyright by Telkom University

Popup check-in FM screen	Hard key main, Audio button, View check, FM soft key, Popup	Passed	
Swipe check on the main screen	Hard key main, Swipe coordinates	Passed	
Check for a browse button press on the main screen	Hard key main, Browse button, View check	Passed	
Check transition from AM screen from the main screen	Hard key main, Audio button, View check, AM soft key	Passed	
Phone call check	Hard key main, Audio button, View check, Phone button, dial pad, popup	failed	Dial pad
Message check	Hard key main, Audio button, View check, the message button	Passed	
Volume increase in FM screen	Hard key main, Audio button, FM soft key, View check, Right Volume button, popup	Failed	Popup
Volume decrease in FM screen	Hard key main, Audio button, FM soft key, View check, Right Volume button, popup	Passed	
FM station change	Hard key main, Audio button, FM soft key, View check, Forward button, Backward button, pattern	Failed	Backward button
AM station change	Hard key main, Audio button, AM soft key, View check, Forward button, Backward button, pattern	Passed	
Play music	Hard key main, Audio button, Music soft key, View check, Forward button, Backward button, Pause button, pattern	Passed	

5. Conclusion

Every software or application might have bugs reported by the customers. Solving such issues by reproducing each reported bug manually by touching each GUI component makes the test ineffective. It is eventually vivid that manual reproduction works well for small projects. However, as the size of the project increases, manually recreating all the reported issues might consume a very significant amount of time. In addition to this, the ambiguity in the reports may not convey the actual thoughts of the customer. So, to solve all these issues, this paper introduces a novel approach that can automatically analyze bug reports that were written in natural languages and reproduce the bugs reported by the customers to solve the functionality issues of non-android applications. It helps developers to automatically reproduce bugs by identifying the GUI events from the description given by the customer that are capable of reproducing the bugs. One of the limitations of this approach is that the external triggers are not integrated. In the upcoming update, we can integrate external triggers to make the reproduction more effective. And the stored log file information cannot be used for different scopes. So, for each scope, we have to do all the processes separately.

Disclaimer

The authors whose names are written certify that they have no conflict of interest.

References

[1] Bauer, W., Hämmerle, M., Schlund, S., & Vocke, C. (2015). Transforming to a Hyper-connected Society and Economy – Towards an "Industry 4.0." Procedia Manufacturing, 3(Ahfe), 417–424. https://doi.org/10.1016/j.promfg.2015.07.200

[2] Joseph, G. V., & Thomas, K. A. (2020). Moderating effect of social media usage on technology barriers to technology adoption by teachers. International Journal of Advanced Science and Technology, 29(3), 5504–5512.

[3] Zhao, Y., Yu, T., Su, T., Liu, Y., Zheng, W., Zhang, J., & Halfond, W. (2019). ReCDroid: Automatically Reproducing Android Application Crashes from Bug Reports. Proceedings - International Conference on Software Engineering, 2019-May, 128–139. https://doi.org/10.1109/ICSE.2019.00030

[4] Qureshi, I. A., & Nadeem, A. (2013). GUI Testing Techniques: A Survey. International Journal of Future Computer

and Communication, 2(2), 142–146. https://doi.org/10.7763/ijfcc.2013.v2.139
[5] Bugzilla, T., Release, G., & Barnson, M. P. (2003). The Bugzilla Guide. Changes, 69(10), 80. http://ftp.wcss.pl/pub/doc/linux-docs/linux-doc-project/bugzilla-guide/Bugzilla-Guide.pdf
[6] Heck, P., & Zaidman, A. (2013). An analysis of requirements evolution in open source projects: Recommendations for issue trackers. International Workshop on Principles of Software Evolution (IWPSE), 43–52. https://doi.org/10.1145/2501543.2501550
[7] Canovas, J., Cosentino, V., Bergel, A., Canovas, J., Cosentino, V., Bergel, A., Gila, J. C., & C, J. L. (2015). GiLA : GitHub Label Analyzer To cite this version : GiLA : GitHub Label Analyzer. 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER).
[8] Moran, K., Linares-Vasquez, M., Bernal-Cardenas, C., Vendome, C., & Poshyvanyk, D. (2016). Automatically Discovering, Reporting and Reproducing Android Application Crashes. Proceedings - 2016 IEEE International Conference on Software Testing, Verification and Validation, ICST 2016, 33–44. https://doi.org/10.1109/ICST.2016.34
[9] Ormandjieva, O., Hussain, I., & Kosseim, L. (2007). Toward a text classification system for the quality assessment of software requirements written in natural language. SOQUA'07: Fourth International Workshop on Software Quality Assurance - In Conjunction with the 6th ESEC/FSE Joint Meeting, January, 39–45. https://doi.org/10.1145/1295074.1295082
[10] Roehm, T., Gurbanova, N., Bruegge, B., Joubert, C., & Maalej, W. (2013). Monitoring user interactions for supporting failure reproduction. IEEE International Conference on Program Comprehension, 73–82. https://doi.org/10.1109/ICPC.2013.6613835
[11] Gu, Y., Xuan, J., & Qian, T. (2016). Automatic reproducible crash detection. Proceedings - 2016 International Conference on Software Analysis, Testing and Evolution, SATE 2016, November 2016, 48–53. https://doi.org/10.1109/SATE.2016.15
[12] Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., & Poshyvanyk, D. (2018). FUSION: A tool for facilitating and augmenting android bug reporting. ArXiv, 1–4.
[13] Gómez, M., Rouvoy, R., Adams, B., & Seinturier, L. (2016). Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring. Proceedings - International Conference on Mobile Software Engineering and Systems, MOBILESoft 2016, 88–99. https://doi.org/10.1145/2897073.2897088
[14] White, M., Linares-Vásquez, M., Johnson, P., Bernal-Cárdenas, C., & Poshyvanyk, D. (2015). Generating Reproducible and Replayable Bug Reports from Android Application Crashes. IEEE International Conference on Program Comprehension, 2015-August, 48–59. https://doi.org/10.1109/ICPC.2015.14
[15] Chaparro, O., Lu, J., Zampetti, F., Moreno, L., Di Penta, M., Marcus, A., Bavota, G., & Ng, V. (2017). Detecting missing information in bug descriptions. Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Part F130154, 396–407. https://doi.org/10.1145/3106237.3106285
[16] Sun, J., Yan, K., Liu, X., Zhu, M., & Xiao, L. (2019). Automatically Capturing and Reproducing Android Application Crashes. Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019, 524–525. https://doi.org/10.1109/QRS-C.2019.00106
[17] Just, S., Premraj, R., & Zimmermann, T. (2008). Towards the next generation of bug tracking systems. Proceedings - 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008, June, 82–85 https://doi.org/10.1109/VLHCC.2008.4639063
[18] Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., & Zimmermann, T. (2007). Quality of bug reports in Eclipse. 21–25. https://doi.org/10.1145/1328279.1328284
[19] Zhang, T., Chen, J., Jiang, H., Luo, X., & Xia, X. (2017). Bug Report Enrichment with Application of Automated Fixer Recommendation. IEEE International Conference on Program Comprehension, 230–240. https://doi.org/10.1109/ICPC.2017.28
[20] Moran, K., Linares-Vasquez, M., Bernal-Cardenas, C., Vendome, C., & Poshyvanyk, D. (2017). CrashScope: A practical tool for automated testing of android applications. Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, i, 15–18. https://doi.org/10.1109/ICSE-C.2017.16
[21] Jha, A. K., & Lee, W. J. (2013). Capture and replay technique for reproducing crash in android applications. IASTED Multiconferences - Proceedings of the IASTED International Conference on Software Engineering, SE 2013, August, 783–790. https://doi.org/10.2316/P.2013.796-025
[22] Laukkanen, E. I., & Mäntylä, M. V. (2011). Survey reproduction of defect reporting in industrial software development. International Symposium on Empirical Software Engineering and Measurement, 2011, 197–206. https://doi.org/10.1109/esem.2011.28
[23] Hu, C., & Neamtiu, I. (2011). Automating GUI testing for android applications. Proceedings - International Conference on Software Engineering, Section 4, 77–83. https://doi.org/10.1145/1982595.1982612
[24] Chu, E. T. H., & Lin, J. Y. (2019). Automated GUI testing for android news applications. Proceedings - 2018 International Symposium on Computer, Consumer and Control, IS3C 2018, 14–17. https://doi.org/10.1109/IS3C.2018.00013
[25] Vlasceanu, I. V. (n.d.). A study concerning the bug tracking applications. 0–46.

© Copyright by Telkom University