2018 42nd IEEE International Conference on Computer Software & Applications

Test Case Prioritization for GUI Regression Testing based on Centrality Measures

Yijie Ren, Bei-Bei Yin*, Bin Wang
Department of Automatic Control
Beihang University, Beijing 100191, China
ryj_6585@163.com

***Abstract*—Regression testing has been widely used in GUI software testing. For the reason of economy, the prioritization of test cases is particularly important. However, few studies discussed test case prioritization (TCP) for GUI software. Based on GUI software features, a two-layer model is proposed to assist the test case prioritization in this paper, in which, the outer layer is an event handler tree (EHT), and the inner layer is a function call graph (FCG). Compared with the conventional methods, more source code information is used based on the two-layer model for prioritization. What is more, from a global perspective, centrality measure, a complex network viewpoint is used to highlight the importance of modified functions for specific version TCP. The experiment proved the effectiveness of this model and this method.**

Keywords—GUI Testing; Regression Testing; Test Case Prioritization; Event Handler Tree; Complex Network

I. INTRODUCTION

During the rapid development of GUI (Graphical User Interface) software, to ensure the correctness of code modification and avoid the side effects caused by code modification to other modules of the testing program, regression testing is used[1]. However, statistics show that the regression test accounts typically for more than 80% of the software product test budget, accounts for more than 50% of the software maintenance budget [3].

In many scenarios, the actual budget of the project is not allowed to execute all the test cases. For example, Rothermel et al. found in a partner enterprise that it took up to seven weeks to run all the test cases to test a software product containing about 20,000 lines of code[4]. With code modification, new test requirements are generated, and new test cases need to be designed. At this time, it is particularly important to optimize test cases.

According to the optimization objectives, the regression testing optimization technology for conventional software is mainly divided into three kinds [5], regression test case reduction[7], regression test case selection[8][9] and regression test case prioritization (TCP). In this paper, we mainly discuss the application of regression TCP on GUI related software.

Achievements have been made in the research of conventional software TCP. It can be traced back to the work by Wong in 1997[10].For specific program versions, they combine history coverage information in test cases and code modification information to propose a hybrid method. After that, varieties of TCP methods have been introduced and effectively improved the early coverage and early detection rate of defects in the program[10][11][12]

However, there are significant differences between the testing of GUI software and the testing of conventional software. Firstly, in GUI software, the inner layer code is encapsulated by Graphical User Interface (GUI), which makes it hard to cross the interface and consider the underlying code coverage directly. Secondly, the control flow and data flow of GUI software are more tedious, and the input is arbitrary. Thirdly, the definition and generation of test cases is more difficult. Therefore, it is not feasible to use the conventional TCP method in GUI software.

At present, there are few studies on the test case prioritization in GUI software testing, mainly based on event flow graph (EFG), which models events and their interactions. Brycehe and Memon propose a GUI TCP method for cross-window event interaction criteria[14] Huang proposed a TCP method based on weighted events and set different weights for different kinds of events[13].

Based on whether code modification analysis is performed, the TCP problems are divided into prioritization for general versions and that for specific versions. Conventional EFG based methods mainly concentrate on the general versions TCP. They are black-box testing methods, mainly discuss the interaction between events and do not involve in the source code. However, for specific versions TCP, the code modification information is the critical prioritization base, so the white-box testing method is essential. On the other hand, source code information (Event Handler) can make the test case more efficient [16]. It is also potential to use the internal source code information for GUI TCP.

Meanwhile, GUI is of high complexity due to its event-driven feature, the random input, and the increasing scale. The conventional EFG based methods rank the events from a local perspective, which may not achieve global priority goal and may occur state explosion problems.

To solve the above problems, based on the structural features of GUI software, we establish a two-layer model. The outer layer is an event handler tree (EHT) [17] model, the inner layer is a function call graph (FCG), and event handlers are the bridge between the two layers. For a specific test case, it executes specific events sequence, and then their handlers in EHT invoke corresponding functions which will form a specific FCG. The FCG involves internal code information, which has not been thoroughly used in conventional methods.

This work was supported by the National Science Foundation of China (No.61402027) and Equipment Preliminary R&D Project of China (No.41402020102).

0730-3157/18/$31.00 ©2018 IEEE
DOI 10.1109/COMPSAC.2018.10275

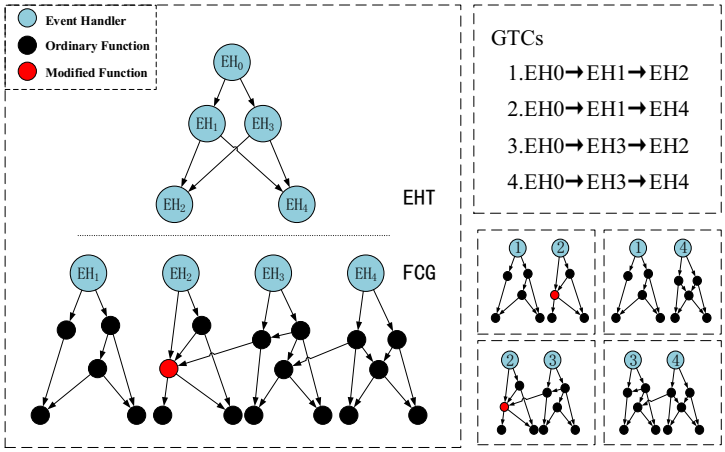

Fig. 1. Two-layer model of GUI software

On the other hand, considering the FCG, the complexity of GUI regression testing will grow even higher. To deal with this, this paper adopts the viewpoint of complex network. Although the internal structure of software network may be complex and there are many variables, the complex network concentrates on the relations between units from a global perspective, ignoring details[22][25]. In TCP, the key point is to analyze the ability of the test cases in covering the important parts. And complex network has a sound theoretical foundation in dealing with such problems by measuring the importance of nodes globally.

This paper focuses on the problem of TCP for specific versions (considering code changes). It is apparent, compared with the FCG corresponding to passed test cases, in the FCG corresponding to failed test cases, the nodes that represent modified function play a more critical role.

The method of network centrality measurement is selected as the ranking criterion to measure such importance. In complex network structure analysis, centrality analysis has been proved to be a very valuable method[22][25]. It can adequately reflect the importance of the node by solving problems from a global perspective. In this paper, the test case one-to-one corresponds to its FCG through the GUI two-layer model. For a GUI test case, the higher the network centrality of the modified function values in the FCG, the higher its priority is in the test case suite.

To sum up, in this paper, a GUI two-layer model is established and an approach to prioritize GUI test cases based on centrality measures is proposed. The advantages lie in two aspects. On the one hand, this method fully combines the characteristics of GUI software, makes use of the source information in test cases, to better guide the prioritization of test cases. On the other hand, as for the specific ranking criteria, it adopts the method based on the network centrality measures, which shows the importance of modified nodes in the test cases of error running from a global perspective.

The rest of this paper is organized as follows: Section II shows the related work. In section III, GUI two-layer model is introduced. In Section IV, the prioritization based on centrality measures is presented. Section V is the experiments and the results, and finally, the conclusion is given in Section VI.

II. RELATED WORK

This section is consist of two parts: a brief introduction of GUI regression TCP, which is the base of our work, will be presented in part A. In part B, the EFG model and methods based on it will be introduced.

A. GUI Regression Prioritization)

TCP can efficiently solve the problem of time and cost during GUI regression testing. The purposes of TCP is as follows: according to the suspiciousness of the test case, that is, whether it is easy to expose defects and execute the test case in turn, from high priority to low priority. This method is not related to when the test is terminated. It can effectively improve the early detection rate of the defect and maximize the efficiency of the tester. The definition of it is as follows[11]:

Definition 1: TCP is to find $T' \in PT$, such that $(\forall T'')(T'' \in PT, T'' \neq T') \rightarrow f(T') \geq f(T'')$. Where T is a test suite, PT is a set of permutations of T and f is a function from PT to a real number.

From the definition, to measure a TCP algorithm, it is necessary to first determine the function of a PT to a real number. The conventional scheme mainly includes maximizing the number of defects, obtaining higher coverage or improving the early detection rate of software defects [19][20]. This article is more concerned with the improvement of the early detection rate. To reduce unnecessary trouble and interaction between defects, we mainly consider the single defect situation. In practice, when software is fully tested, it is rare to introduce a large number of defects.

B. EFG Model

Conventional GUI regression TCP is mostly based on event flow graph[13][14][21], so here is a brief introduction to EFG model. It is defined as follows[2]:

Definition 2: An EFG is a 4-tuple $< V, E, B, I >$ where:

(1) V is a set of nodes in EFG, is the set of all possible events in GUI

(2) $E \subseteq V \times V$ is a set of directed edges in EFG, (u, v) $\in$ E represents event v can be executed after event u.

(3) $B \subseteq V$ is a set of nodes in EFG, represents the event that the can be performed directly when the GUI is first opened;

(4) $I \subseteq V$ is the set of the restricted-focus events.

In GUI software, due to the existence of inclusion and restriction, some events can only be executed after some specific events. To describe such features, Memon proposed the event flow graph (EFG) model. However, we can note that the EFG based method can only model the GUI outer layer features. It does not care about the internal source code information, and the code information is not fully utilized.

III. GUI TWO-LAYER MODEL

According to the two layers structure of GUI software, we establish a two-layer model. The outer layer is an event handler tree (EHT) model, which represents the dependence between external events of GUI software. EHT can solve the problem of generating legal test cases of GUI software. The inner layer is a function call graph (FCG) model, and the FCG represents the association of the internal code in the GUI software. It can combine internal source code information to help test case prioritization. In Fig 1, it is an example of the two-layer model.

A. Event Handler Tree

The event handler can respond to the user's operation; it is responsible for the interaction between the users and the GUI software. It is the bond of the two-layer model. There have been some studies on event handler [16][17][18]. The definition is as follows:

Definition 3: Event handler is such a particular function in GUI software: users write it, and when the corresponding event occurs, it is invoked automatically (it does not respond to the functions written by the users).

After a specific event, the event handler that corresponds to it will be invoked. In GUI software, once an event in the message queue is captured, it will not continue to transmit, so in general, each event at most corresponds to one event handler. However, an event handler may handle multiple events. On the other hand, some events do not call any event handler.

Due to the relationship between GUI software events, the sequence of events needs to be in a specific order. So the sequence of event handlers also needs to be in a specific order. The definition of EHT is as follows[17]

Definition 4: An EHT is a 4-tuple $< V, E, R, \mathrm{T} >$ where:

(1) V is a set of nodes in EHT, is the set of all possible event handlers in GUI.

(2) $E \subseteq V \times V$ is a set of directed edges in EHT, (u, v) $\in$ E represents event handler v depends on event handler u.

(3) $T \subseteq V$ is the set of the restricted-focus events.

(4) $R \subseteq V$ is the set of the termination events.

B. Function Call Graph

The function call graph is merely the call relation between functions. In the two-layer model, it is mainly used to reflect the function call relations between the internal source codes of GUI software. In this paper, the function call graph is defined as follows:

Definition 5: an FCG is a 3-tuple $< V, E, H >$, where:

(1) V is a set of nodes in FCG; it is the set of all possible functions in GUI

(2) $E \subseteq V \times V$ is a set of directed edges in EHT, (u, v) $\in$ E represents function v invokes function u.

(3) $T \subseteq V$ is the set of the event handler.

There are several points about FCG: 1.In an object-oriented language (Java, C++, C#, etc.), the call between functions can be divided into intra-class calls and inter-class calls. 2. The function call graph is not a strong correlation relation. If there is an edge between two nodes, it means that two nodes may have a calling relation.3. The in-degree of event handler is 0. It is because of the characteristics of EH that they are invoked automatically by the system instead of other function calls written by the user.

C. Summary

In summary, the definition of the two-layer model is as follows:

Definition 6: The two-layer model of GUI software is such a model: the upper layer is an EHT (definition 5) model, while the lower layer is an FCG (definition 6) model. EHT and FCG are connected by EH (definition 4).

The EH is the bond between the EHT and the FCG. Each EH exists in the two models. On the one hand, the EHT is built by the EH, so each node on the tree is an EH. On the other hand, in the FCG, the in-degree of EH is 0 due to its characteristics.

Thus, for each specific test case, it goes through a specific path in the EHT, that is, a specific sequence of the EHs. Each test case can construct its FCG through its sequence, so the prioritization of test cases can be transformed into the prioritization of FCGs as shown in Fig 1.

IV. CENTRALITY MEASURES

Ignoring the missing defects in the previous version, if a test case execution exposes a defect, compared with other test cases running regularly, the code modified functions that may introduce defects play a more critical role in the FCG. Therefore, to reflect this importance, this paper adopts the method of centrality measures to prioritize test cases.

The centrality measures is based on graph theory and complex network and measures the importance of nodes in the network. So far, many methods of centrality measures have been proposed[22], including degree centrality, decay centrality, closeness centrality, betweenness centrality, etc.

Considering the characteristics of GUI software network and the location of modified nodes [25] this paper mainly adopts two metrics: degree centrality and betweenness centrality.

Degree centrality uses the degree of nodes to measure the centrality; it is a relatively simple but obvious centrality measures. For example, in Fig 2, point B and point F have a large degree than the other points, that means these two nodes are more important. In the transmission of defects, because they are exposed to more nodes, they are more likely to be infected or infected with other nodes. In this paper, by dividing by the largest result, we get a value from 0 to 1 as follows:

$$C_i^{\deg}(\mathrm{g}) = \frac{d_i(\mathrm{g})}{\max(\mathrm{d}_i(\mathrm{g}))} \tag{1}$$

$d_i(\mathrm{g})$ represents the degree of node i in graph g. After normalization, the result is the degree centrality.

Degree centrality regards the popularity of the node as the measure, which can better reflect the direct influence of nodes on other nodes. But at the same time, there is no description of the location information or the structure of the node. So besides degree centrality, betweenness centrality is also used in this paper.

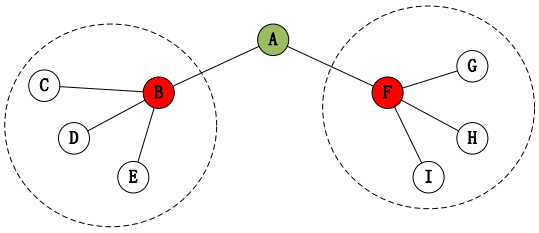

Fig. 2. Centrality measures

As shown in Fig 2, in addition to the B and F in the graph, the A point plays a vital role in the diagram. Although its degree is not very high, it connects the subnetwork in the two circles, so we can see that the node A is also essential as a bridge center. Without it, the defect can't propagate between the two circles of the network.

To better describe the structure centrality, betweenness centrality is selected, defined as the proportion of all the shortest paths passing through the node in the network as follows:

$$b_i(\mathrm{g}) = \sum_{(j,k), j \neq k \neq i} \frac{v_g(i:j,k)}{v_g(j,k)} \tag{2}$$

$$C_i^{bet}(\mathrm{g}) = \frac{b_i(\mathrm{g})}{\max(b_i(\mathrm{g}))} \tag{3}$$

$v_g(j,k)$ represents the number of shortest paths between nodes j to nodes k, and $v_g(i:j,k)$ represents the number of shortest paths between nodes j to nodes k and through nodes i. After normalization, the betweenness centrality is shown in equation (3). Therefore, the betweenness centrality is the probability that the network flows through of the node we are evaluating while it flows from one node to another node. It can reflect the structural importance in a graph as shown in Fig 2.

In this paper, the two central measures, including degree centrality and betweenness centrality are weighted as ranking criteria. If the modified node has a higher centrality in a function call graph corresponding to a test case, that test case will have a high prioritization. As shown in equation (4), Ψ is the set of centrality measurement methods used, the sum of the weights of various techniques is 1.

$$C_i(\mathrm{g}) = \sum_{\psi \in \Psi} (k_\psi \cdot C_i^\psi(\mathrm{g})) \tag{4}$$

In this way, we can sort the test cases in Fig 1; it can find that the modified node in test case 3 is critical regarding degree centrality or betweenness centrality. If there is no analysis of the source code, it is difficult to sort the test case 1 and test case 3 only through the black box analysis method.

V. EXPERIMENTS

A. Research Questions

Two major research questions will be studied in this article :

RQ1: In the practical TCP, is the two-layer model and centrality based method in this paper suitable for GUI software, and whether it has the potential and it is stable to improve the ranking effect?

RQ2: Whether there is a guide to choose different centrality for different GUI software?

B. Subject Applications

In this article, we choose the TerpPaint (TP) and TerpSpreadSheet (TSS) in the TerpOffice series as our subjects. They are often used for GUI software testing by researchers [16][17].

The information of the two subjects is shown in Table I. The defective versions of the two objects can be downloaded from the Benchmarks website of the Memon team [24]. However, most of these defects are directly implanted and are always triggered at runtime. But in reality, most of the defects need multiple combinations of interaction events to expose[23]. Therefore, we choose these defects as the defects of this experiment. The test case suites are generation by EHT method [17], each test case suite corresponds to a modified function.

TABLE I. SUBJECT APPLICATIONS

Subjects	Windows	Widgets	LOC	Classes	Methods	Faulty
TP	16	301	11,803	330	1253	263
TSS	9	176	5,381	135	764	234

C. Experiments Process

In this paper, we set up three separate experiments for the two research questions. But their primary experimental process is the same, as follows:

1. The two-layer model of the subject application is constructed.
2. A simple test case selection will be done. In this article, test cases not associated with a modified function will be eliminated.
3. For each of the different test cases, a function call graph is obtained through the event handler that invoked in the two-layer model.
4. Sorting test cases based on the centrality of the modified nodes in the function call graph.

Experiment 1and Experiment 2 is to verify the potential and stability of the method. In experiment 3, the setting of centrality is preliminarily explored.

Considering that a single modified node does not introduce too many defects, experiment 1 is set up: for each modified node, only one defect is introduced. After TCP, the position of a test case for the first discovery of a defect in the test case suite is used to measure the result. TF is the index of the first test case in the reordered test suite, n represents the scale of test suite as follows:

$$R = \frac{TF}{n} \tag{5}$$

To reduce randomness and enhance the persuasiveness of the experimental results, experiment 2 was set up. A single modified node introduced a lot of defects and calculated the APFD value of the centrality based prioritization method and the random prioritization method. APFD is used to measure the average of the percentage of faults detected in the execution process of test cases. It is described as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + \ldots + TF_m}{\mathrm{n} \cdot m} + \frac{1}{2n} \tag{6}$$

n represents the scale of a test suite, m represents the total number of defects that can be found in the test suite, TFi is the index of the first test case in the reordered test suite that reveals fault i. Generally speaking, the higher the APFD value, the better the sorting result.

For the second research questions, we choose the same subject application, the same modified node, and the corresponding defect, change the centrality measure method, compare the location of the first finding defect as in equation(5).

D. Experiment Results

Different or same modified nodes cause the defects in implantation. They are in the same or different research objects and files.

The result of Experiment 1, one defect experiment, is shown in Fig 3, ID is the number of the modified function, and R indicates the location of the first finding of defects in the test case suite. In general, 70% of the test case suites can detect defects before the position of 5%. For TSS, all the results are less than 25%. For TP, only one result is slightly more than 25%. For these barely satisfactory results, we found that the defects happen just after a specific isolated event and thus the centrality of modified function is low. But these defects have little interaction and can be found easily by other methods.

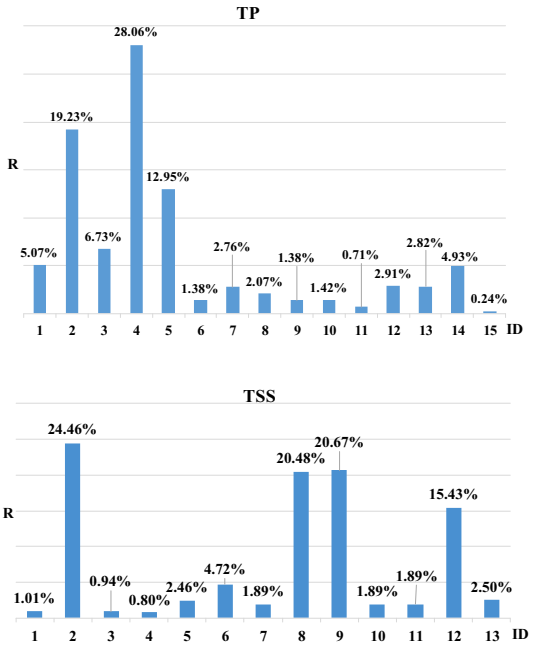

Fig 3. Experiment 1

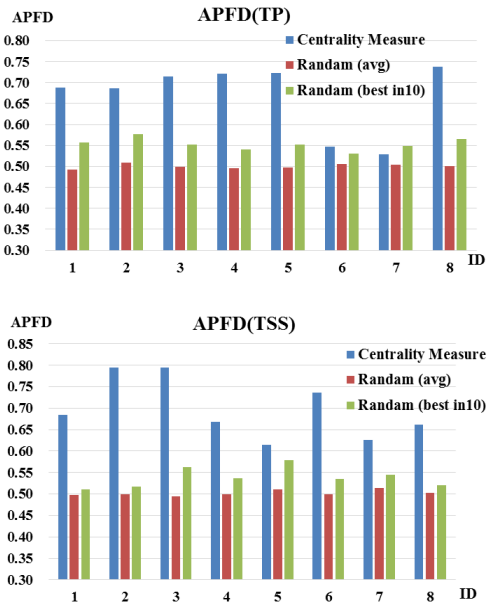

Fig 4. Experiment 2

In Experiment 2, the sorting results are in Fig 4; it is worth noting that APFD in the random ranking includes the average and the best result in ten times. It can be found that for all test

case suits in TP or TSS, the APFD value of this method is much better than that of the random prioritization method. It is known that this method has good stability.

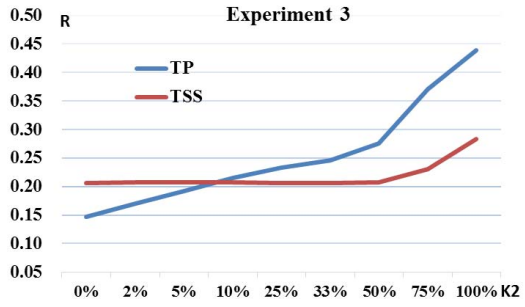

Fig 5. Experiment 3

The results of Experiment 3 are shown in Fig 5. For TSS, the ratio has little influence on the result, but if the ratio of betweenness is too big, the result may be worse. And for TP, the effect monotonously becomes worse as the ratio of betweenness centrality increases. It is consistent with the characteristics of betweenness centrality, which is more suitable for a network with a large scale. In TSS, the function call relationship is more complex than TP, so betweenness centrality does not have an adverse effect. Therefore, this paper suggests that the betweenness centrality is not suitable for the software on a smaller scale.

VI. CONCLUSION

In this paper, based on the characteristics of GUI software, we proposed a two-layer model of GUI software. And then, using this model, for regression TCP problem, we introduced a prioritization method based on centrality measure.

This method has the following advantages. (1) For specific version GUI TCP, we established a white-box testing model, and the source code information is fully used. (2)From the perspective of global viewpoint, we can find problems that cannot be found by local methods.

However, there are still many works to do for the two-layer model and centrality measure based prioritization. (1) From a global perspective, we consider using more complex network perspectives for prioritization. (2) The function call graph contains more source information and has not been fully utilized. (3) The combination of centrality method and existing conventional methods have a high potential for improving prioritization effect.

REFERENCES

[1] Memon, Atif M. Comprehensive Framework for Testing Graphical User Interfaces. Pittsburgh: University of Pittsburgh, 2001.

[2] Memon, Atif M., Mary Lou Soffa, and Martha E. Pollack. "Coverage criteria for GUI testing." ACM SIGSOFT Software Engineering Notes 26.5 (2001): 256-267.

[3] Leung, Hareton KN, and Lee White. "Insights into regression testing (software testing)." Software Maintenance, 1989., Proceedings., Conference on. IEEE, 1989.

[4] Rothermel, Gregg, et al. "Prioritizing test cases for regression testing." IEEE Transactions on software engineering 27.10 (2001): 929-948.

[5] Yoo, Shin, and Mark Harman. "Regression testing minimization, selection and prioritization: a survey." Software Testing, Verification and Reliability 22.2 (2012): 67-120.

[6] Sampath, Sreedevi, Renee Bryce, and Atif M. Memon. "A uniform representation of hybrid criteria for regression testing." IEEE transactions on software engineering 39.10 (2013): 1326-1344.

[7] Chen, Tsong Yueh, and Man Fai Lau. "Dividing strategies for the optimization of a test suite." Information Processing Letters 60.3 (1996): 135-141.

[8] J. Hartmann, and D. J. Robson, “Revalidation During the Software Maintenance Phase” [A], Proceedings of the 1989 Conference on Software Maintenance[C], 1989, pp.70-79.

[9] S.S. Yau, and Z. Kishimoto, “A Method for Revalidating Modified Programs in the Maintenance Phase” [A], Proceedings of 1987 Annual International Computer Software and Applications Conference[C], 1987, pp.272-277.

[10] Wong, W. Eric, et al. “A Study of Effective Regression Testing in Practice.” Eighth International Symposium on Software Reliability Engineering IEEE Computer Society, 1997:264.

[11] Rothermel, Gregg, et al. "Test case prioritization: An empirical study." Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on. IEEE, 1999.

[12] Elbaum, Sebastian, Alexey G. Malishevsky, and Gregg Rothermel. "Test case prioritization: A family of empirical studies." IEEE transactions on software engineering 28.2 (2002): 159-182.

[13] Huang, Chin-Yu, Jun-Ru Chang, and Yung-Hsin Chang. "Design and analysis of GUI test-case prioritization using weight-based methods." Journal of Systems and Software 83.4 (2010): 646-659.

[14] Bryce, Renée C., and Atif M. Memon. "Test suite prioritization by interaction coverage." Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting. ACM, 2007.

[15] Chaudhary, Neha, Om Prakash Sangwan, and Yogesh Singh. "Test case prioritizationusing fuzzy logic for gui based software." the Proceedings of International Journal of Advanced Computer Science and Applications 3.12 (2012).

[16] Zhao, Lei, and Kai-Yuan Cai. "Event handler-based coverage for GUI testing." Quality Software (QSIC), 2010 10th International Conference on. IEEE, 2010.

[17] Wang, Bin, Bei-Bei Yin, and Kai-Yuan Cai. "Event Handler Tree Model for GUI Test Case Generation." Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual. Vol. 2. IEEE, 2016.

[18] Yuan, Xun, and Atif M. Memon. "Iterative execution-feedback model-directed GUI testing." Information and Software Technology 52.5 (2010): 559-575.

[19] Huang, Chin-Yu, and Michael R. Lyu. "Optimal release time for software systems considering cost, testing-effort, and test efficiency." IEEE transactions on Reliability 54.4 (2005): 583-591.

[20] Huang, Chin-Yu, and Michael R. Lyu. "Optimal testing resource allocation, and sensitivity analysis in software development." IEEE Transactions on Reliability 54.4 (2005): 592-603.

[21] Belli, Fevzi, Mubariz Eminov, and Nida Gokce. "Prioritizing coverage-oriented testing process-an adaptive-learning-based approach and case study." Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International. Vol. 2. IEEE, 2007.

[22] Bloch, Francis, Matthew O. Jackson, and Pietro Tebaldi. "Centrality measures in networks." (2017).

[23] Memon, Atif M., and Qing Xie. "Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software." IEEE transactions on software engineering 31.10 (2005): 884-896.

[24] A.M. Memon, http://www.cs.umd.edu/~atif/Benchmarks/UMD2006b.html

[25] Zhu L Z, Yin B B, Cai K Y. Software Fault Localization Based on Centrality Measures[C] IEEE, Computer Software and Applications Conference Workshops. IEEE Computer Society, 2011:37-42