http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information and Technology Journal 5 (4): 648-654, 2006
ISSN 1812-5638

Important Usage Paths Selection for GUI Software Testing

[1]Ye Mao, [1]Feng Boqin, [1]Huang Zhenfang and [2]Zhu Li
[1]Department of Computer Science Technology,
Xi'an Jiaotong University, Xianning Road, Xi'an 710049, China
[2]Institute of Software, Xi'an Jiaotong University, Xianning Road,
Xi'an 710049, China

Abstract: From the user's point of view, the reliability of software depends greatly on the manner in which the software is used. As a result, it is necessary to test the software according to some model that highlights critical usage. Windows Navigation Networks (WNN) were proposed to model the usage of GUI software. Vertexes in the model represent windows and arcs represent transitions between windows. Each transition has probability of occurrence. Algorithm was proposed to obtain the transition probability from software usage log automatically. Important paths can be selected based on the WNN. Existing testing technologies then be used to test the important paths. WNN can describe the usage of GUI software from users' view and reduce the complexity of modeling GUI software. Important usage paths of GUI software can be got from WNN. It can focus the testing to reveal more important faults.

Key words: Software testing, GUI, WNN model, statistical testing, usage model

INTRODUCTION

GUI (Graphical User Interface) testing is difficult because it has characteristics different from those of traditional software (Memon, 2001). It is graph oriented and event driven. The input space size is extremely large due to the number of different permutations of inputs and events that affect the GUI (White and Almezen, 2000). Testing all these permutations is time and labor consuming. To reduce the number of test cases, different methods were proposed. When FSM was used to model GUI software (Chen and Subramamiam, 2002; Belli, 2001; Shehady and Siewiorek, 1997; Petrenko and Yevtushenko, 2005; Yip and Robson, 1991). Offutt *et al.* (2003) presented formal testing criteria for generating test inputs, which are transition coverage, full predicate coverage, transition-pair coverage, and complete sequence coverage. When the test cases generated satisfied the criterion, it is no need to generate more test cases. White *et al.* (2003) can reduced the number of test cases from 50 to 8 by identifying CIS of GUI software. Bogdanov and Holcombe (2004) discovered that the existing state chart testing method requires a rather large test set. By introduction of additional constraints, test set size could be reduced without weakening of the conclusions obtained by testing. Memon *et al.* (2005) used event-flow graph to represent GUI. Intra-component coverage and inter-component coverage criteria were used to control the test cases generated for testing GUI. Du *et al.* (2002) used interface component relating chart to model GUI. Vertex coverage, arc coverage, path coverage, and input/output space coverage criteria were presented to control the size of test cases. Nie and Xu (2003) proposed a minimal test suite generation method based on the interrelations among the testing requirements. All the effort above was to reduce the number of test cases and at the same time insure the reliability of the software.

In last years, the software engineering community has turned its attention to statistical software testing (Sayre, 1999; Cristiano *et al.*, 2004; Bjorn *et al.*, 2000). The main idea is that the reliability of software depends greatly on the manner in which the software is used. Not all software failures have an equal impact on user. The likelihood (probability) of encountering the failure has a significant affect on the importance of the failure to the user. As a result, it is necessary to focus testing so as to reveal the more important faults, i.e., test the software according to some model that highlights critical usage. Usage model is important in statistical testing. It characterizes the operational use of a software system. For almost a decade the software engineering community has been using Markov Chain (MC) to describe usage models (Jame and Michael, 1994; Yan *et al.*, 2005; Prowell, 2005; Beyer *et al.*, 2003). Farina *et al.* (2002) admit the use of Stochastic Automata Networks (SAN) to model applications. SAN formalism has exactly the same application scope as the

Corresponding Author: Ye Mao, Department of Computer Science Technology, Xi'an Jiaotong University, Xi'an 710049, China

MC. SAN formalism may cope the state space explosion by a modular way to model and an efficient numerical treatment of the infinitesimal generator matrix.

A usage model of Windows Navigation Networks (WNN) was proposed in this study. It models the usage of GUI software from its characteristics. The purpose of this paper is to show WNN can model GUI effectively and can be used to select important usage paths of GUI software for testing. The specific contributions of this work include:

- Proposing WNN to model the use of GUI software.
- Presenting the algorithm to obtain transition distribution of WNN from usage log automatically.
- Selecting important usage paths from WNN for testing.

PROBLEM DOMAIN

The field reliability, or reliability of software from the user's point of view, depends greatly on the manner in which the software is used. For example, consider a system that performs a number of functions correctly and one function incorrectly. If the incorrect function is rarely used few field failures will be observed and the software will have high field reliability. However, if the incorrect function is frequently used, the software will exhibit a large number of field failures and have low field reliability. As a result, it is necessary to focus testing so as to reveal faults on functions that are used frequently. We name these functions as critical usage. The important characteristics of GUI are their graphical orientation that is implemented by windows. Windows supply usage paths for users to accomplish functions.

Definition: GUI is a graphical front-end to a software system that accepts as input user-generated and system-generated events and produces deterministic graphical output. GUI contains windows. At any time during the execution of GUI, only one window is active.

We represent windows in GUI as $W = \{w_i | 1 = i = k\}$ k is the number of windows in GUI software. It needs to note that windows here include dialog and menu. When user operates GUI software, windows are invoked in sequence. Let it be $w_1\ w_2 ......w_n$ After many users have operated the software, there must be windows sequences that have high occurrence probability. These windows sequences represent the critical usage of GUI software from the views of users. For example, if one sequence is $w_1w_3w_5w_3w_6w_4w_9$, another sequence is $w_1w_3w_4w_9$, the sequence $w_1w_3w_4w_9$ has high probability to occur. It is necessary to focus testing so as to reveal the faults in $w_1w_3w_4w_9$. But to find the critical usage of GUI software is difficult because the number of possible usage sequences is extremely large due to the different permutations of inputs and events. The purpose of the paper is to find these sequences and focus testing to them.

WINDOWS NAVIGATION NETWORKS

In software testing there are several model to describe the usage of software, for example FSM, MC. In the paper, we proposed a model of WNN. It can model the GUI software from its characteristics.

Definition: WNN is a structure (S, T, R), where:

$S = \{s_i | 0 \le i \le n\}$ is a set of states. s_0 is initial state. Application is in state s_0 before being invoked. s_n is terminal state. Application is in state s_n after being terminated.

$s_i (1 \le i \le n-1)$ is the state when window in GUI software is active.

$T = \{t_i | 0 \le i \le m\}$ is a set of transitions. Each transition is a function t_i: $S \rightarrow S$. Let $t_i = <s_u, s_v>$, s_u and s_v are source state and destination state of the transition t_i, respectively, $0 \le u, v \le n$.

$R = \{r_i | 0 \le i \le m\}$ is a set of transition probability functions, one for each transition. Each function r_i: $T \rightarrow [0, 1]$ describes the probability of occurrence of the transition.

Usage matrix M is used to maintained transition probability. $M[u][v] = R(t_i)$ if $t_i \in T$ and $M[u][v] = 0$ if $t_i \notin T$. As a result, M is the probability distribution of WNN. M represents usage profile of GUI software.

Usage path starts from state s_0 and ends at state s_n. Let p[k] be the kth state in path P. For example, in path $p = s_0s_3...$, $P[1] = s_0$ and $P[2] = s_3$.

Definition: Usage path P is a sequence of states $s_0....s_i....s_n$, $i<n$. Usage path P_i is same as P_j iff $\forall k$, $p_i[k] = p_j[k]$. Usage path P_i is different from P_j iff, $\exists k$, $p_i[k] \ne p_j[k]$.

Usage path can also be regarded as a sequence of transitions because each transition is a function t_i: $S \rightarrow S$. Assume that the probability of occurrence of every transition in WNN is independent statistically. Let t_i be transitions in path p.

Definition: The occurrence probability of the path p_i is
$$\eta_i = \prod_{t_j \in p_i} t_j$$

It means that occurrence probability of a path depends on each transition in the path.

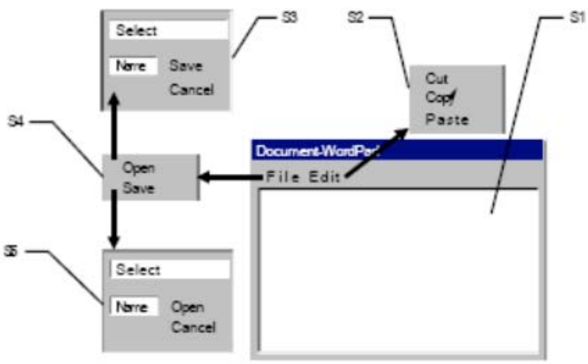

Fig. 1: The example GUI

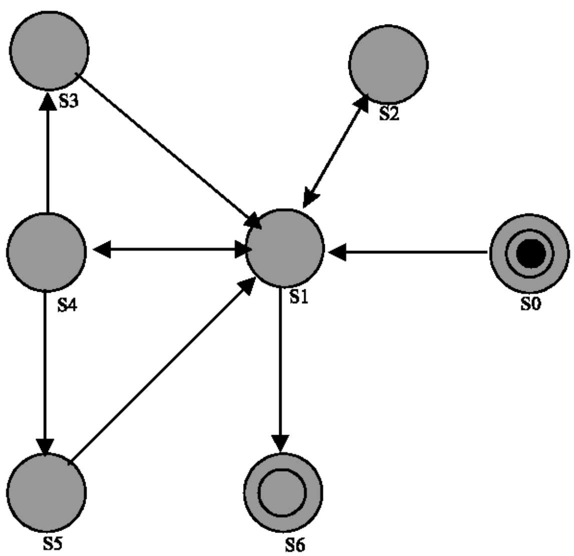

Fig. 2: WNN model

WNN has been shown to have a number of advantages for modeling GUI software in comparison with FSM and MC. Firstly the construct of WNN is intuitive because each physical window is represented by one state. Secondly, the number of states in WNN is bounded by $k + 2$, where k is the number of windows in GUI. As a result, the number of states is reduced largely. Thirdly, each transition is associated with a probability, which can reflect the usage frequency as well as the importance of the transition. Finally, the path from the initial state to terminal state can describe a usage of the GUI software.

The benefit of using a WNN can be seen clearly with an example. Consider a simple user interface with five windows. The first window, s_1, is the main window, where user can choose to go to other windows. s_4 is pull-down menu of file where user can choose to go to either save window (s_3) or open window (s_5). If user clicks edit menu in the main window, pull-down menu (s_2) of edit appears. An intuitive diagram of this interface is presented in Fig. 1. Note pull-down menu is treated as a window.

A WNN model of this interface is presented in Fig. 2. Initial and terminal states are s_0 and s_6, respectively. State from s_1 to s_5 corresponding to one window in GUI respectively. The transition from s_3 to s_1 means user can either save the data or cancel back to the main window. Transitions of $<s_4, s_1>$, $<s_5, s_1>$ and $<s_2, s_1>$ have similar means.

```
//begin
S0
S1
S4
S5
S1
S2
...
//end
```

Fig. 3: The example of usage log

OBTAIN SOFTWARE USAGE PROFILE

Transition probability is needed for generating usage paths statistically. It is obtained from the usage log. Usage log records sequence of the windows operated by user. The usage log has the form as Fig. 3. Technique such as window hook can be used to generate usage log. Alternatively, the codes record the windows usage can be inserted into the debug version of the software to generate usage log.

In the testing process, users are asked to use software. Usage log can be got at the same time. After obtaining the usage log files, we can compute the transition probability automatically. Let FILE be the set of all log files. Procedure CompTranProb was used to compute transition probability of WNN model. The input to procedure is FILE and the output is usage matrix M. The algorithm is as follows. Firstly, compute the frequency of every transition from usage log files (lines 2...13). Secondly, compute transition probability according to frequency (lines 14...28). Variable of freqSum (line 17, 21, 26) is used to maintain the sum of frequency of each transition leading from a state.

PROCEDURE CompTranProb(FILE, M)
Input: FILE;
Output: M;
Begin
1. Initiate M to zero;
2. For All file∈FILE do

```
3.  {
4.        w = Get the first window from file;
5.        u = Get the state in WNN corresponding to w;
6.        while(!eof)
7.        {
8.           w = Get the next window from file;
9.           v = Get the state in WNN corresponding to w;
10.          M[u][v]++;
11.          u = v;
12.       }
13. }
14. For All u∈S do
15. {
16.       E = Get all transitions leading from state u;
17.       freqSum = 0;
18.       For All transition ∈ E do
19.       {
20.          v = Get destination state of the transition;
21.          freqSum + = M[u][v];
22.       }
23.       For All transition ∈ E do
24.       {
25.          v = Get destination state of the transition;
26.          M[u][v] = M[u][v]/ freqSum;
27.       }
28. }
End.
```

SELECTING IMPORTANT USAGE PATHS

Matrix M maintains the probability of each transition. It represents usage profile of GUI software. The next step is to generate usage paths according usage profile and select important ones. Paths are generated by randomly walking in the WNN according to probability distribution. Each path begins from state s_0 and end at the state s_n. The problem is when to stop the paths generation process. We adopted Euclidean distance as stop criterion. Euclidean distance is commonly used to provide an indication of the degree to which the paths generated from the Markov chain matches the probability distribution. It is computed as indication $\sqrt{\sum_{i,j}(u_{i,j}-t_{i,j})^2}$ where, u_{ij} and t_{ij} are probabilities of going from state i to state j in the usage chain and testing chain, respectively (Sayre, 1999). In our paths generating process, $u_{i,j}$ is equal to M[i][j]. $t_{i,j}$ is the distribution of paths-generating process. It is computed as follows. During paths generating process, let $t_{i,j}$ maintain the frequency that the transition t = <s_i, s_j> is selected. After process finished, compute the probability that each transition is selected. The algorithm is same as line 14 to 28 of the procedure CompTranProb. Once $\sqrt{\sum_{i,j}(u_{i,j}-t_{i,j})^2} \le \varepsilon$, where ε is a threshold, stop path-generating process.

Let P be the set of all paths generated from initial state to terminal state. Paths in P are different from each other. Each $p_i \in P$ has an occurrence probability η_i, which has been defined. We define indicator $\rho_i = \eta_i / \sum_{i=1}^{|P|}\eta_i$. As a result, $\sum_{i=1}^{|P|}\rho_i$ must be equal to 1. We will use the ρ instead of η as occurrence probability because ρ has same distribution as η and $\sum_{i=1}^{|P|}\rho_i = 1$.

Occurrence probability ρ is computed for every path. The paths will be select according to the value of ρ. The paths with large value are importance paths, and then these paths can be tested by existing testing technology (White and Almezen, 2000; Chen and Subramamiam, 2002; Belli, 2001).

A CASE STUDY

We will show that the method is effective for GUI software. We performed a preliminary case study, which consisted of the following steps:

- Choice an example GUI application;
- Construct WNN for objective application;
- Generate usage log files;
- Compute the transition probability of WNN from the usage log files;
- Generate usage paths from WNN;
- Select important paths according to occurrence probability.

The application we selected was Microsoft WordPad5.0. Figure 4 gives the WNN of the application. It only contains twenty-one windows, which is a part of windows in WordPad5.0. s_0 is initial state and s_{22} is terminal state. s_1 is the main window. Every state $s_i(1 \le i \le 21)$ in the WNN corresponds to a window in WordPad. There are totally twenty-three states and thirty-six transitions in WNN.

Usage log files can be got after users use the software in testing process. According to Fig. 4, we generated files manually by walking in the graph from state s_0 to s_{22}. The number of log files we generated is 100. Figure 5 is a fraction of records from the 93th log file. We can see from the log file that the first state is s_0.

From the log files, we can compute the probability of every transition in WNN by running the algorithm CompTranProb. The probability of every transition that

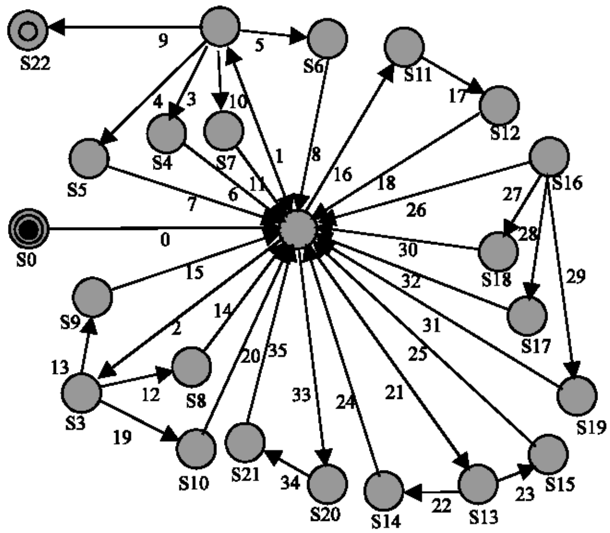

Fig. 4: WNN for WordPad5.0

//begin
S0
S1
S16
S19
S1
S11
S12
S1
S13
S14

Fig. 5: A fraction of log records

results from all the 100 log files is shown in Table 1. The column of ID shows transition number and Probability shows the probability of occurrence. The table describes the usage profile (probability distribution) of the WordPad. We can see that the probability is different from transition to transition.

The next step was to generate all usage paths by probability distribution. Euclidean distance is used in this step as stop criteria. Threshold was set to 0.0008. When Euclidean distance converges to threshold, the process of generating paths was stoped. From Fig. 6 that when path is generated one by one, paths-generating process can converge to usage profile. The general trend of curve is downward because the generating process approach to usage profile statistically. It shows that Euclidean distance is an effective indicator for stop criterion. The number of paths generated was 538.

In this step we selected the important paths according the value of ρ. Before selecting, some path same to other in these 538 paths must be deleted to form

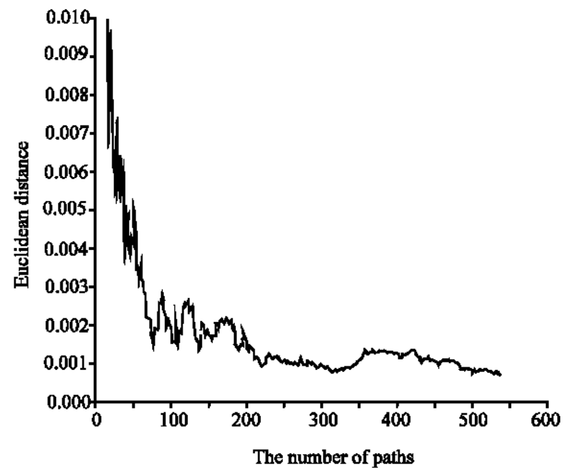

Fig. 6: Convergence of Euclidean distance to threshold

Table 1: Probability of every transiton

ID	Probability	ID	Probability	ID	Probability
0	1.0000	12	0.3924	24	1.0000
1	0.1115	13	0.4522	25	1.0000
2	0.1680	14	1.0000	26	0.2051
3	0.2282	15	1.0000	27	0.0590
4	0.2209	16	0.2867	28	0.4316
5	0.1289	17	1.0000	29	0.5094
6	1.0000	18	1.0000	30	1.0000
7	1.0000	19	0.1554	31	1.0000
8	1.0000	20	1.0000	32	1.0000
9	0.2075	21	0.1234	33	0.1053
10	0.2146	22	0.9125	34	1.0000
11	1.0000	23	0.0875	35	1.0000

Table 2: Probability of every transiton

ID	ρ	Path
174	0.558959	S0S1S2S22
161	0.160248	S0S1S11S12S1S2S22
76	0.062924	S0S1S13S14S1S2S22
314	0.058866	S0S1S20S21S1S2S22
165	0.042449	S0S1S3S9S1S2S22
137	0.036843	S0S1S3S8S1S2S22
151	0.016876	S0S1S20S21S1S11S12S1S2S22
491	0.01459	S0S1S3S10S1S2S22
471	0.014226	S0S1S2S4S1S2S22
503	0.010563	S0S1S3S8S1S11S12S1S2S22
384	0.006627	S0S1S20S21S1S13S14S1S2S22
211	0.006151	S0S1S16S19S1S20S21S1S2S22
293	0.003489	S0S1S11S12S1S11S12S1S3S9S1S2S22

the paths set of P. After processing, there were total 520 paths left in P. Table 2 gave some paths from P. The first column is path number. The second column is ρ and the third column is the path described by state sequence. Paths have been sorted by the value of ρ from large to small. It shows that some paths have larger value of ρ than others. The first 13 paths have the value of $\rho = \sum \rho_i = 0.992811$. It means that these paths out of all paths have very high probability to occur statistically while other 507 paths just have occurrence probability of $1\text{-}\rho = 0.007189$. It is needed to emphasize again here that paths we get may not be real path but reflect the statistical distribution of real paths. We determined these 13 paths as important paths. Existing testing technique then will be used to test these paths thoroughly.

It is shown from the case study that the important paths in GUI software are not too much but they have more chance to occur than other paths. By testing these important paths, we can insure the reliability of important usage. The cost of testing is not too expensive because important paths are just a little part from all possible paths of the GUI application.

CONCLUSIONS

Testing GUI software has become extremely important as GUIs become increasingly complex and popular. Testing all function is expensive because the input space size is extremely large due to the number of different permutations of inputs and events that affect the GUI. It is necessary to focus testing so as to reveal the more important faults. We construct model for GUI software from users' view. Important usage paths can be selected from the model. Existing technology then can be applied to test these paths. At this point, some preliminary conclusions are available:

- WNN can reduce the complexity of modelling GUI software.
- WNN can describe the usage of GUI software from users' view.
- Important usage paths of GUI software can be easily got from WNN. It can focus the testing to reveal more important faults.
- It is possible to obtain transition probability of WNN from usage log automatically.

The experiment described here is first case study. The target application is relatively simple, and the usage log we used here is generated from simulation. The next steps include: analysing the number of importance paths from the model, obtaining the usage log from real testing environment. Also, we will develop experiments with more complex target applications.

ACKNOWLEDGMENT

This research was supported by the National High-Tech Research and Development Plan of China under Grant No. 2003AA1Z2610.

REFERENCES

Belli, F., 2001. Finite state testing and analysis of graphical user interfaces. In: Proc. of the 12th International Symposium on Software Reliability Engineering , IEEE, pp: 34-43.

Beyer, M. *et al.*, 2003. Automated TTCN-3 test case generation by means of UML sequence diagrams and Markov chains. In: Proc. of the 12th Asian Test Symposium, IEEE, pp: 102-105.

Bjorn, R. *et al.*, 2000. Towards integration of use case modeling and usage-based testing. Systems and Software, 50: 117-130.

Bogdanov, K. and M. Holcombe, 2004. Refinement in statechart testing. Software Testing, Verification and Reliability, 14: 189-211.

Chen, J. and S. Subramaniam, 2002. Specification-based testing for GUI-based applications. Software Quality Journal, 10: 205-224.

Cristiano, B. *et al.*, 2004. Test case generation using stochastic automata networks: quantitative analysis. In: Proc. of the Second International Conf. on Software Engineering and Formal Methods, IEEE, pp: 251-260.

Du, S. *et al.*, 2002. Software functional testing technology based on interface component relating chart. China: J. Computer Res. Develop., 39: 148-152.

Farina, A. *et al.*, 2002. Representing software usage models with stochastic automata networks. In: Proc. of the 14th International Conf. on Software Engineering and Knowledge Engineering, ACM, pp: 401-407.

Jame, W. and T. Michael, 1994. A markov chain model for statistical software testing. IEEE Trans. Software Eng., 20: 812-824.

Memon, A., 2001. A comprehensive framework for testing graphical user interfaces. Ph. D Thesis, University of Pittsburgh, USA., 2001.

Memon, A. *et al.*, 2005. Automating regression testing for evolving GUI Software. J. Software Maintenance and Evolu., 1: 27-64.

Nie, C. and B. Xu, 2003. A minimal test suite generation method. China: Chinese J. Comput., 26: 1690-1695.

Offutt, J. *et al.*, 2003. Generating test data from state-based specifications. Software Testing, Verification and Reliability, 13: 25-53.

Petrenko, A. and N. Yevtushenko, 2005. Testing from partial deterministic FSM specifications. IEEE Trans. Comput., 54: 1154-1165.

Prowell, S., 2005. Using Markov chain models to test complex systems. In: Proc. of the 38th Hawaii International Conference on System Sciences, IEEE, pp: 318-323.

Sayre, K., 1999. Improved techniques for software testing based on Markov chain usage models. Ph. D Thesis, University of Tennessee, Knoxville, USA, 1999.

Shehady, R. and D. Siewiorek, 1997. A method to automate user interface testing using variable finite state machines. In: Proc. of the 27th Ann. Intl. Symposium on Fault-Tolerant computing, IEEE, pp: 80-88.

White, L. and H. Almezen, 2000. Generating test cases for GUI responsibilities using complete interaction sequences. In: Proc. of the 11th International Symposium on Software Reliability Engineering, San Jose, CA, IEEE, pp: 110-121.

White, L. *et al*., 2003. Firewall regression testing of GUI sequences and their interactions. In: Proc. of the International Conf. on Software Maintenance. IEEE, pp: 398-409.

Yan, J. *et al*., 2005. Deriving software Markov chain usage model from UML models. China: J. Software, 16: 1386-1394.

Yip, S. and D. Robson, 1991. Applying formal specification and functional testing to graphical user interfaces. In: Proc. of the 5th Annual European Conf. on Advanced Computer Technology: Reliable Systems and Applications, IEEE, pp: 557-561.