J. Software Engineering & Applications, **2008, 1: 38-43**
Published Online December 2008 in SciRes (www.SciRP.org/journal/jsea)

Development of an Improved GUI Automation Test System Based on Event-Flow Graph

Yongzhong Lu[1], Danping Yan[2], Songlin Nie[3], Chun Wang[1]

[1]School of Software Engineering, Huazhong University of Science & Technology, Wuhan 430074, P. R. China, [2]School of Public Administration, Huazhong University of Science & Technology, Wuhan 430074, P. R. China, [3]School of Mechanical Science and Engineering, Huazhong University of Science & Technology, Wuhan 430074, P. R. China
Email: hotmailuser@163.com

Received November 24th, 2008; revised November 30th, 2008; accepted December 1st, 2008.

ABSTRACT

A more automated graphic user interface (*GUI*) *test model, which is based on the event-flow graph, is proposed. In the model, a user interface automation API tool is first used to carry out reverse engineering for a GUI test sample so as to obtain the event-flow graph. Then two approaches are adopted to create GUI test sample cases. That is to say, an improved ant colony optimization* (*ACO*) *algorithm is employed to establish a sequence of testing cases in the course of the daily smoke test. The sequence goes through all object event points in the event-flow graph. On the other hand, the spanning tree obtained by deep breadth-first search* (*BFS*) *approach is utilized to obtain the testing cases from goal point to outset point in the course of the deep regression test. Finally, these cases are applied to test the new GUI. Moreover, according to the above-mentioned model, a corresponding prototype system based on Microsoft UI automation framework is developed, thus giving a more effective way to improve the GUI automation test in Windows OS.*

***Keywords*:** *Automated Software Testing, Graphic User Interface, Event-Flow Graph, Regression Testing, Ant Colony Optimization, UI Automation*

1. Introduction

Testing GUI is a hard and monotonous labor. So far, a large number of scholars and experts have been addressing themselves to the study of related fields. In the 1970s, some scholars suggested that testing software design be modeled by finite-state machines and testing software errors be found [1]. Thereafter another researchers applied the approach to the domain of testing GUI. It was called an improved model of finite-state machines, i.e. complete interaction sequence (CIS) [2]. After having come to recognize the fact that it increasingly did not satisfy the modeling requirements of GUI automation test, experts proposed an event-flow model based on event-flow graph. They investigated a variety of automatic generation approaches to GUI test cases, which were closely connected with the adopted GUI model like above-mentioned CIS. Besides, they simultaneously presented an algorithm to check the complete testing cases [4]. And an AI planning-based approach to GUI test was employed [5,6], which utilized the partial ordering planning in the field of AI Planning and attained test cases by the goal-driven method of searching state point. During the process of generating test cases by an AI planning-based approach, hierarchical GUI test case generation is derived [7]. In addition, other contribution like Memon and his colleagues at the University of Maryland are worth attention and they have made great progress in the theories of coverage criteria for GUI testing [8] and test oracles for GUI-based software applications [9−11]. In recent years, McMaster together with Memon presented call stack coverage for GUI test-suite reduction [12]. Moreover, AI and data mining have been applied to the relevant study of the deep regression test. Ye et al. investigated an approach to select a better way of the deep regression test by training neural network [13]. White suggested a method to use the mathematical model of Latin square to reduce case quantities [14]. Memon et al. put forward a proposal that the adaptability to software variation was improved through choosing event relationships in the deep regression test [15].

However, these approaches have not yet fully been put into practice in GUI automation test systems of industry fields for the time being, which are roughly classified into three categories: capture and replay mode, scripts-driven mode, and data-driven mode. There exists several distinct defects among them such as heavily depending on manual work, being characteristic of low adaptability to software variation, and lacking systematic management for testing cases and their coverage. Accordingly, in an effort to enhance the automation test, a more highly automated GUI testing model, which is based on the event-flow graphs, is proposed. In the model, an automation tool is first used to carry out reverse engineering for testing GUI sample so as to obtain the

event-flow graph. Then two approaches are adopted to create testing GUI sample cases. That is to say, an improved ACO algorithm is employed to establish a sequence of testing cases in the course of the daily smoke test. The sequence goes through all object event points in the event-flow graph. On the other hand, the spanning tree obtained by deep BFS approach is utilized to obtain the testing cases from goal point to outset point in the course of the deep regression test. Finally these cases are applied to test the new GUI. Moreover, according to the above-mentioned model, a corresponding prototype system based on Microsoft UI automation framework is developed, thus giving a more effective way of improving the GUI automation test in Windows OS.

Section 2 gives a brief description of GUI automation test model based on event-flow graph and also describes two types of algorithms of generating automation test cases. Section 3 depicts the development of a corresponding prototype system based on Microsoft UI automation framework. Finally, the conclusions and future work are given in Section 4.

2. A GUI Automation Test Model Based on Event-Flow Graph

In References [8, 9, 10, 11], Memon et al. presented an event-flow graph model when deeply studying the coverage criteria for GUI testing, whose purpose was to describe the mutual relationship among the object events more clearly. Thus a model, which was equipped with the most complete functions for GUI test, came into existence. But our event-flow graph model is obtained by simplifying the above model. It is actually a two-dimension vector $<V, E>$, where V denotes event sets in GUI and E represents order relationships of event execution in GUI. Their definitions are the same as the origin. In this model, non hierarchy modeling means neglecting the process of constructing components for GUI objects, thus enhancing the automation level for GUI test. What we have to do is to find out the GUI events which are executed immediately after previous events occur in terms of GUI states. In the course of reverse engineering, every GUI event has been gone through to discover the GUI events. Based on these GUI execution events, the vector event-flow graph is established. Then aiming at the requirements of GUI automation test, an improved ant colony optimization algorithm is employed to establish a sequence of testing cases in the course of the daily smoke test. In addition, the spanning tree obtained by deep BFS approach is utilized to obtain the testing cases from goal point to outset point in the course of the deep regression test. These cases are applied to test the new GUI. These algorithms are elaborated as follows.

The improved ACO algorithm for the daily smoke test suggested in the paper defines elicitation variables and a $tabu_k$ list and takes into consideration the consanguineous combination of a max-min ant system (MMAS), an ant colony algorithm based on an adaptive pheromone, and a type of rewards and penalty mechanism of pheromone volatilization. Its concrete formulae are concisely expressed below as subsection functions (1)–(3) and equations (4)–(6).

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{t\notin tabu_k}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}} & \text{if } j\notin tabu_k\\ 0 & \text{others}\end{cases} \tag{1}$$

$$j=\begin{cases}\arg\max_{j\notin tabu}\{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}\} & \text{if } q<q_0\\ J & \text{others}\end{cases} \tag{2}$$

$$\tau_{ij}(t+1)=\begin{cases}\rho\cdot\tau_{ij}(t)+\Delta\tau_{ij}(t) & \text{vector borders on optimal paths}\\ \rho\cdot\tau_{ij}(t)-\Delta\tau'_{ij}(t) & \text{vector borders on worst paths}\\ \rho\cdot\tau_{ij}(t) & \text{others}\end{cases} \tag{3}$$

$$\eta_{ij}=(\lambda_i+1)/(\omega_j+1) \tag{4}$$

$$\Delta\tau_{ij}(t)=Q/\mu \tag{5}$$

$$\Delta\tau'_{ij}(t)=\mu/Q' \tag{6}$$

where the number of crunodes is the rank, the number of ants is M. $\tau_{ij}(t)$ is pheromone density of vector border (i, j). η_{ij} is a elicitation variable which denotes the elicitation factor during the solution process. λ_i is the total number of crunodes which are not accessed from the crunode i while ω_j is the total number of crunodes which are accessed from the crunode j. α,β are corresponding to a pheromone elicitation factor and a self-elicitation factor. $tabu_k$ is an accessed crunode list when next crunode is searched. q is a stochastic variable of average distribution among [0,1] while q_0 is a given constant beforehand. ρ is the coefficient of pheromone volatilization. $\Delta\tau_{ij}(t),\Delta\tau'_{ij}(t)$ are pheromone increments. Q and Q' are both constants. μ is the number of repetitive crunodes, J is the result of subsection functions (1). At the beginning, initial pheromone density $\tau_{ij}(t)$ in the MMAS is equally set to maximum. When ant k moves from the crunode i at t, $P_{ij}^{k}(t)$ is the probability of choosing the crunode j.

According to MMAS, each pheromone density of vector border is situated in between $\tau_{\max}$ and $\tau_{\min}$ which are set in advance. If the value is bigger than $\tau_{\max}$, it is set to be equal to $\tau_{\max}$; Vice versa. Such disposal is beneficial to sufficient search and getting the optimal solution. Furthermore, if the goal crunode is not accessed and its λ is equal to 1, it should be preferentially considered when another goal crunode is selected. If the algorithm is convergent, the generated event crunode sequences are the desired GUI sample test cases for testing new GUI.

The algorithm based on the spanning tree obtained by deep breadth-first search (BFS) approach for the deep regression test is described as follows.

```
ALGORITHM: BFS(G,s){
      FOR ALL u ∈V[G]-{s} { /*the initial crunode*/
         color[u] = White;
       }
    color[s] = Gray;          /*deal with the initial crunode*/
    π[s] = Ø;
    Q = Ø;
Enqueue(Q, s);
    WHILE Q ≠ Ø {
       u←Dequeue(Q);
       FOR ALL v ∈Adj[u] {
          IF color[v] = White {          /*(u,v) is the tree border*/
                color[v] = Gray;
                π[v].Add(u,v);
                Enqueue(Q,v);
          }
       color[u] = Black;
    }
}
```

According to the theory of the spanning tree which shows simple path is corresponding to the shortest distance [16], GUI sample event cases can be gained as follows.

```
ALGORITHM:GetTestCaseOfEvent(Vertex v ∈ V){
    TestCase = Ø;
FOR ALL InEdge ∈ v.InEdges{
        u = BFSTree.Find(InEdge.SourceVertex)
        TestCase[u].Add(v);
TestCase[u].Add(u);
        WHILE u.Parent != StartVertex{
            TestCase[u].Add(u.Parent);
            u = u.Parent;
}
TestCase[u].Add(StartVertex);
    }
}
```

3. Developing the GUI Automation Test System

In the above-mentioned model, GUI hierarchy modeling is not taken into consideration and the process of components construction is neglected. Because GUI hierarchy modeling relies on the GUI logic relationships and needs manual operation, it inevitably influences the process of GUI automation test. Furthermore, with regard to GUI test case generation, an adaptive max-min ACO above based GUI test case generation algorithm is used for GUI daily smoke test, and a deep BFS based GUI test case generation algorithm is exploited for GUI deep regression test. The developing flow of GUI automation test system is shown below in Figure 1.

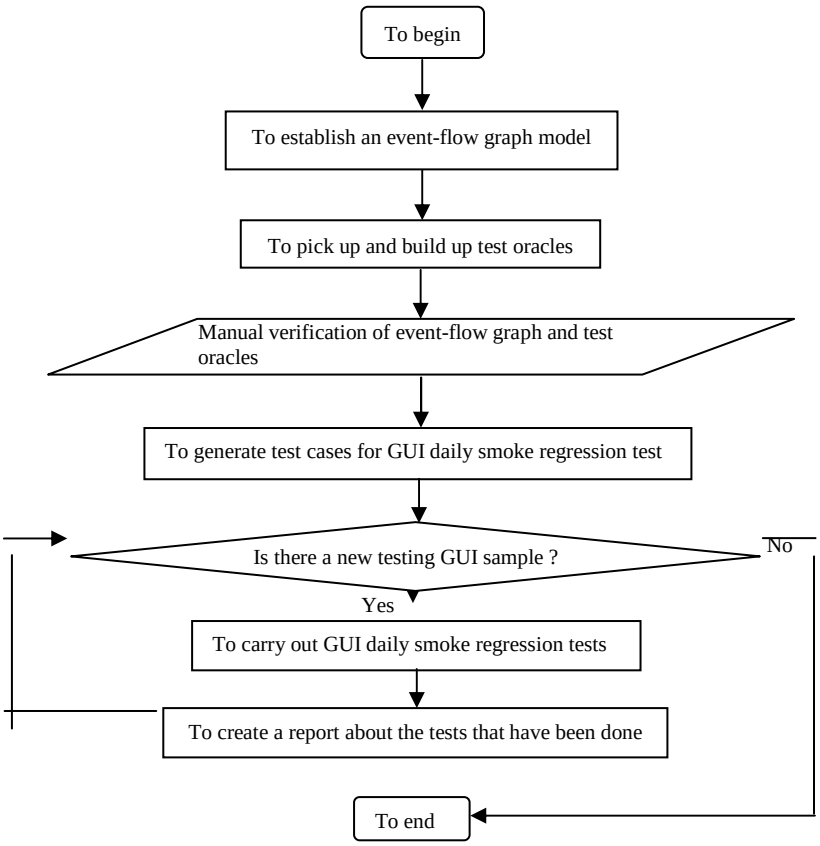

Figure 1. Developing flow of GUI automation test system

The GUI automation test prototype system is developed by taking advantage of Microsoft UI Automation frame, Visual Studio 2005, and advanced language C#. The Microsoft UI Automation frame can provide the developers with more uniform and convenient access to GUI in Windows OS than before. In the past, GUI automation operation usually requires indirect or direct usage of Windows API. Microsoft UI Automation acts as a part of Windows Presentation Foundation (WPF) in Windows SDK v6.0. It completely supports Windows Vista, Microsoft Windows XP and Windows Server 2003. It is deemed as a uniform access frame for the development of the systems based on WPF, standard Win32, Windows Form and Web UI.

The prototype system is divided into three main functional modules as follows. 1) one includes event-flow graph modeling based on reverse engineering and test oracles pick-up, 2) another one is for test case generation, 3) the last one is to finish testing execution and report. The output of three parts is documentary format so as to facilitate the interaction with each other and partially manual verification. Their interaction is presented in Figure 2. The hollow arrow points to the data flow direction. As Figure 2 shows, the sub-module of test oracles pick-up and another sub-module of event-flow graph modeling are used to acquire the relevant information from GUI sample, and then output test oracles and event-flow graph. Thereafter, partially manual verification module is also exploited to inquire about whether there are some faults about GUI objects or not. After the performance, test case generation module is transferred to generate test cases for GUI daily smoke regression test. Then these cases are used for testing new GUI. Finally, testing results are passed into test report module to work out an ultimate testing document.

The first module is the most difficult one in the system because Microsoft UI Automation frame is needed to perform a dynamic automatic analysis to GUI sample. The analysis is dynamic, that is to say, the GUI information is constantly changing and there exists a extremely complex relationship between the analytic tool and GUI. This module is based on reverse engineering of GUI event-flow graph. As a result, the documentary files about vector information in event-flow graph are obtained. Figure 3 shows the interface of test oracles pick-up sub-module and event-flow graph modeling sub-module.

In test case generation module, the documentary files above are called, and then are parsed to attain hash codes of crunodes and their vector borders, and establish a vector graph objects. The above mentioned GUI test case generation algorithms are utilized to generate test cases. In particular, the function of event-flow graph plotting is designed in this module. In the process, the generally professional plotting software Graphviz is used. Figure 4 shows the interface of GUI test case generation.

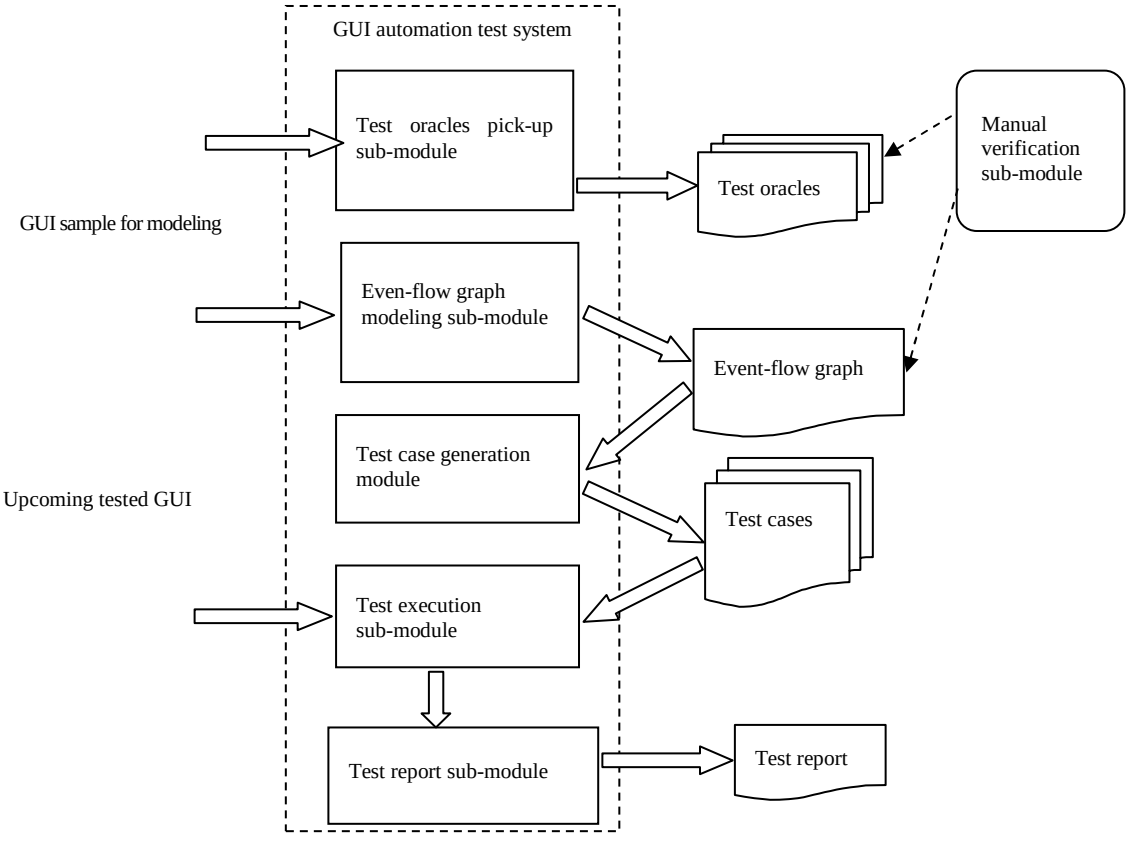

Figure 2. Interaction among three modules of GUI automation test system

In the last module of test execution and report, the required test event information can be obtained by the hash codes of new GUI. Microsoft UI Automation is used to acquire the controllers and their control modes of new GUI. The test types are selected and GUI daily smoke regression test are done. If the test is a daily smoke one, the test result is evaluated after each event is finished. If the test is a deep regression one, the test result is evaluated after the goal event is finished. Figure 5 shows the interface of GUI test execution and report.

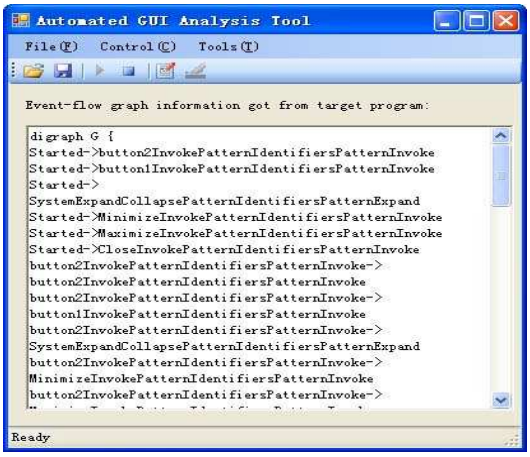

Figure 3. The interface of dealing with test oracles pick-up and event-flow graph modeling

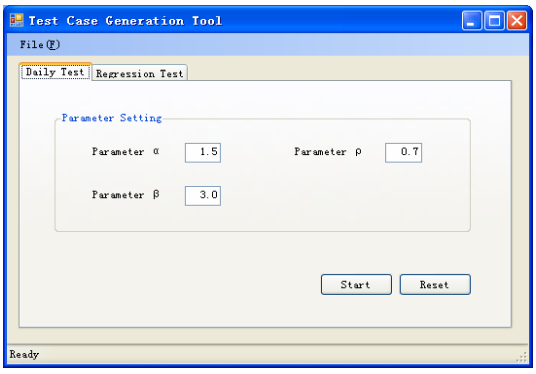

Figure 4. The interface of GUI test case generation

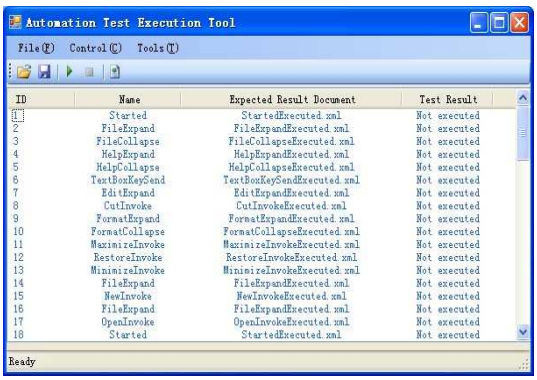

Figure 5. The interface of GUI test execution and report

4. Conclusions

Based on the event-flow graph modeling, a new GUI automation test model is presented. In the model, an improved ACO is put forward to generate test cases in the daily smoke test and a spanning tree is utilized to create test cases in the deep regression test. These test cases are generally applied in new GUI test. Moreover, a prototype system is developed on the basis of Microsoft UI Automation frame, thus giving a more effective way of improving the GUI automation test in Windows OS.

In the future, the systematic function test and contrast test with traditional GUI automation test software should be done in order to verify the validation of the model. And the adaptability of the studied system to the various GUI in other OS should be facilitated. In addition, the event-flow graph needs improving so as to solve the complex logic problem and reduce the involvement of manual verification.

5. Acknowledgement

The support from the Natural Science Foundation at Huazhong University of Science and Technology, the Natural Science Foundation in Hubei Province, and the National Natural Science Foundation in P. R. China, grant numbers 2007Q006B, 2006ABA085, 50775081, and 50675074 respectively, is gratefully acknowledged for this work by the authors.

REFERENCES

[1] T. Chow, “Testing Software Design Modeled by Finite-State Machines,” IEEE Transactions on Software Engineering, Vol. 4, No. 3, pp. 178-187, May 1978.

[2] L. White and H. Almezen, “Generating test cases for GUI responsibilities using complete interaction sequences,” in Proceedings of the International Symposium on Software Reliability Engineering, San Jose, California, USA, pp. 110−121, October 2000.

[3] A. M. Memon, “An event-flow model of GUI-based applications for testing,” Software Testing, Verification and Reliability, Vol. 17, No. 3, pp. 137−157, September 2007.

[4] L. White, H. Almezen, and N. Alzeidi, “User-based testing of GUI sequences and their interaction,” in Proceedings of the International Symposium on Software Reliability Engineering, Annapolis, Maryland, USA, pp. 54−63, November 2001.

[5] A. M. Memon, M. E. Pollack, and M. L. Soffa, “A planning-based approach to GUI testing,” in Proceedings of The 13th International Software/Internet Quality Week, San. Francisco, California, USA, May 2000.

[6] A. M. Memon, M. E. Pollack, and M. L. Soffa, “Plan Generation for GUI Testing,” in Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling, Menlo Park, California, USA, pp. 226−235, April 2000.

[7] A. M. Memon, M. E. Pollack, and M. L. Soffa, “Hierarchical GUI test case generation using automated

planning," IEEE Transactions on Software, Vol. 27, No. 2, pp. 144−155, May 2001.

[8] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for GUI testing," in Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, New York, USA, pp. 256−267, September 2001.

[9] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for GUI-based software applications," ACM Transactions on Software Engineering and Methodology, Vol. 16, No. 1, pp. 4−es, February 2007.

[10] A. M. Memon, M. E. Pollack, and M. L. Soffa, "Automated test oracles for GUIs," in Proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering: twenty-first century applications, San Diego, California, USA, pp. 30−39, November 2000.

[11] A. M. Memon, I. Banerjee, and A. Nagarajan, "What test Oracle should I use for effective GUI testing," in Proceedings of the IEEE International Conference on Automated Software Engineering, Montreal, Quebec, Canada, pp. 164−173, October 2003.

[12] S. McMaster and A. M. Memon, "Call stack coverage for GUI test-suite reduction," in Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering, Raleigh, North Carolina, USA, pp. 33−44, November 2006.

[13] M. Ye, B.Q. Feng, and Y. Lin, "Neural networks based test cases selection strategy for GUI testing", in Proceedings of the 6th World Congress on Intelligent Control and Automation, Dalian, China, pp. 5773−5776, June 2006.

[14] L. White, "Regression testing of GUI event interactions," in Proceedings of the International Conference on Software Maintenance, Monterey, California, USA, pp. 350−358, November 1996.

[15] A. M. Memon and M. L Soffa, "Regression testing of GUIs," in Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, New York, USA, pp. 118−127, September 2003.

[16] A. M. Memon, I. Banerjee, and A. Nagarajan, "GUI ripping: reverse Engineering of graphical user interfaces for testing," in Proceedings of the 10th Working Conference on Reverse Engineering, Victoria, B.C., Canada, pp. 260−269, November 2003.

