ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 254-261
doi: 10.18178/wcse.2017.06.044

A Grey-Box Approach to Getting GUI Test Model

Juncheng Chen [1+], Hua Wu[2] and Wenbo Zhang [1]

[1] BJUT Faculty of Information Technology, Beijing University of Technology Beijing, 100124, China
[2] Network and Educational Technology, Anyang Normal University, Anyang, 455000, China

Abstract. Graphic User Interfaces(GUI) is an important component of modern software, and incorrect implementation of GUI will reduce the usability and reliability of the overall software application. Testing is an effective way to discover defects and increase the quality of GUI. Generation of test cases is an important phase of GUI testing, and it can be achieved in manual or automatic approach. Due to the high complexity of GUI, it is impractical to generate test cases manually. So, it is a reasonable way to generate test cases from a GUI model automatically. What is the GUI model and how to create the model are still difficult tasks. In this paper, we redefine the event, propose an event handler graph(EHG) model which is considered as a GUI model, and put forward a grey-box approach which combines the dynamic method and static analysis of source code to create a GUI model automatically. The GUI model can be used as a basis for generating test cases automatically subsequently.

Keywords: GUI model; GUI Testing; grey-box;

1. Introduction

GUI(Graphic User Interface) is a very important part of modern software. On one hand, GUI provides users an agile way which make users facilitate operations and functionality of software quickly. On the other hand, GUI makes software testing more hard, and manual GUI testing is a very cost performance activity.

It is necessary to automate GUI testing for the high complex modern software. Model-based testing methods and related techniques are applied in GUI testing gradually. What kind of model should be applied and how to get the model are two prime problems in automated GUI testing.

In 2000, Lee White etc proposed a concept named complete interaction sequence(CIS), which is a finite state machine(FSM) and constructed by tester manually[1][2]. In order to reduce the scale of generating test cases, hierarchy finite state machine(HFSM) was put forward by Jessica Chen etc[3][4], and it is still created in a manually way.

Due to the state explosion of application under test, Atif M.Memon proposed event-flow model(EFM)[5], in which each event is triggered by one operation. The event-flow model can be obtained automatically or semi-automatically by reverse engineering[6]. Based on the event-flow model, many test coverage criterion were put forward and many significant experiments were carried out[7-12].

UML is another useful approach applied in GUI testing. With UML use case and activity diagram, GUI functionalities which should be tested are described, test cases are generated by the UML use case and activity diagram[13]. These UML models are created by designers during the design phase.

Above three models are applied in GUI testing in a blackbox way, and all the models are created by designing documents or related manual in a manual way or automatic/semiautomatic way.

If the source code of AUT can be obtained, then, some testing techniques in traditional testing is effective for GUI testing. Svetoslav Ganov etc put forward a GUI testing framework based on symbolic

+ Corresponding author. Tel.: +86-010-67392871
E-mail address: juncheng@bjut.edu.cn.

execution[14][15]. In the framework, event handler functions and variables related GUI controls are resolved through analysis of source code, and test cases generated through symbolic execution and constraint solving. Based on the framework, Svetoslav Ganov proposed an event-listener graph(ELG) further, which is created by formal and semantic analysis of source code. Similar to the ELG[16], Lei Zhao, etc. proposed two test coverage criteria based on the relations among event handler functions, and the relations are defined by Def-Use pair. The result of experiments shows that the criteria help select more effective test cases in an accurate way[17].

Only applying black-box testing techniques in GUI Testing leads to a great deal of useless test cases and difficulty of maintaining test cases[8][9]. On the contrary, only using white-box testing techniques in GUI testing makes test verdict difficult due to be lack of external help. But if we combine these two methods, external help in black-box testing should be a very useful complement to white-box testing.

How to get GUI test model in an automatic/semiautomatic way is an important aspect for GUI testing. Literal [6] shows how to get EFM with reverse engineering, and its main task is to get structure information of GUI. Similar to Literal [6], Paiva, etc. proposed a semi-automatic approach to create GUI test model, with Spec# technique[18][19]. As to ELG, it is very difficult to analyze the semantics for GUI , and the authors do not provide the approach of getting ELG.

In this paper, we redefine event and Event Flow Graph(EFG), and propose a definition of event handler function and a model named Event Handler Function Graph(EHG), which is similar to the ELG mentioned above and used to describe event handler function and correlations among the functions. The EFG is used to describe use's behavior, however, the EHG not only describes user's behaviour, it also expresses some information of internal implementation. Through discovering connections between nodes in EFG and nodes in EHG, we provide a grey-box approach to getting EHG in an automatic way.

This paper is organized as follows: section I introduces the state of the art of GUI model and the approaches of creating GUI model; section II proposes the redefinition of GUI event and the GUI test model; section III puts forward a grey-box approach to creating GUI test model, and we will explain the detail of the process of how to creating the test model and some related algorithms will be introduced in this section; the initial experiment will be presented in section IV; and the last section presents some conclusions and future work.

2. GUI TEST MODEL

In order to create GUI test model automatically, this section redefines event and event flow graph based on the definition of Atif Memon[5] from user's perspective, then defines Definition, Use, and Def-Use Pair, and introduces the definition of event handler function and event handler function graph from programmer's viewpoint at last.

2.1. Event and Event Flow Graph

Each event represents an user's operation, and it can be described by a five-tuple as the following:

evtName(control, operation, preEvt, postEvt, IOPair)

Each attribute in the definition is explained as follows:

- evtName is the name of the event, and it is unique in one application;
- control represents where the event is triggered;
- operation represents user's operation which may be input some text or click a button etc;
- preEvt is a set of event, in which all events can be triggered before this event;
- postEvt is also a set of event, all events in the set can be triggered after this event taking place.
- IOPair is regarded as test oracle, and it is a set of threetuple which has form (input, preStates, postStates) where input represents user's input for current event, preStates represents state of application before current event is triggered, and postStates represents the state of application after current event is triggered. This attribute is set by tester manually or by recording during the running of AUT.

Based on the definition of event, event flow graph is defined as follows

EFG(V, E)

For each application, there is an event flow graph, in which V is the set of all events, and E is the set of all relations among events. For example, if v1∈V; v2∈V , and (v1, v2) ∈E, then we can say that v2 can be triggered after v1 is executed. If all the events of one application have been obtained, we can create event flow graph automatically.

As a result of unknowing internal logic of AUT, the EFG can not express semantic of AUT accurately, so, many of test cases generated from the EFG model are useless and don't run at all. If we combine information of source code and EFG, a more accurate model will be created.

2.2. Definition and Use

In GUI applications, user's any effective input is received by the applications, and specified event handler function is executed. The event handler function takes user's input as input parameters, executes statements according program logic, and changes some related controls which will be showed in appearance. According to the attributes of programming language such as C#, Java, etc. GUI controls, which receive user's inputs and output the changes of appearance, are fields of class. At the same time, there are some internal fields which are not controls, but they influence controls or are influenced by controls. If we consider the control fields and non-control fields as variables, data flow testing technique will be applied in GUI testing.

For data flow testing technique [20], there are two important concepts, Definition and Use. Assuming that there is an event handler function ehf, and a field v is referenced in ehf. The Definition and Use is defined as the following: Definition: if v is used as a left value in a statement, we can say that v is regarded as a Definition variable in ehf. Use: if v is used as a right value in a statement or a parameter in an invoked function, v is considered as a Use variable in ehf.

The relation between two events is very important component for GUI testing. Based on the definition of **Definition** and **Use**, we define the relation named **Def-Use Pair** as the following:

Def-Use Pair: assuming that there are two events ehf_1, ehf_2, and a variable *v*, if v is a Definition variable in ehf1, and is a *Use* in ehf_2, then, we can say that (ehf_1, ehf_2) is a Def-Use pair about variable *v*, and the relation can be represented as the form (ehf_1, ehf_2;,v).

2.3. Event handler function and EHG

Based on the definition of *Definition*, *Use* and *Def-Use Pair*, we form the event handler function if the source code can be obtained, and get an accurate test model for GUI testing.

Event handler function achieves what to be executed and how to execute when an event is triggered, and it is completed by programmers. The form of event handler function is defined as the following:

EHName(control, operation, Def, Use, IOPair, preEH, postEH)

Each attribute in the definition is interpreted as follows:

- EHName is the name of event handler function, each event handler function has an unique name;
- control represents GUI control where event take place, it is the same as the definition in event;
- operation indicates which kind of operation take place on the control, and it usually is click, select, etc;
- Def is a set of all variables which are defined or reassigned in current event handler function, and its definition is the same as the definition in II-B;
- Use represents a set of all variables which are used in current event handler function, and its definition is also the same as the definition in II-B;
- IOPair is used to be test oracle, and its definition is the same as the definition in event;
- preEH is a set of event handler function, and each element of the set represents the predecessor of the current event handler function;
- postEH is also a set of event handler function, which represents the successor of the current event handler function.

If we connect all event handler function according to the above definition, the event handler function graph can be obtained. The definition of event handler function graph(EHG) is defined as the following:

EHG(V, E)

Through formal analysis of source code, we can't get complete event handler function because the correlation among event handler functions can't be interpreted without semantic analysis of source code which is a difficult task for GUI semantics.

3. The Process of Getting GUI Model

EHG is the kernel model for GUI testing, and our target is to create EHG automatically. The process of getting EHG is depicted as figure 1.

The process is divided into three phases, in the first phase, event flow graph is captured by executing the application under test (AUT), then incomplete event handler functions are obtained through analysis of source code in the second phase, in the last phase, EHG is created automatically through mapping the event flow graph to the incomplete event handler functions. This section will introduce three phases in detail.

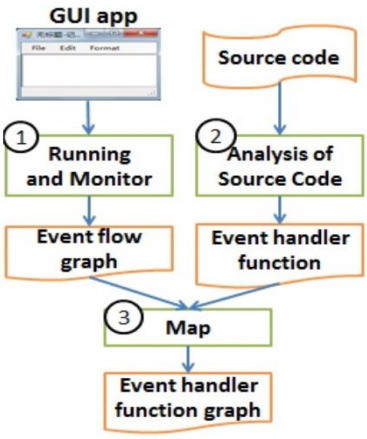


Fig.1: Process of Getting GUI Model

3.1. Phase 1: Getting Event Flow Graph

This phase is a dynamic exploration process, it runs the AUT, mocks user's operations on GUI controls and discovers the correlations among all the events. In the process, window is a basic unit. We will explore the relations among events in one window when the window is opened, and discover the correlations among events which are in different windows during the process. The algorithm of getting event flow graph is described in [5].

3.2. Phase 2: Getting Incomplete Event Handler Function

There are two categories of event handler function, one is provided by system library, and the other is implemented by programmers. The former one is usually considered as to be right in most situation, so we don't pay attention to the former and the latter is our consideration.

For source code of GUI applications, there are some attributes which is described as the following:

- The name of event handler function has stable pattern, which is like controlName operation, for example, the function *openMenuItem click*(*object sender*, *EventArg e*) which will be executed when menu *File* → *open* is triggered;
- All user inputs to the application must be through the GUI controls, so the control variables must be obtained in order to fill the *Def* or *Use* in event handler function defined in section II;
- There may be some non-control variables which may influence control variables or be influenced by control variables, therefore, these non-control variables should be added into *Def* or *Use* in event handler function.
- There may be some auxiliary functions which are invoked by event-handler functions. So the Def and Use in such functions should be analyzed.

The procedure of getting incomplete event handler function is divided into three phases which are described as the following steps:

- Step 1: get all variables named *SVariables* which are not temporal and all functions named *SFuncs*, and identify all control variables and all event handler functions named *SEhf*.
- Step 2: analyze all non-event handler functions line by line. Parameters of non-event handler function are

regarded as *Use*. During the analysis process, if a variable in *SVariables* is used as right value, then the variable is input to the current function's *Use*, otherwise, it should be put into *Def*. If a function in SFuncs is invoked, then both the invoked function's *Def* and *Use* are filled in the current function's *Def* and *Use*, respectively.

- Step 3: analyze all event handler function. The step is the same to the step 2.

There are two iterations in the procedure. In the first iteration we scans source code and gets all class fields which are variables and all functions, and in the second iteration, we analyzes each function's body line by line. Assuming that the number of class fields and functions are *N*, the running time of the first iteration is $O(N)$. For the second iteration, the running time is proportional to the number of lines of all functions' body. In summary, the procedure's complexity is decided by the scale of source code, and its a very efficient procedure.

3.3. Phase 3: Mapping

Based on the phase 1 and the phase 2, the event flow graph and the incomplete event handler function are obtained. Due to be lack of partial order of event handler function, we can't achieve the complete **EHG**. With the help of event flow graph, the incomplete event handler function can be completed and related **EHG** will be drawed out.

Algorithm 1 explains the phase of mapping. The algorithms 1 has two inputs, *efg* and *ehfs*, which represents event flow graph and event handler functions, respectively, and one output, *EHG*, which is our targeted model.

Algorithm 1 Mapping

```
Input:
 efg; ehfs;
Procedure:
 1: for all evt 2 efg and ehf 2 ehfs do
 2:   if evt.control = ehf.control and evt.operation =ehf:operation
        then
 3:       map = ConstuctMap(evt; ehf);
 4:   end if
 5: end for
 6: MarkNonEhfInEfg(efg);
 7: ConstructEhg(efg;map;EHG);
Output:
  EHG: event handler function graph
```

In algorithm 1, Line 1-5 create map between event in *efg* and event handler function in *ehfs* according to the event's attributes and the event handler function's attributes. Line 6 marks events which have no responsible event handler functions in *ehfs*. Line 7 is the key step in the algorithm, it refers to a concept of **adjacent** of event which is defined as the following:

Adjacent Assuming that $v_1 \in efg$; $v_2 \in efg$ and v_1 and v_2 have related event handler function in ehfs, if there is an event sequence (v_1, v_{11},… v_{1i} ,v_{1n}, v_2), $v_{1k} \in$ efg($1 \le k \le n$) and v_{1k} has no responsible event handler function in *ehfs*, then v_1 and v_2 are adjacent events.

Based on the definition of adjacent, Line 7 connects all events ,which have related event handler function, in *efg*, and copy the relation of the adjacent to the EHG according to the map obtained in line 3, then a complete EHG is achieved. In algorithm 1, Line 7 is the key step, and it traverses all nodes and all edges in efg. Assuming that the number of nodes and edges are *M* and *N*, respectively, the time complexity is $O(M + N)$. In worst situation, *N* and *M* is satisfied with $N = (M - 1)^M$. However, there are many dependencies among events for general application and no edges for many pair of nodes, so the size of *N* is much lower than the worst situation.

4. Initial Experiment and Discussion

This section introduces our prototype tool, initial experiment and discusses and analyzes the results of experiment and some threats and challenges.

4.1. Initial Experiment

The prototype tool was implemented using C# language and it can draw out EHG of winform applications automatically/semi-automatically. With the help of UI Automation[21], the tool mocks user's behavior, captures states of AUT, and generates event flow graph in an automatic/semi-automatic way. The event handler functions are obtained by analysis, which is achieved through Roslyn project [22], of source code in .Net solution.

In order to validate the approach, the tool was used to get EHG model of a Notepad application which is implemented using C# language and is similar to Microsoft Notepad.

This application is very simple, and it is not necessary to define extern condition. The application's appearance is depicted as figure 2. The application has an input area, which is a RichTextBox control. This control can accept two operations, which are input string and select some text. When the content is not empty, related menu such as Find, Find Next Replace, etc. can be triggered. This relation is finished in the first phase mentioned above.

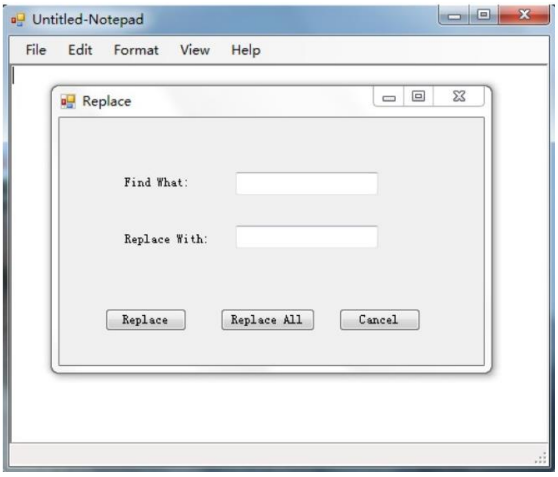
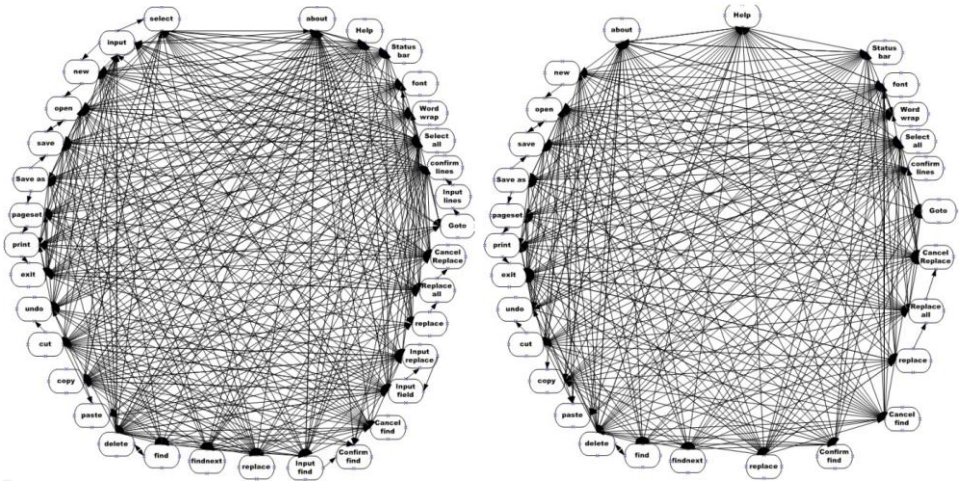


Fig. 2: Application's appearance

During the first phase of getting EHG model, the tool discovers two errors in the implementation. The two errors are raised by unexpected exception, the one is triggered when clicking Confirm button in the Goto dialogue but the line value is not digital string, and the other is raised when clicking Replace button or Replace All button but the Replace with control is set empty in Replace dialogue.

After correcting the errors, the tool generates an event flow graph, which is depicted as figure 3(a). Through analysis of the source code, we get 28 event handler functions, other 13 auxiliary functions, 50 control variables and 19 auxiliary variables. Combining event flow graph and event handler functions together, we generate an EHG showed as figure 3(b). The amount of nodes in EHG is six less than in EFG, and the amount of edges in EHG is 496 less than in EFG.



(a)event flow graph (b)event handler function graph

Fig.3: Event flow graph of Notepad

4.2. Discussion

In the above result, the correlations among the events is very complicate, because there is almost an edge for any pair of events. Though the number of nodes in EHG is only six less than in EFG, the number of edges in EHG is 496 less then EFG.

In addition to less nodes and less edges, nodes in EHG contain more information than nodes in EFG. For EFG, test coverage criteria is scheduled by the partial relations between events, however, we can define test coverage criteria according to partial relations and Def-Use pair, which has been proved to be effective to select test cases[17], for EHG.

There are two categories of controls in application from programmer's viewpoint, the first category of controls has related event handler function which is implemented by programmer, and the second one's functionality is provided by system or library. EHG only contains events handler function which is implemented by programmer, so, if AUT has more visitable controls which has no responsible event handler function, there will be more reduction in amounts of nodes and edges in EFG.

Due to some causes of techniques, our prototype tool faces the following challenges and threats:

- Not support non-standard control, because UI Automation library does not support non-standard control.
- Lack of testing of dialogue provided by system of library, such as *openFileDialogue* or *fontDialogue*. In our tool, we design a fixed pattern for each dialogue, for example, the open file is input and confirm button is clicked when the *openFileDialogue* is invoked.

5. Conclusion and Future Work

This paper redefines event and event flow graph, proposes a new GUI model named EHG, and gives the process of generating EHG model automatically in detail. We have finish the prototype tool and carried out an initial experiment. The result shows that the edges of EHG will be reduced in a large amount though only several nodes are deleted. Based on the EHG, the following topics will be developed in the future:

- **Test coverage criteria:** what test coverage criteria should be applied in EHG, and what is the relations among test coverage criteria.
- **Test oracle:** how to generate test oracle in a more automatic way. How to apply the approaches in traditional testing to create test oracle.
- **Test Maintaining:** how to maintain test cases based on EHG model during the process of developing software.
- **More Experiments:** select more AUT and carrying out more experiments to validate our approach.

6. References

[1] L. White, H. Almezen, and N. Alzeidi, "User-based testing of gui sequences and their interactions," in *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings*. 12th International Symposium on, nov. 2001, pp. 54 – 63.

[2] L. White and H. Almezen, "Generating test cases for gui responsibilities using complete interaction sequences," in *Software Reliability Engineering, 2000.* pp. 110 –121.

[3] J. Chen and S. Subramaniam, "A gui environment to manipulate fsms tor testing gui-based applications in java," in *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on, jan*. 2001, p. 10 pp.

[4] Chen, Jessica and Subramaniam, Suganthan, "Specificationbased testing for gui-based applications," *Software Quality Journal*, vol. 10, pp. 205–224, 2002, 10.1023/A:1021634422504.

[5] 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, 2007. [Online]. Available: http://dx.doi.org/10.1002/stvr.364

[6] A. 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, ser*. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 260–. [Online]. Available: http://dl.acm.org/citation.cfm?id=950792.951350

[7] 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, ser. ESEC/FSE-9. New York, NY, USA: ACM, 2001, pp. 256–267.

[8] A. Memon, I. Banerjee, and A. Nagarajan, “What test oracle should i use for effective gui testing?” in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on,* oct. 2003, pp. 164 – 173.

[9] A. M. Memon, “Automatically repairing event sequencebased gui test suites for regression testing,” *ACM Trans. Softw. Eng. Methodol*., vol. 18, no. 2, pp. 4:1–4:36, Nov. 2008.

[10] S. Huang, M. Cohen, and A. Memon, “Repairing gui test suites using a genetic algorithm,” in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on, april* 2010, pp. 245 –254.

[11] Q. Xie and A. Memon, “Model-based testing of communitydriven open-source gui applications,” in *Software Maintenance, 2006. ICSM ’06. 22nd IEEE International Conference on, sept*. 2006, pp. 145 –154.

[12] Qing Xie and Memon, A.M., “Studying the characteristics of a ”good” gui test suite,” in *Software Reliability Engineering, 2006. ISSRE ’06. 17th International Symposium on, nov*. 2006, pp. 159 –168.

[13] M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, and J. Kazmeier, “Automation of gui testing using a modeldriven approach,” in *Proceedings of the 2006 international workshop on Automation of software test, ser*. AST ’06. New York, NY, USA: ACM, 2006, pp. 9–14.

[14] S. R. Ganov, C. Killmar, S. Khurshid, and D. E. Perry, “Test generation for graphical user interfaces based on symbolic execution,” in P*roceedings of the 3rd international workshop on Automation of software test, ser*. AST ’08. New York, NY, USA: ACM, 2008, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/1370042.1370050

[15] S. Ganov, C. Kilmar, S. Khurshid, and D. Perry, “Barad c a gui testing framework based on symbolic execution,” 2009.

[16] S. Ganov, C. Killmar, S. Khurshid, and D. Perry, “Event listener analysis and symbolic execution for testing gui applications,” in *Formal Methods and Software Engineering, ser*. Lecture Notes in Computer Science, K. Breitman and A. Cavalcanti, Eds. Springer Berlin / Heidelberg, 2009, vol. 5885, pp. 69–87.

[17] L. Zhao and K.-Y. Cai, “Event handler-based coverage for gui testing,” in *Quality Software (QSIC), 2010 10th International Conference on, july* 2010, pp. 326 –331.

[18] A. Paiva, J. Faria, N. Tillmann, and R. Vidal, “A modelto-implementation mapping tool for automated model-based gui testing,” in *Formal Methods and Software Engineering, ser*. Lecture Notes in Computer Science, K.-K. Lau and R. Banach, Eds. Springer Berlin / Heidelberg, 2005, vol. 3785, pp. 450–464.

[19] A. Paiva, J. Faria, and P. Mendes, “Reverse engineered formal models for gui testing,” in *Formal Methods for Industrial Critical Systems, ser*. Lecture Notes in Computer Science, S. Leue and P. Merino, Eds. Springer Berlin / Heidelberg, 2008, vol. 4916, pp. 218–233.

[20] K. Naik and F. Tripathy, *Software Testing and Quality Assurance*. A John Wiley and Sons, Inc., 2008.

[21] Microsoft, “Ui automation overview,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms747327.aspx

[22] Microsoft, “Microsoft roslyn ctp,” Nov. 2011.