ZANCO Journal of Pure and Applied Sciences
The official scientific journal of Salahaddin University-Erbil
https://zancojournals.su.edu.krd/index.php/JPAS

ISSN (print):2218-0230**, ISSN (online):** 2412-3986**, DOI: http://dx.doi.org/10.21271/zjpas**

RESEARCH PAPER

GENERATING MODELS OF SOFTWARE SYSTEMS DURING EXPLORATORY

Akreen M. Saleh[1], Moayad Y. Potrus[2]

[1]Department of Software and Informatics, College of Engineering, Salahaddin University-Erbil, Kurdistan Region, Iraq
[2]Department of Software and Informatics, College of Engineering, Salahaddin University-Erbil, Kurdistan Region, Iraq

A B S T R A C T:

One of the major parts of the software testing is event-driven software (EDS), all actions of the software come from events. Interaction of the user to the graphic user interface (GUI) of the web and desktop applications will generate events, or for the embedded systems events and signals received from equipment, these are examples of Event-Driven Software. using EDS for software testing gives to the software tester a great result to test software because it generates a large number of events that could be cover most of the EDS's area. In this paper, an automated full-model and sub-model generation have been introduced during the system under testing, it produces test cases of websites to overcome faults and long time-consuming. The stage of the testing procedure includes generating full-model of the websites then extracting sub-model from the full-model in the next stages, test cases generated with path coverage. The proposed testing procedure has been analyzed with the four case studies consisting of Fault Detection and Fault Detection Effectiveness. Has been testing with a manual testing method and it proved its efficiency regarding test generation and time. Further, the sub-model test generation provides more accurate test case suite generation than full-model testing.

KEY WORDS: MBT, Graph theory, Software testing, full-model automation, Sub-model extraction
DOI: http://dx.doi.org/10.21271/ZJPAS.32.4.2
ZJPAS (2020) , 32(4);12-21 .

1.INTRODUCTION

The recent software development market is characterized by the increasing complexity of implemented systems, a decline in the time to market, and a demand for real-time operation of these systems on various platforms. One of the most important software development applications is a website, Websites are client-side software applications and accessed through browsers, mostly consists of (HTML) HyperText Markup Language pages it might be static and simple (Utting and Legeard, 2007).

Testing techniques are usually classified as black-box and white-box. Black-box (functional testing) depends on knowledge requirements and the client needs to determine the test cases. White box testing means structural test or interior testing, based on internal code structure and depends on the programmer's skill, this testing is usually done at the unit level. One of the best techniques that can be applied for testing efficiency and software quality is Model-Based Testing (MBT). According to Mark Utting and Bruno Legeard (Utting and Legeard, 2007), MBT allows the automatic generation of test cases through a model built based on the expected behaviour of the software under test (SUT).

* **Corresponding Author:**
Akreen M. Saleh
E-mail: akreen.muhialdeen@su.edu.krd or agrin.muhyeadin@gmail.com
Article History:
Received: 15/10/2019
Accepted: 15/02/2020
Published: 08/09 /2020

MBT is an approach that has several advantages reported in the literature, such as the automatic test case generation, fault detection effectiveness, and reduction in time and cost for testing (Utting and Legeard, 2007).

In this research, an algorithm proposed to generate automatically full-model of the website under testing and sub-model extraction, that helps the tester to generate automatic test case suite. The test case suite generated from the smaller models extracted from the full-model. In the empirical evaluation find out that the proposed algorithms generated better test cases suite and less time-consuming against manually exploration strategy and full-model compared with sub-model of the proposed algorithm, it is clearly seen that the testers easily can be managed sub-models for generating .a set of test cases and more accurate test cases suite will be generated.

There are several sources for problems in the website testing area. The main problem is manually generating test cases, it causes a decline in system release time in the system under testing (SUT). Change a requirement from the website tester must be re-generate test cases manually.

This paper proposes to evaluate the use of the model-based testing MBT concepts in the design and execution of automated tests in websites and using mutation testing to evaluate the efficiency of automated test.

2. BACKGROUND

To face faults in software testing in development, there are a large number of techniques that can be applied (Myers et al., 2012). Among them techniques defined for the automated test case generation using behavioral or structural model, also called a test model of the system under testing (SUT). This approach is known as Model-Based Testing (MBT) (De Cleva Farto and Endo, 2015). Model-based testing depends on three key technologies: 1. notation used for the data model, 2. test-generation algorithm, 3. tools to generate tests. Unlike the generation of test infrastructure, model notations and test-generation algorithms are portable across(Dalal et al.). MBT divided into four main steps: 1. modeling, 2. test generation, 2. Concretization 4. test execution (El-Far and Whittaker, 2002).

In modelling, the tester uses her/his knowledge to build a test model of the system under testing. the requirements are source of the information for the functionality of software being tested. An operating system, competing solutions, libraries and other specifications are factors that software product in an environment presented. The tester should be learning and understand the system under testing and test execution. It is advised to build test models based on the requirements, to maximize the independence between the model and the system under testing (Utting et al., 2012), to know and building a test model software analysis and design can be used. The test case generation algorithm based on the technique used to define the test model.

Modelling techniques must own features to create test case generation cheaper and easy automation (El-Far and Whittaker, 2002). For the automatic test case generation, a tool has been used. The test model has submitted as input then a set of test cases generated from test selection criterion and tool. during the system under testing, generated test cases not executable and at the abstract level. Finally to convert test cases from abstract level to the executable level in the system under testing concretization will involve. Execution of the test cases in the system under testing comes after conversion from the abstract level into the executable test cases. After the execution process, the results are analyzed and corrective actions are taken. If the test model defines both input and output values an automatic check may be performed.

Event Sequence Graph "ESG", can be used to show precisely the requirements and functionality of the system under testing to build the test model, some modelling techniques. ESG used to build a test model and modelling techniques. It is assumed that the modelling technique adopted for the model-based testing "MBT" is formal (Hierons et al., 2009). In MBT there are several modelling techniques used, like Finite State Machines (Lee and Yannakakis, 1996), Labeled Transition Systems(Tretmans, 1996), and UML(Hierons et al., 2009).

In this paper, the ESG technique has been adopted because of its ease to express communications between events and the simplicity to learn the requirements and functionality of the system under testing. An ESG is a directed graph,

used to model interactions between the software events and consists of nodes that represent events while the edges are valid sequences of these events (Belli et al., 2006, Yuan et al., 2011).

3. Related Work

In this section, several existing works have presented a categorized survey of generating models during the testing process. These related works are categorized according to the different strategies they have chosen for ways to create their own test cases.

An event-flow graph consists of nodes which represent events and edges that connect two events. In EDS can show each event changes, e.g. changing the colour of an input on a page in a website (Memon, 2007).

Memon (Memon, 2007) has introduced a technique based on the event-flow graph. In this paper, a tool used named GUIRipper for dividing the application and then event flow graph generated this solution categorized as a semi-automatic method. Event flow graph compared with another graph such as Finite State Machine (FSM), Complete Interaction Sequence (CIS) (Li et al., 2007) and Genetic Model(Pargas et al., 1999). After dividing the application overlapped sections created (Li et al., 2007) and by using the GUIRipper enabled features on the page will detect between the problems.

Belli et al. (Belli et al., 2006) have proposed an Event Sequence Graph (ESG) to model the behaviour of GUI. A tool has been created named GATE tool used to create and run test cases. The tool can be work with the ESG matrix and user input.

Herbold et al. (Herbold et al.) have suggested usage-based testing for EDS. This approach has three layers, that used to find which functionality of the application is used by the user. In the first layer users, actions are registered. Then in the next layer, registered actions are converted. Finally, in the last layer, the usage profile is generated from the events.

Herbold (Herbold and Steffen, 2012) for test case generation has proposed three new strategies based on usage-based testing. The first strategy, test paths are picked with a high probability. However, the number of valid sequences will increase exponentially. The second strategy, to reduce the sequences number of event, the first strategy and the random walk technique is used. The third strategy is to provide more gain in selecting test paths, a heuristic greedy strategy uses. Besides, AutoQUEST platform has developed by Herbold and Harms (Herbold and Harms) for EDS testing. on the AutoQUEST platform, usage-based testing is implemented and many testing techniques have been implemented. AutoQUEST is to present a testing technique independent from platforms and this was one of the main goals

Tonella et al. (Tonella and Ricca, 2004) have introduced an extraction of the website model in dynamic analysis. On the website, HTML code generated by the server-side and user actions on input values is required. In this paper, ReWeb tool created the model, is presented as a Markov Chain with expectation values on the edges. The method is known as semi-automatic because of using a tool named TestWeb tool which is including a test generator and test executer.

In the test section test, the criterion can be selected by the tester. In this method, like (Herbold et al.), the model does not cover all functionalities of the website if the input values are not selected entirely by the user.

(Ahmed and Bures, 2019) present an automated approach for generating models of smart TV applications based on black-box reverse engineering. The approach explored the user interface of the TV apps by using a remote control device and a model constructed cumulatively. The approach is used as a black-box technique, a tool has been implemented called EvoCreeper. the model generated in runtime mode by exploring the state of the user interface, this step is done without any information of the internal structure of the app.

4. Methodology

In this section proposed algorithm will be explained for the automated full-model generation and the strategy of extracting a sub-models from the full-model explained that the extraction process is done by selecting a node as a first node and another node as a last node of the sub-model then the algorithm extracted the sub-model from full-model automatically and then the sub-models ready and node-event coverage testing applied to generated a set test cases for the website under testing.

4.1 . Model Generation and Sub-model Exclusion

In this section, new strategy has been presented of automatically generating models of websites during testing. The technologies used in the strategy was google chrome extension and javascript for model creation. Figure 1,2 and 3 example of the developed google chrome extension. The method rely on the black-box technique for graph generation and does not need information about the code part of the website. The website is modelled as a directed graph G=(N, E), where N is a set of nodes at least one node exist, $N \neq 0$, and E is a set of edges. E is a subset of N x N possibilities.

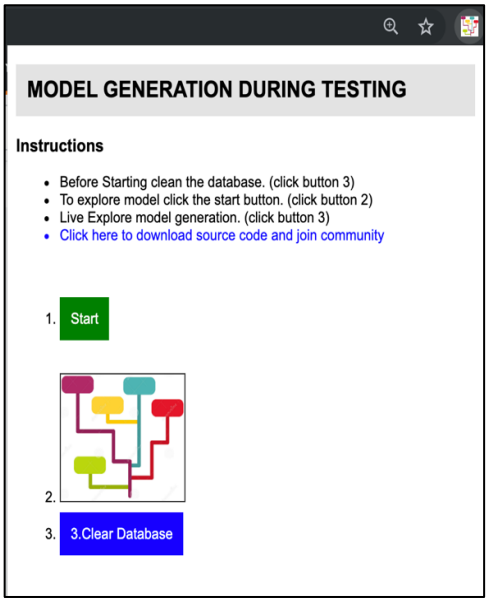


Figure 1: Proposed algorithm browser extension main view.

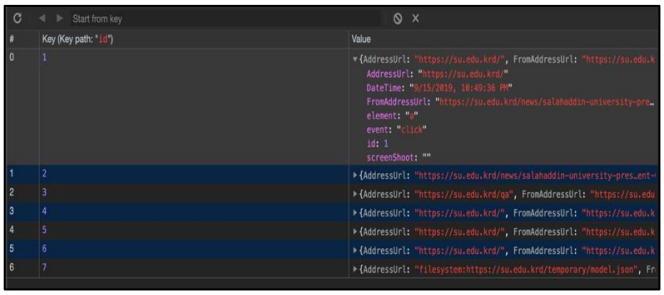


Figure 2: Proposed algorithm (node and event) database.

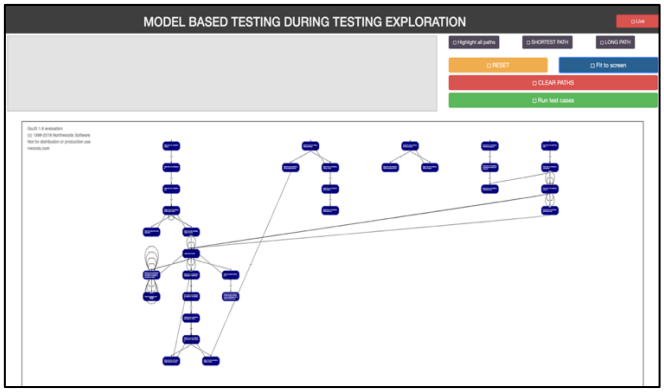


Figure 3: Extension Full-Model and Sub-Model Generator.

In the graph, one starting node defined, ns $\in$ N. The set Ne $\subseteq$ N contains the end nodes of the graph, where Ne $\neq$ 0. Each node corresponds to a UI element (state) of the website. Every edge communicates to possible events of the website. These events can be triggered by google chrome extension,

In this study, e $\in$ {key up, key down, keypress, blur, focus, click, double click, mouse hover, mouse down, navigation} for each e $\in$ E. To identify all the important states in the website user interface for exhibition in the graph, an algorithm developed to detect all events and navigations of the website, the algorithms called GUIRipper used to detect GUI elements that rip over the website that developed by HTML.

Algorithm 1 presents the levels of the Model Generator Algorithm. The algorithm detects testers all evens over GUI elements of the website to be tested, the events are edge E and the web page address converted to nodes of the model N. Start node Ns is that page the algorithm started, the algorithm first checks element type of the HTML tags of the web page and then detect events of each elements for instance form elements such as 'input, radio button, checkbox and select' for navigation checks anchor URL, when event detected the algorithm will add to the list of events e $\in$ E and page address add to the nodes N, the process will continue until the tester stops the algorithm by clicking the stop button in extension and end node of the model N algorithm stopped and there was nothing to test.

Algorithm 1 Model Generator Algorithm

```
node ←[]
link ←[]
referreAddress ←""
elementType ←getTagType
if elementType =anchor then
    referreAddress = anchorURL;
  if elementType = "input,radio,check box,select" then
    referreAddress = mainURL;
      for each event E do
          node ←[key: E.fromAddress, title: E.fromAddress]
          link ← [from : E.fromAddress, to : E.referreAddress]
      end for
```

4.2. An Automated Framework

According to the problems and challenges so far, an automated framework would propose to test websites. This framework shows our image for a strategy to automate the testing process. The framework working on the web browser which the nowadays browsers include an extension for developers and the framework developed for google chrome extension. Figure(4) reveals a summary of this framework and explains the fundamental components and their relationship to each other. The frameworks support only black-box testing the tester install the extension on google chrome web browser by clicking the start button in the extension, the testing algorithm started on the opened tab of the browser then algorithm detects testers event on the website, the events used in algorithm such as input events "key up, key down, keypress, blur, change, focus, on select, on submit, on reset", mouse events " mouse over, mouse out, mouse down, mouse up, mouse move" and click events "click, double click" detail of the algorithm will present in section 4.2.1.

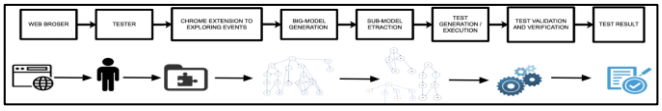


Figure 4: An automated framework of the proposed algorithm.

The algorithm uses a browser database to store logs, figure(3) shows a part of the full-model data in the browsers local database. then the testing process done tester stops the algorithm, then full-model will generate that contains nodes and edges. The tester will generate a sub-model from a full-model to generate test cases and validate test cases finally test results will be presented to the user.

4.2.1 Proposed Algorithm

To catch all the important events in the website to present in the model for test generation, an algorithm has been developed, that algorithm can be able to detect all web events during user tests the website.

Algorithm 1 shows the steps of the model generator algorithm. when tester hits the start button algorithm will be start and the current web page address is the main node of the model, the algorithm checks the type of the elements and it detects events to know at which element this event happens, another ability to detect navigation from the anchor tag. The address of the page will be adding to the list of the nodes and during detecting an event algorithm checks the address URL from and to the page then add to the link list to be model. Stopping condition by tester side after clicking the stop button of the algorithm stops. Full-model will be generated and ready to extract the sub-model then generate test cases.

4.2.2 Proof of Concept

In this section, the proposed algorithm will be illustrated as shown in algorithm 1. in this case a website developed that contains 10 nodes(pages) and 26 events(interactions) to implement the proposed algorithm. the algorithm implemented in google chrome extension, at first the tester must be installed on the browser and then the exploration process starts by clicking the start button in the program.

The website under testing will ready then tester start the exploration process and the proposed strategy will generate the full-model(graph) automatically during the exploration process, then the exploration process continued until all nodes explored.

in this stage the automated full-model ready to break into smaller parts called sub-model. sub-model is a part of full-model, the extraction process of sub-model from full-model will be done by selecting the first node as a start node Ns and another node as a last node of Nl and then the sub-model will be ready. Sub-model that will help the tester to be more accurate and less time losing during generating a set of test cases against full-model.

5. Experiential Evaluation

For evaluating the power of the model generation and test case generation strategy. Four case study of the website online chosen, during the evaluation process we address the following research questions.

- *Research question 1(RESQ1): Is the proposed strategy able to explore and detect nodes(pages) and events(interactions) of the website under testing?*
- *Research question 2(RESQ2): Is an automated full-model and sub-models are valid, and it complete model(graph) in term of number of nodes(pages) and events(interactions)?*
- *Research question 3(RESQ3): What is the performance of the algorithm to detect events of manual exploration for chosen websites?*

Research in the area of exploration test cases on the websites is very difficult. A long-time process may be needed for the testers to generate models and extract test cases, as it is new in software testing. As a result, not many websites and repositories are available for benchmarking. Most developers develop websites in different techniques and it is hard for the tester to test the website during development. However, there are some tools for generating models after completing the system.

Four different websites chosen of different sizes online, to illustrate the effectiveness of our approach. These websites are from different areas and have differing numbers of functionalities. Table 1 reveals the title and address of websites.

A set of experiments has been handled to address RESQ1 and RESQ3. The purpose was to differentiate the models and test cases of the websites created through manual exploration. To test our strategy against manual testing, We guided students from software engineering study program, we divided students into four groups of 30 participants.

Table (1) Case Studies for the implementing proposed algorithm.

#	Title	Link
1	Test Case study website	github.com/agreensaleh/mbt.git
2	Salahaddin university website	https://su.edu.krd
3	Webmail Horde	http://demo.horde.org
4	CPanel for web hosting	https://cpanel.net

The extension installed on laptop or desktop computers on google chrome browser, algorithm stores the target events, states and timestamps of the exploration. Every student in each group of the website testing has been assigned to explore and export the database of exploration logs. The finishing of the testing was the student decides all parts of the website to have been explored. Then the algorithm prepared events to generate models during testing then automatically full-model generated. Several attributes have been analyzed for each model, like as the time required to build the model, the number of nodes in the model and the unique nodes in the model. The proposed modelling, for generating test cases it will extract sub-model from full-model then it will be easy to generate test cases.

To address RESQ2, the proposed model generation was compared against other existed model generating algorithm such as (Ahmed and Bures, 2019). The requirements of the generated model at least have one node and the model must not be empty, should have its starting node, and all nodes of the model must have at least one incoming edge and except the starting node. additionally, in the model, at least one end node must be presented. the generated model must be able to reach from any node to the end node, finally, every node must be reachable from the main node

Figure 5 shows the produced directed full-model graph and sub-model of the test case study website. The nodes represent the pages and the arrows(edges) represent the events. The website consists of 10 nodes and derives by 87 Events as shown partially in figure 5. Figure 8 and 9 show the box plot of the results for the comparison between the algorithm and manual generation. Due to the personal and nondeterministic nature of the outcomes gotten by each student, and to ensure reasonable comparisons with more details, box plots used to compare the result.

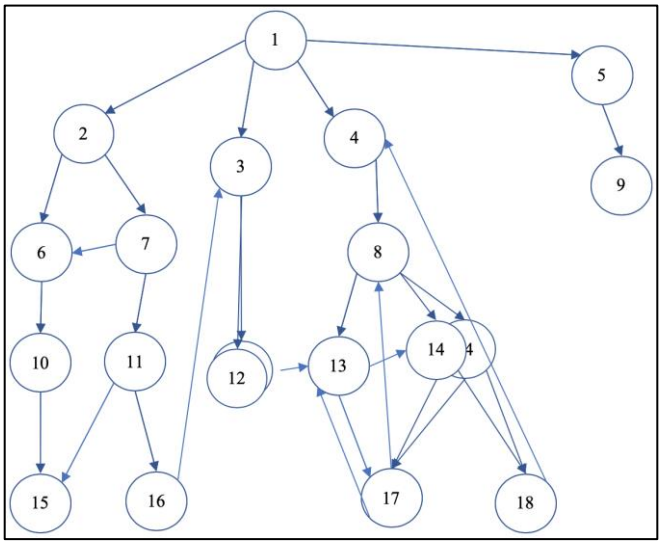


Figure 5: Full-model a part of test case study.

Figure 6 and 7 show the process of applying the proposed sub-modeling technique on the main website. As it can be seen from figure 7, the full-model has been divided into three sub-models in which each consist of 8,5 and 6 nodes respectively. Moreover, these graphs consist of 4,4 and 3 events shown in table 2.

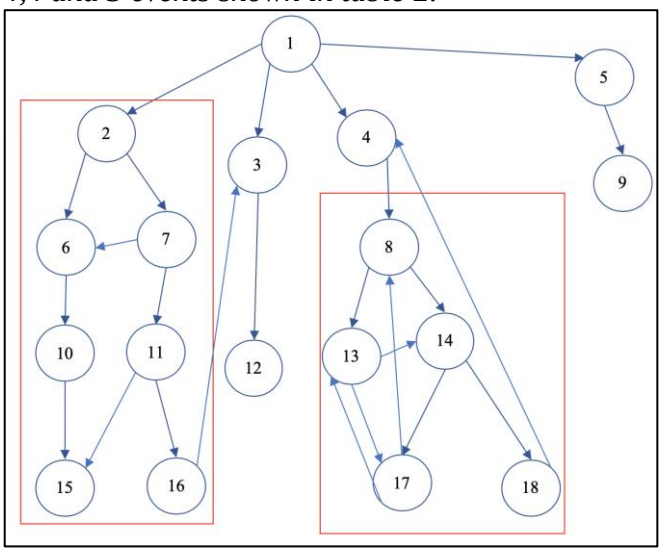
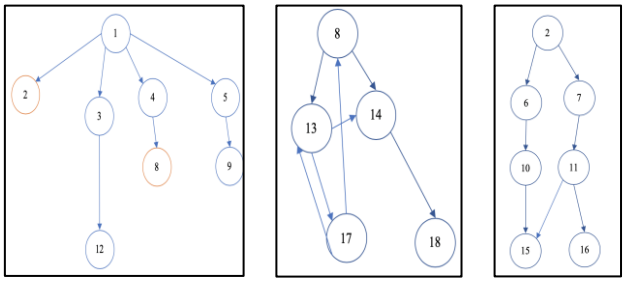


Figure 6: Extracting sub-models from full-model.



a) sub-model 1 b) Sub-Model 2 c) Sub-Model 3

Figure 7: Sub-models extracted from the automated full-model(graph).

Table (2) Test case generation result.

Model	No. Nodes	Test Cases
Full-Model	18	468
	TOTAL	468
Sub-Model-1	8	30
Sub-Model-2	5	35
Sub-Model-3	7	49
	TOTAL	114

Figure 8a shows the box plot of algorithm exploration versus manual exploration of the graph of the website, the blue color operate as manual exploration and the orange color identified for the algorithm exploitation. It is clear seen that both exploration for detecting nodes of the website are the same, due to the small number of pages in website. Figure 9a and table 2 show the difference between exploring time for manual and algorithm exploration, the exploring time for the manual exploration may vary from each participant's, generating the full graph of the website, exploring time will change from 15-20 minutes, algorithm generate the full graph during website exploration no need time to generating full graph. The completion of manual exploration depends on the participant's understanding and experience with the website. While the strategy explored graph without need to write pages and derive events of the website, the participant's had to write pages and events then generate graphs.

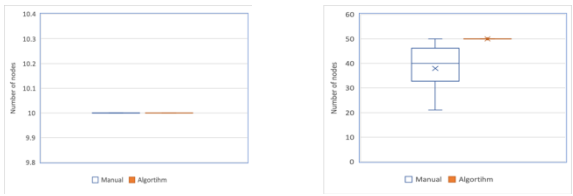
Figure 8b shows the result of the su website, all pages detect by the both exploration is less than 50 pages, manual exploring vary from each participant's, exactly half of the participant's detect pages between 32 and 46 pages, at least 25% of the participant's between 21 and 31 pages. Exploring time shown in figure 9b and table 3, most of the participant's generate

the full graph between 52 and 58 minutes, while the rest of them more than 58 and less than 110 minutes.

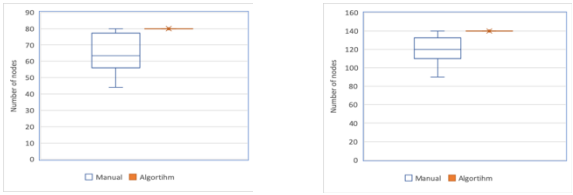
Figure 8c and 8c shows the manual exploration result of the webmail and CPanel websites. It can be seen that in table 5 shows page comparison of the webmail, participant's explore pages between 44 and 80 pages and the time exploration time exploration between 60 and 146 minutes, table 6 shows CPanel page exploration that 90 and 140 pages explored and the time exploration time exploration between 130 and 170 minute.

Table (3) No of pages and time exploration of the manual against proposed algorithm.

Case study	Time explored (Manual)	No. of pages explored (Manual)	Time explored (Algorithm)	No. of pages explored (Algorithm)
Case study website	15-20 minutes	10 pages	0 minutes	10 pages
Su website	33-110 minutes	12-50 pages	0 minutes	50 pages
Webmail website	55-150 minutes	44-80 pages	0 minutes	80 pages
CPanel website	130-170 minutes	90-140 pages	0 minutes	140 pages

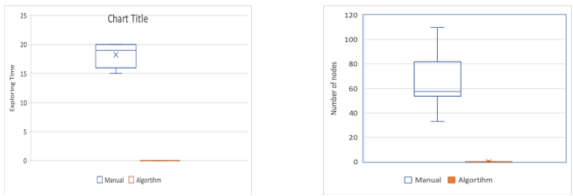


a) Test case website b) Su Website

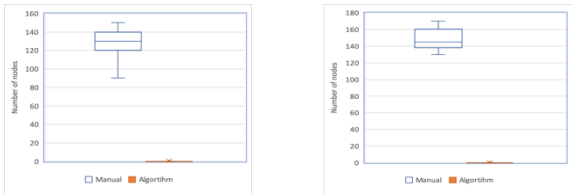


c) Webmail web application d) CPanel web application

Figure 8: Comparing the number of unique nodes detected by algorithm exploration with the manual exploration.



a) Test case website b) Su Website



c) Webmail web application d) CPanel web application

Figure 9: Exploration Time comparisons between Algorithm and Manual exploration.

One of the main features of the website is that it can grow dynamically. This means pages can be add or links can be increased. This will make the regression testing more time consuming when a node is added. the subgraph (model) will help generate test cases that are only relayed to the affected graphs(models) instead of the full model. This is one of the advantages of using this technique.

5. CONCLUSIONS

In this paper, the strategy for automatically reverse engineering websites presented by using the chrome extension. Events explored from websites by involving navigation in GUI, this happens without knowing the internal structure of the website "it is an example of the black-box". The directed graph model generated after extensively exploring events and states in a given website. The strategy implemented in the google chrome extension that works in a web browser or laptop and personal computers.

To evaluate the strategy, two medium-sized and two large websites used. The effectiveness and performance of our strategy demonstrated from the evaluation results. It can be seen that the proposed strategy generated test cases less than automated full-model and less-time losing during generating set test cases.

To generate test cases node-event coverage testing strategy was implemented and in this strategy nodes(pages) N and events(interactions) E will be multiplied which is N X E, as a result, the number of test cases generated for the full-model was 18 X 26 = 468 test cases and for all sub-models is 114 test cases, the number of test cases of the sub-model is less than full-model and the test cases is more accurate and more efficient.

The strategy can be used to detect events during testing and models automatically generated tester easily can be tracked generated model. The generated models have an advantage in stages of developing a website in term of quality enhancement, software testing, finding disappeared requirements, evaluating the user experience and provided virtualization of the website for an understanding the events and states.

There are several possibilities for future research. A quick step forward is to use the strategy-generated models to test numerous applications and identify new errors. as well, for automatically traverse on all website events and stated we are planning to use this strategy.

Acknowledgements

First and above all, I would like to thank God Almighty for providing me with this opportunity and granting me strength, knowledge, and capability to undertake this research study and to proceed successfully. Without his blessings, this achievement would not have been possible.

I would like to express my deep gratitude to my supervisor Asst. Prof. Dr. Moayad Yousif Potrus for his patience, enthusiasm, insightful comments, and immense knowledge. His guidance helped me in all the time of research and writing of this paper.

References

AHMED, B. S. & BURES, M. 2019. EvoCreeper: Automated Black-Box Model Generation for Smart TV Applications. IEEE Transactions on Consumer Electronics, 65, 160-169.

BELLI, F., BUDNIK, C. J. & WHITE, L. 2006. Event-based modelling, analysis and testing of user interactions: approach and case study. Software Testing, Verification and Reliability, 16, 3-32.

DALAL, S. R., JAIN, A., KARUNANITHI, N., LEATON, J. M., LOTT, C. M., PATTON, G. C. & HOROWITZ, B. M. Model-based testing in practice. ACM, 285-294.

DE CLEVA FARTO, G. & ENDO, A. T. 2015. Evaluating the model-based testing approach in the context of mobile applications. Electronic Notes in Theoretical Computer Science, 314, 3-21.

DEMILLO, R. A., LIPTON, R. J. & SAYWARD, F. G. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. Computer, 11, 34-41.

EL-FAR, I. K. & WHITTAKER, J. A. 2002. Model-Based Software Testing. Hoboken, NJ, USA: John Wiley & Sons, Inc.

HERBOLD, S., GRABOWSKI, J. & WAACK, S. A Model for Usage-Based Testing of Event-Driven Software. 2011/06//. IEEE, 172-178.

HERBOLD, S. & HARMS, P. AutoQUEST -- Automated Quality Engineering of Event-Driven Software. 2013/03//. IEEE, 134-139.

HERBOLD, S. & STEFFEN. 2012. Usage-based Testing of Event-driven Software.

HIERONS, R. M., KRAUSE, P., LÜTTGEN, G., SIMONS, A. J. H., VILKOMIR, S., WOODWARD, M. R., ZEDAN, H., BOGDANOV, K., BOWEN, J. P., CLEAVELAND, R., DERRICK, J., DICK, J., GHEORGHE, M., HARMAN, M. & KAPOOR, K. 2009. Using formal specifications to support testing. ACM Computing Surveys, 41, 1-76.

LEE, D. & YANNAKAKIS, M. 1996. Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE, 84, 1090-1123.

LI, P., HUYNH, T., REFORMAT, M. & MILLER, J. 2007. A practical approach to testing GUI systems. Empirical Software Engineering, 12, 331-357.

MA, Y. S., KWON, Y. R. & OFFUTT, J. 2002. Inter-class mutation operators for Java. Proceedings - International Symposium on Software Reliability Engineering, ISSRE, 2002-Janua, 352-363.

MEMON, A. M. 2007. An event-flow model of GUI-based applications for testing. Software Testing, Verification and Reliability, 17, 137-157.

MYERS, G. J., SANDLER, C. & BADGETT, T. 2012. The art of software testing, John Wiley & Sons.

PARGAS, R. P., PARGAS, R. P., HARROLD, M. J. & PECK, R. R. 1999. Test-Data Generation Using Genetic Algorithms. SOFTWARE TESTING, VERIFICATION AND RELIABILITY, 9, 263--282.

TONELLA, P. & RICCA, F. 2004. Statistical testing of Web applications. Journal of Software Maintenance and Evolution: Research and Practice, 16, 103-127.

TRETMANS, J. 1996. Conformance testing with labelled transition systems: Implementation relations and test generation. 29, 49-79.

UTTING, M. & LEGEARD, B. 2007. Practical Model-Based Testing.

UTTING, M., PRETSCHNER, A. & LEGEARD, B. 2012. A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability, 22, 297-312.

VOAS, J. M. & M, J. 1992. PIE: a dynamic failure-based technique. IEEE Transactions on Software Engineering, 18, 717-727.

YUAN, X., COHEN, M. B. & MEMON, A. M. 2011. GUI Interaction Testing: Incorporating Event Context. IEEE Transactions on Software Engineering, 37, 559-574.

POTRUS, M. Y. 2016. Maintenance Scheduling Optimization for Electrical Grid Using Binary Gray Wolf Optimization Technique. ZANCO Journal of Pure and Applied Sciences.

KANAR SHUKR MOHAMMED, M. Y. P. B. F. A. D. 2018. Effect of Hybrid Teaching Methodology and Student Group Policy on Object Oriented Problem Solving. ZANCO JOURNAL OF PURE AND APPLIED SCIENCES, 30.