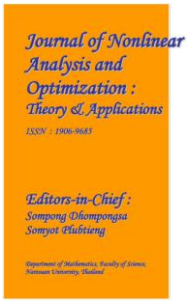
Journal of **N**onlinear **A**nalysis and **O**ptimization
Vol. 16, Issue. 1: 2025
ISSN : **1906-9685**



VIRTUALIZED PAGE RECOVERYAND PERMISSION FOR WEBAPPLICATIONS USING GENETIC ALGORITHMS

[1]MD.FATHIMA TABASSUM [2]Dr. Y. CHITTI BABU

[1]M.Tech Scholar, Dept. of CSE, St. Ann's College of Engineering & Technology, Chirala

[2]Associate Professor, Dept of CSE, St. Ann's College of Engineering & Technology, Chirala

Email: mft1713@gmail.com

Abstract: Web system to helps users and administrators of web applications recover from intrusions such as SQL injection, cross-site scripting, and attacks, while preserving legitimate user changes..We propose many methods for automating results bottleneck finding using search-based input-sensitive application profiling. Our key idea is to use a genetic algorithm as a search heuristic for obtaining combinations of input functions values that maximizes number of function to represents the elapsed execution time of the application. We present TAO tool is a software testing tool result automated test and oracle generation based on a semantic model. TAO is worked grammar-based test generation with automated semantics evaluation using a denotation semantics framework. The quality of web application is a broad review of recent Web testing advances model and discuss their goals, targets, techniques employed, inputs/outputs and stopping criteria. Finally test case can be generated automatically by solving and modify the problem using evolutionary algorithm.This model is attractive because it take a suite of adaptive automated and semi-automated solutions in situations many large complex problem spaces with multiple competing and conflicting objectives.

Index Terms: Search-based Software Engineering, Evolutionary Algorithms, Optimization Problem, Evolutionary Testing, Online Information Services

,

1. INTRODUCTION

Search based optimization techniques have been applied to a number of software engineering activities[2] such as requirements engineering, project planning and cost estimation through testing, to automated maintenance, service-oriented software engineering, compiler

optimization and quality assessment. can be the optimization can be applied over the software engineering activity.[1] We propose a novel approach for automating performance bottleneck detection using search-based application profiling. Our key idea is to use a genetic algorithm (GA) as a search heuristic for obtaining combinations of input parameter values that maximizes a fitness function that guides the search process [4]. We implemented our approach, coined as Genetic Algorithm-driven Profiler (GA−Prof) that combines a search-based heuristic with contrast data mining [3] from execution traces to automatically and accurately determine bottleneck. The testing of web based applications has much in common with the testing of desktop systems like testing of functionality, configuration, and compatibility. Web application testing consists of the analysis of the web fault compared to the generic software faults. There are various non-equivalence issues between traditional software testing and web application testing. As web applications become more complex, testing web applications have also become complex. Performance testing is very important to improve reliability and feasibility of web applications for satisfying users. As web application usage is enormous, traditional testing technique is not suitable to solve the problem because of several difficulties as follows [5]

1) Some metrics need to be predicted such as, the type of users, the number of concurrent users, and access methods because of difficulties for simulating real scenarios.

2) Since performance is mostly related to user satisfaction, issues related to reactivity should be considered in performance testing

3) Performance testing and scalability are the focus of system testing, because a large number of users will access a service in one distributed web application concurrently.Evolutionary Algorithms are used to guide the search. The Fig.1 shows the structure and interaction of test activities including test case design by means of evolutionary algorithms.[6].

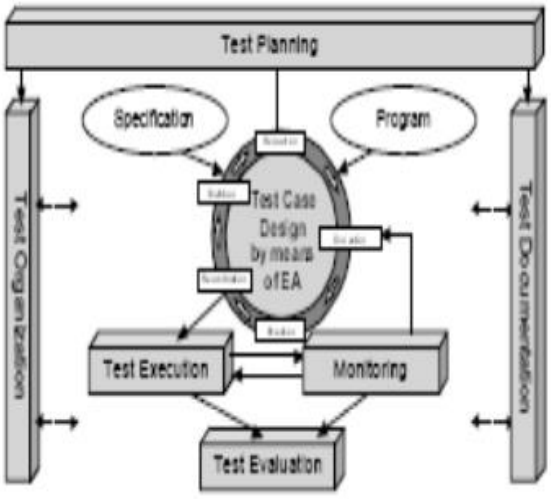


Fig. 1: Structure of evolutionary algorithms

We firstly introduce a declarative tool, named TAO, which performs automated test and oracle generation based on the methodology of denotational semantics [7]. TAO combines our previous work on a grammar-based test generator [8] and a semantics-based approach for test oracle generation [9], using a formal framework supporting the denotational semantics. Our framework incorporates grammar-based testing and semantics-based oracle generation into the Selenium web testing automation to generate an executable JUnit test suite. Selenium is an open source, robust set of tools that supports rapid development of test automation for Web-based applications.

2. RELATED WORK

The goal of web is to recover the integrity of a web application after it has been compromised by an adversary [10]. More specifically undo all changes made by the attacker to the system, including all indirect effects of the attacker's changes on legitimate actions of other users [11], and non-functional fault detection Korel provided an approach that generates test cases based on actual executions of AUT to search for the values of input variables, which influence undesirable execution flow, by using function minimization methods[12]. Artzi et al. used the Tarantula algorithm to localize source codes which lead to failures in web application by combining the concrete and symbolic execution information [9]. Chilimbi et al. provided a tool. Genetic Algorithms (GAs) is widely used in many areas of software engineering such as software maintenance textual analysis [13], cloud computing and testing Test generation is a key point in software testing. Alshahwan et al. used dynamically mined value seeding into search space to target branches and generate the test data automatically [4]. To achieve higher branch coverage, McMinn et al. used a hybrid global-local search algorithm, which extended the Genetic Algorithm with a Memetic algorithm, to generate the test cases Harman designed an approach by using the dynamic symbolic execution and search-based algorithms to generate test data, which can kill both the first order and higher order mutants for mutation testing [14]. Ali et al. provided a systematic review for the search-based test case generation, which built a framework to evaluate the empirical search-based test generation techniques by measuring cost and effectiveness.

Grammar-based test generation (GBTG) provides a systematic approach to producing test cases from a given context-free grammar. Unfortunately, naive GBTG is problematic due to the fact that exhaustive random test case production is

often explosive. Prior work on GBTG mainly relies on explicit annotational controls, such as production seeds [15], combinatorial control parameters [9], and extra-grammatical annotations [7]. GBTG with explicit annotational controls is not only a burden on users, but also causes unbalanced testing coverage, often failing to generate many corner cases. TAO takes a CFG as input, requires zero annotational control from users, and produces well-distributed test cases in a systematic way.

3. OPTIMIZATION TECHNIQUES

Some of the optimization techniques that have been successfully[4] applied to test data generation are Hill Climbing(HC) ,Simulated Annealing(SA), Genetic Algorithms(GAs),

Meta-heuristic Search Techniques Meta-heuristic techniques have also been applied to testing problems in a field known as Search Based Software Testing [2], [3], a sub-area of Search Based Software Engineering (SBSE) [1]. Evolutionary algorithms are one of the most popular meta-heuristic search algorithms and are widely used to solve a variety of problems. The local Search techniques generally used are

i. Hill Climbing
ii. Simulated Annealing
iii. Tabu Search

Hill Climbing In hill climbing, the search proceeds [16] from randomly chosen point by considering the neighbors of the point Once a neighbor is found to be fitter then this becomes the current point in the search space and the process is repeated. The search terminates and a maximum has been found HC is a simple technique which is easy to implement and robust in the software engineering applications of modularization and cost estimation.

Simulated Annealing Simulated annealing is a local search method. It samples the whole domain and improves the solution by recombination in some form. Cost functions define the relative and desirability of particular solutions. Minimizing the objective function is usually referred to as a cost function; whereas, maximizing is usually referred to as fitness function.

Tabu Search Tabu search is a met heuristic algorithm that can be used for solving combinatorial optimization problems, such as the travelling salesman problem (TSP). Tabu search uses a local or neighborhood search procedure to iteratively move from a solution x to a solution x' in the neighborhood of x, until some stopping criterion has been satisfied. To explore regions of the search space that

would be left unexplored by the local search procedure, Tabu search modifies the neighborhood structure of each solution as the search progresses.

Algorithm 1 GA–Prof's algorithm for automating application profiling

```
1: Inputs: GA Configuration Ω, Input Set I
2: P ← Initial Population(I)
3: while Terminate() == FALSE do
4:    P ← Crossover(P, Ω)
5:    P ← Mutation( P, Ω, I)
6:    for all p ∈ P do
7:       F ← FitnessFunction( p)
8:    end for
9:    P ← Selection( F, P)
10: end while
11: return P
```

Genetic Algorithms GA forms a method of adaptive search in the sense that they modify the data in order to optimize a fitness function. A search space is defined, and the GAS probe for the global optimum. A GA starts with guesses and attempts to improve the guesses by evolution. A GA will typically have five parts: (1) a representation of a guess called a chromosome, (2) an initial pool of chromosomes, (3) a fitness function, (4) a selection function and (5) a crossover operator and a mutation operator. A chromosome can be a binary string or a more elaborate data structure. The initial pool of chromosomes can be randomly produced or manually created. The fitness function measures the suitability of a chromosome to meet a specified objective: for coverage based ATG, a chromosome is fitter if it corresponds to greater coverage. Genetic programming results in a program, which gives the solution of a particular problem. The fitness function is defined in terms of how close the program comes to solving the problem. The operators for mutation and mating are defined in terms of the program's abstract syntax tree. Because these operators are applied to trees rather than sequences, Most of the work on Software Testing has concerned the problem of generating inputs that provide a test suite that meets a test adequacy criterion. The schematic representation Often this problem of generating test inputs is called 'Automated Test Data Generation (ATDG)' though, strictly speaking, without an oracle, only the input is generated..

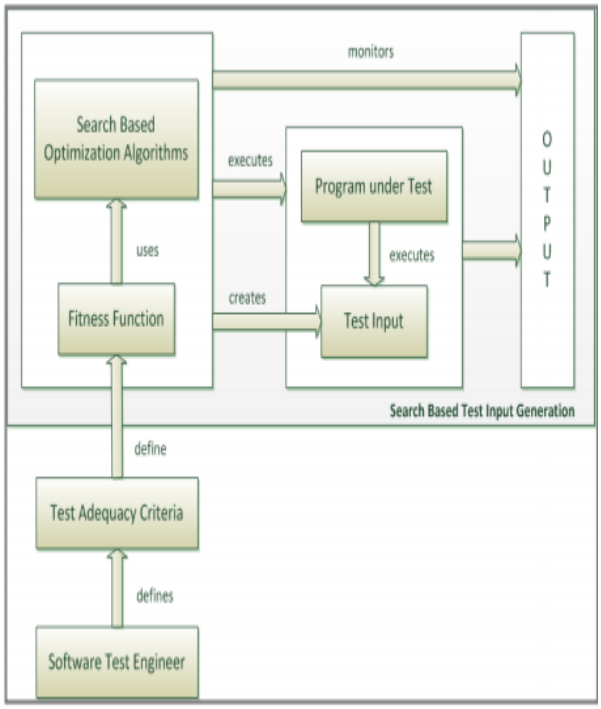


Fig. 2: A generic search generation schema

4. METHODOLOGY

We use web-based subject applications the inputs for these applications are URL requests webbased client-server architecture. Its GUI front-end communicates back-end that accepts HTTP requests in the form of URLs. Its back-end can serve multiple URL requests from multiple users concurrently. Each URL exercises different components of the application. For each subject application, we traversed the web interface and source code of these systems and recorded all unique URLs sent to the back-end, in order to obtain a complete set of URL requests. We define a transaction as a set of URLs that are submitted by a single user. To answer RQ1, we issued multiple transactions in parallel collecting profiling traces and computing the total elapsed execution time for the back-end to execute the transactions. Our goal is to evaluate if GA−Prof can automatically find combinations of URLs that cause increase in elapsed execution time. In our experiments [12] as competitive approach We conducted comparison experiments on subject applications, with artificial delays injected, and compared the effectiveness of both approaches identifying them. To choose the delay length and methods to inject bottlenecks into, we ran the subject applications without injected bottlenecks and obtained a ranked list of methods. On top of this list we obtained natural bottlenecks. A threat to validity for our empirical study is that our experiments were performed on only three open-source web-based applications, which makes it difficult to generalize the results to other types of applications that may have different logic, structure, or input types. However, JPetStore and Dell DVD Store were used in other empirical studies on performance testing [14] and Agile fant is representative of enterprise-level applications, we expect our results to be generalizable to at least this type of web-based software applications. Our current implementation of GA−Prof deals with only one type of inputs - URLs, whereas other programs may have different input

types.

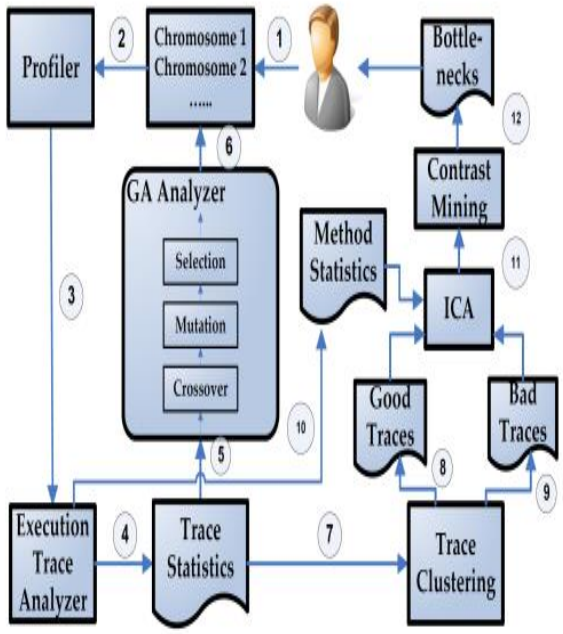


Fig. 3: The architecture and workflow of GA−Prof.

5. STATE OF ART OF WEB TESTING

To bridge the gap between existing web testing techniques and main new feature provided by web application. The server side can be tested using any conventional testing technique. Client side testing can be performed at various levels. The selenium tool is very popular capture-replay tool and allows DOM based testing by capturing user session events fired by user. Such tool can access the DOM and shows expected UI behavior and replay the user session. So today's need is a testing tool which can test user session and generate test cases on the basis of expected UI behaviour as per event fired by user. State Based Testing: Marchetto proposed a state based testing technique [17]. Idea is that the states of client side components of an AJAX application need to be taken into account during testing phase [18]. Web page, and their corresponding values are used for building its finite state model. State based technique results indicate that state based testing is powerful and can reveal faults otherwise unnoticed or very hard to detect using existing techniques.

Traces	Event Sequence
1	add
2	rem
3	add, add, rem
4	add,add, rem,rem,rem
5	add, empty
6	add, empty, rem

Fig. 4: Traces for Cart Events

6. TAO (Testing Assist par Ordinateur)

TAO is an integrated tool performing automated test and oracle generation based on the methodology of denotational semantics. It extends a grammar-based test generator [19] with a formal framework

supporting the three components of denotational semantics, syntax, semantics domains, and the valuation functions from syntax to semantics. TAO takes as inputs a context-free grammar (CFG) and its semantic valuation functions, and produces test cases along with their expected behaviors in a fully automatic way. An online version of TAO is available at [8]. Denotational semantics [20,21] is a formal methodology for defining language semantics, and has been widely used in language development and practical applications. Broadly speaking, for a webbased application under test (WUT) which requires grammar-based structured inputs, the specification of the structured inputs is a formal language; for those testing scripts (or methods) running together with a WUT, the specification of those scripts is a formal language. Denotational semantics is concerned with finding mathematical objects called domains that capture the meaning of an input sentence — the expected result of the WUT, or the semantics of a testing script the running behavior of the script itself along with the WUT. In automated test script generation, it would be ideal that runtime assertions can be automatically embedded into a test script, so that when a test script is invoked for software testing, the running result immediately indicates either success or failure of testing; otherwise, a post-processing procedure is typically required to check the running result against the oracle. It allows users to create a tagging variable as a communication channel for passing results from semantics generation to test generation

An automated web testing framework based on our testing tool TAO and Selenium browser automation. The framework consists of the following main procedures. (i) A WUT is modeled using a methodology of denotational semantics, where CFGs are used to represent the GUI-based execution model of the WUT, semantics domains are used to describe functional behaviors of the WUT, and valuation functions map user interactions to expected web behaviors. (ii) TAO takes the denotational semantics of the WUT as an input, and automatically generates a suite of JUnit tests, supported in the Selenium browser automation tool. (iii) Through Selenium’s web drivers, a suite of JUnit test scripts can be executed to test different scenarios of the WUT.

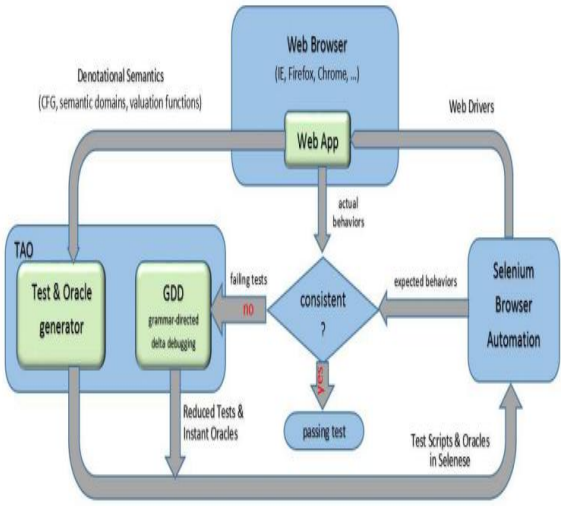


Fig. 5: Automated Web Testing Framework

The function APPLY(α, test, root, pNode), defined in Algorithm 2, applies the reduction strategy α on each node pNode in the derivation tree rooted at root in a top-down, depth-first order. For each nonterminal node pNode (line 21), the sub-function checks whether the the reduction strategy α is applicable on the subtree rooted at pNode (line 22). We highlight two implementation details in this algorithm. (1) We adopt the first-child/next-sibling data structure for representing derivation trees; thus, reducing a subtree of pNode can be achieved by changing its first child link. (2) We use a store/restore mechanism to maintain the original subtree of pNode (line 23). In case that the reduced test case is not failureinducing, we have to restore the original subtree of pNode (lines 27−30); otherwise, the reduced one will be used for further reduction. The function REDUCEBY(pNode, α) applies the reduction strategy α on pNode.

Algorithm 2 APPLY: a search-based reduction procedure

```
15: Input: (1) α, a grammar-directed reduction strategy; (2) test, a failure-inducing test case;
16:        (3) root, the root of the derivation tree of test; (4) pNode, a node in the derivation tree
17: Output: a reduced failure-inducing test case
18: function APPLY(α, test, root, pNode)
19:     if (pNode is a CFG terminal) then
20:         return test
21:     else                                                              ▷
22:         if (α is applicable on pNode) then
23:             store the first child link of pNode                       ▷ use first-child
24:             REDUCEBY(pNode, α)
25:             reduced ← GETTESTCASE(root)
26:             oracle ← INSTANTORACLE(root)
27:             if (TESTING(SUT, reduced, oracle) fails) then
28:                 test ← reduced
29:             else
30:                 restore the first child link of pNode
31:             end if
32:         end if
33:     end if
34:     for each child node cNode of pNode do
35:         test ← APPLY(α, test, root, cNode)
36:     end for
37:     return test
38: end function
```

. Only a failure-inducing reduced test case will be kept for further delta debugging. In both cases, either reduced or not, the function will continue with applying the reduction strategy α on each child node recursively (lines 34−36).

7. EXPERIMENT RESULTS

We first show our preliminary experimental results on automated delta debugging by applying the GDD approach on applications which require structured

inputs, and then show experimental results on Selenium-based web testing

We used the extended TAO with new capabilities, instant oracle and grammar directed reduction strategies, to generate 1000 arithmetic expressions and locate the failure-inducing patterns.

Each of those failing test scripts may contain one or multiple rounds of parking cost calculations, and in each round of parking cost calculation, users may set entry/exit dates and times in any order and modify them repeatedly. Our GDD approach was able to reduce a failing test script to a simplified one, with an average reduction ratio about 22%. We found out that most failures were caused by different time-boundary issues

Table 1: Faults Summary for the Online Parking Calculator

Lot Types	Faults
Garage, Surface, Economy	1. weekly maximum was violated 2. daily maximum was violated 3. wrong parking cost was given when the leave time is earlier than the entry time
Short-term	4. daily maximum was violated 5. half hour price was not properly calculated
Valet	6. wrong parking cost was given when the leave time is earlier than the entry time

Both automated instant oracle generation and grammar-directed delta debugging are critical to automating web testing and fault localization.

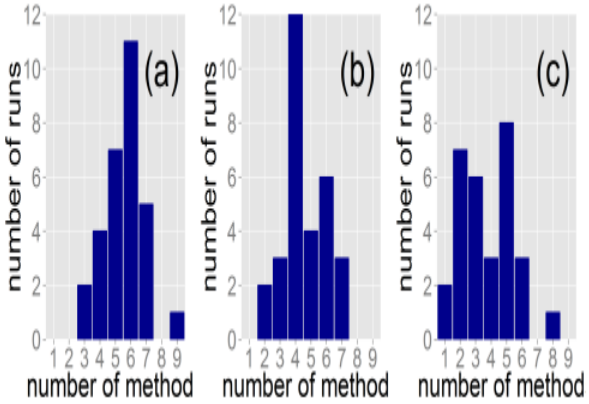


Fig. 6: Distribution of the quantity of captured injected bottlenecks. The x-axis corresponds to the number of injected bottlenecks that are captured by one certain GA−Prof run

8. CONLUSIONS

Our framework is able to generate a suite of executable JUnit test scripts by utilizing grammar-based test generation and semantics-based oracle generation. The main goal is to make a study of the use of search-based optimization techniques to automate the evolution of solutions for software engineering problems.We presented TAO, a testing tool performing automated test and oracle generation based on a semantics based approach, and showed a new automated web testing framework by integrating TAO with Selenium based web testing for web testing automation. For example, real world problems such as optimizing software resource allocation, triangle classification, software clustering,

component selection and prioritization for next release. These are the basis input to our automated test case generation model. In our future work we will be implementing the automated test case generation model using evolutionary algorithm that is genetic algorithm. For example, real world problems such as optimizing software resource allocation, triangle classification, software clustering, component selection and prioritization for next release.

9. FUTURE WORK

As more and more web technologies have moved a long way to create web application. Web testing plays an important role. Here in this paper we discussed two well known testing techniques:-state based testing and invariant based testing. While these approaches are tested successful on various case studies This finding remarks that, there is a need to generate a test environment to test latest web technology designed web application and exercise each of them. New testing issues can arise for testing web services for improving effectiveness and efficiency of web

10. REFERENCES

[1] WasifAfzal, Richard Torkar, and Robert Feldt.A systematic review of search-based testing for non-functional system properties. Inf. Softw. Technol., 51:957–976, June 2009

[2] Mark Harman, “The Current State and Future of SBSE”, Future of Software Engineering (FOSE'07), IEEE Computer Society, 2007, pp. 1-16.

[3] Elbaum, S. ,Karre,S., Rothermel,G., Improving web application testing with user session data. In International conference of software Engineering, pages 49-59, 2003.

[4] Fujiwara, S., Bochmann, G., Khendek, F., Amalou, M., Ghedasmi, A. , Test selection based on finite state models, IEEE Transactions on Software Engineering 17(6):591-603, June 1991

[5] TiantianGao, YujiaGe, Gongxin Wu and Jinlong Ni, “A Reactivity Based Framework of Automated Performance Testing For Web Applications” 9 thInternational Symposium on Distributed Computing and Application to Business, Engineering and Science.

[6] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper,and M. Shepperd, “Reformulating Software Engineering as a Search Problem,” IEE Proceedings - Software, vol. 150,no. 3, 2003, pp. 161–175.

[7] Daniel Malcolm Hoffman, David Ly-Gagnon, Paul Strooper& Hong-Yi Wang (2011): Grammar-based test generation with YouGen. Software Practice and

Experience 41(4), pp. 427–447, doi:10.1002/spe.1017.

[8] UNO LASER Lab (2014): TAO online.

[9] Ralf Lämmel& Wolfram Schulte (2006): Controllable combinatorial coverage in grammar-based testing. In: International conference on Testing of Communicating Systems, pp. 19–38, doi:10.1007/11754008_2.

[10] A. Baars, M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, P. Tonella, and T. Vos. Symbolic search-based testing. In ASE ’11, pages 53–62, 2011.

[11] J. Bach. What is exploratory testing? stickyminds.com.

[12] L. C. Briand, Y. Labiche, and M. Shousha. Stress testing real-time systems with genetic algorithms. In GECCO ’05, pages 1021–1028, 2005.

[13] J. Burnim, S. Juvekar, and K. Sen. Wise: Automated test generation for worst-case complexity. In ICSE ’09, pages 463–473, 2009.

[14] Y. Cai, J. Grundy, and J. Hosking. Synthesizing client load models for performance engineering via web crawling. In ASE ’07, pages 353–362, 2007

[15] P. Saxena, D. Akhawe, S. Hanna, S. McCamant, D. Song, & F. Mao (2010): A symbolic execution framework for JavaScript. In: 31st IEEE Symp. on Security and Privacy, doi:10.1109/SP.2010.38.

[16] Yuanyuan Zhang, “Multi-Objective Search - based Requirements Selection and Optimisation”, Ph.D Thesis, King’s College, University of London, February 2010, pp. 1-276.

[17] Marchetto, A., Ricca, F., and Tonella, P. (2008a). A case study-based comparison of web testing techniques applied to ajax web applications.

[18] Marchetto, A., Tonella, P., and Ricca, F. (2008b). State-based testing of Ajax web applications.

[19] EminGünSirer& Brian N. Bershad (1999): Using production grammars in software testing. In: the 2nd conference on Domain-specific languages, pp. 1–13, doi:10.1145/331960.331965.

[20] A. Stout (2001): Testing a Website: Best Practices. The Revere Group.

[21] Watir: Web Application Testing in Ruby.

[22] Andreas Zeller & Ralf Hildebrandt (2002): Simplifying and Isolating Failure-inducing Input. IEEE Transactions on Software Engineering 28(2),