Evolution of Automated Web Application Testing in SBSE

Nakul Shukla
NCSU
nshukla@ncsu.edu

Nitin Sharma
NCSU
nsharm10@ncsu.edu

Sakthi Sambasivam
NCSU
ssambas@ncsu.edu

Abstract

In this paper we review the developments in the field of automated software engineering, specifically concentrating on web application testing. We take a look at literature from circa 2008 to 2014 and examine the change in approach of utilizing automated software engineering to optimize testing of web application. We see the advances made for search based software optimization in various fields like dynamic test input generation and reverse engineering of software components.

Keywords

Automated Web Application, Automated Testing, Input generation

1. Introduction

A web application is a software system that is accessed over the web via the Hyper Text Transport Protocol (HTTP).The importance of automated web application testing derives from the increasing reliance on these systems for business social, organizational and governmental functions. [5] The service provided by a web application is not limited by location or time, since geographically separated users may have concurrent access. However, these advantages impose a demand for high availability.

Web technologies change more frequently and their adopters seek early acquisition of market share. This pressure on development time squeezes the testing phase, especially when it is un-automated, labor intensive and therefore slow. However, inadequate testing poses significant risks: Studies showed that trust and convenience are major factors affecting customer loyalty using web applications [9].

The main idea of this paper is show the progress made to improve automated web application testing by approaches such as increasing branch coverage with reduction in testing effort [5] or dynamic test generation technique for the domain of dynamic web applications [4]. We look at various aspects of automated testing including different platforms and some challenges associated with them.

We start with discussing certain "traditional" routes for automated web testing and then move on to Genetic Engineering and other such techniques that help tune specific

parameters that improve performance of automated testing systems. Some focus is put on reverse engineering and the considerable potential for the development of new forms of Genetic Programming (GP) and Genetic Improvement (GI) to reverse engineering

The next section starts with the motivation behind such endeavors. We move on to related work and then present some specific aspects that are of interest in these papers. We finally make some comments and recommendations and explore future work in this area.

2. Motivation

Testing is a widely used approach for identifying bugs and for providing concrete inputs and traces that developers use for fixing bugs. However, manual testing requires extensive human effort, which comes at significant cost. Additionally, QA testing usually attempts to ensure that the software can do everything it ought to do, but it does not check whether the software can do things it ought not to do; such functionality usually constitutes security holes [1].

The term 'search based software engineering' was introduced in 2001 [10] to capture the (then emerging) interest in the use of computational search as a means of optimizing software engineering problems. The motivation was that search based optimization was ideal for the multiple conflicting and competing objectives with which software engineers routinely contend with.

Our goal in this work is to summarize the developments pertaining to the amalgamation of these two ideas.

3. Related Work

Early work on web application focused primarily on static pages and the coverage metric was page-coverage. Ricca and Tonella propose a technique for using UML models of web applications to analyze static web pages via testing [11]. Kung et al. model web applications as a graph and develop tests based on the graph in terms of web page traversals [12]. Search Based Software Engineering (SBSE) has been widely used in testing both functional and non-functional properties of applications [13], [14]

Although there is lots of research made in Search based software engineering, Search based test data generation used for these studies is specifically for web applications. Other papers used this approach for Ajax based web applications using Hill climbing Algorithm [3]. Another team introduced an algorithm that uses symbolic execution of the source code to group inputs into interfaces [2].

Several techniques and a few tools have been presented in the literature to support testing of Web applications. Functional testing tools of Web applications record the interactions that a user has with the graphical interface and repeat them during regression testing. Model-based testing of Web applications: Two ways namely, Coverage criteria are defined with reference to the navigational model and navigational model is a finite state machine with constraints recovered by hand by the test engineers directly from the Web application.

4. Sampling Procedure

We started out with reviewing some highly cited papers published in 2011 related to several fields of Automated Software Engineering.

The paper on automated web application testing using SBSE [5] caught our eye and we explored it further. Finding the area sufficiently exploration worthy for the purposes of this paper we went backwards in literature till 2008[1] [2] [3] [4] based on works cited and discussed in the original paper.

Having established a historical context and background on the topic, we proceeded from year 2012[6][7][8] and beyond to examine the latest research on the topic of using Automated Software Engineering to test and debug web applications.

In the next section, we introduce each paper reverse chronologically and mention some points on each paper that will cast some light on each of the papers.

5. Literature Survey

5.1 Dynamic Test Input Generation for Web Applications [1]

5.1.1 Motivation

Automated Test Generation frameworks have proven useful for finding bugs and improving test coverage on Languages like C and Java which is dominated by numerical values and pointer based data structures. However, scripting languages such as PHP promote a style of programming for developing web applications that emphasizes string values, objects and arrays. So the authors propose an approach for analyzing web applications by generating testing inputs for them automatically using information from previous executions. This approach handles dynamic language features more gracefully than static analysis. They generated automated input test generation algorithm that uses runtime values to analyze code, models the semantics of string operations.

5.1.2 Baseline Results

In the first phase of their analysis the authors' team perform source to source translation of PHP so that they can write results in a file to trace the execution (like log file). They recorded the results of execution times and corresponding log sizes for one of the Application (Mantis). They also evaluated how long it took in terms of number of test inputs generated and the total time to generate them. In their observation two applications required relatively few test inputs before they generated an attack.

5.1.3 Areas of Improvements

1. The author just chooses three different real world web applications, but apart from some preliminary details, he didn't mention how these applications qualified for experimental studies. It would have been useful if the criteria for selecting applications were specified.

2. The author and his team chosen only PHP Web application for analyzing generating test inputs. It could have been better if they consider other scripting languages like Java Script to generalize their idea.

3. Their implementation lags few important requirements such as requiring full automation but still requiring some manual procedures like loading the pages and invoking analyzers etc.

5.2 Precise interface identification to improve testing and analysis of web applications [2]

5.2.1 Motivation

Accurate interface identification is fundamental for many of the automated quality assurance techniques, as the components of a web application communicate extensively via implicitly-defined interfaces to generate customized and dynamic content. However, most techniques for identifying web application interfaces can be incomplete or imprecise, which hinders the effectiveness of quality assurance techniques. To address these limitations, the authors present a new approach for identifying web application interfaces that is based on a specialized form of symbolic execution.

5.2.2 Delivery Tools

To evaluate their approach, the authors developed a prototype tool called WAM-SE (Web Application Modeling with Symbolic Execution). The implementation consists of three modules: transform, SE engine, and PC analysis, which correspond to the three steps of their approach.

Transform: The input to this module is the byte code of the web application and the specification of program entities to be considered symbolic (in this case, symbolic strings).

SE Engine: The input to this module is the byte code of the transformed web application, and the output is the set of all Path Conditions (PC) and corresponding symbolic states for each component in the application.

PC Analysis: The input to this module is the set of PCs and symbolic states for each component in the application, and the output is the set of Interface Domain Constraints and accepted interfaces. The module iterates over every PC and symbolic state, identifies the accepted interfaces, and associates the constraints on each IP with its corresponding accepted interface.

5.2.3 Areas of Improvement

1) Due to time constraints of manual checking, the Precision metric was checked only against one subject and may not accurately depict the precision results

2) The results do not talk about fault detection but simply code coverage by test suits generated by this approach

3) The authors do not go into details about why their approach is algorithmically more expensive than some of the techniques they compared against in the paper.

5.3 Search-based testing of Ajax web applications [3]

5.3.1 Motivation

Paper discusses a technique based on dynamic extraction of a finite state machine for an Ajax application and its analysis with the aim of identifying sets of test cases based on semantically interacting events. This paper also investigates the use of a search-based approach to address the problem of generating long semantically interacting event sequences while keeping the test suite size reasonably small.

5.3.2 Delivery Tools

There are three tools that were developed to support the proposed testing technique:

1. FSMInstrumentor: A Javascript module able to trace the execution of the Web application under test,

2. FSMExtractor: A Java module to analyze the execution traces and build the application FSM,

3. FSMTest CaseGenerator: A Java module to analyze the built FSM and generate test suites according to SEM, ALT, and HILL

5.3.3 Areas of Improvement

1. Making improvements to the Finite State Machine recovery step can help automatically infer proper abstraction functions.

2. The experiments in the paper are done on two applications only. So the results cannot be generalized to arbitrary Ajax Web Applications. So the experiments should be done to get a more general result.

3. The scope for input selection should be increased to get better quality test cases that can cover large portions of the application, and expose more bugs.

5.4 Finding bugs in web applications using dynamic test generation and explicit-state model checking [4]

5.4.1 Motivation

The paper presents a dynamic test generation technique for the domain of dynamic web applications. The technique utilizes both combined concrete and symbolic execution and explicit-state model checking. The technique generates tests automatically, runs the tests capturing logical constraints on inputs, and minimizes the conditions on the inputs to failing tests, so that the resulting bug reports are small and useful in finding and fixing the underlying faults.

5.4.2 Delivery Tools

The team created a tool, Apollo that implements their technique in the context of the publicly available PHP interpreter. Apollo first executes the web application under test with an empty input. During each execution, Apollo monitors the program to record the dependence of control-flow on input. Additionally, for each execution Apollo determines whether execution failures or HTML failures occur (for HTML failures, an HTML validator is used as an oracle). Apollo automatically and iteratively creates new inputs using the recorded dependence to create inputs that exercise different control flow.

5.4.3 Areas of Improvement

1. The HTML output of a PHP script might contain buttons and arbitrary snippets of JavaScript code that are executed when the user presses the corresponding button. The actions that the JavaScript might perform were not analyzed by Apollo

2. Apollo had limited tracking of input parameters through PHP native methods. PHP native methods are implemented in C, which make it difficult to automatically track how input parameters are transformed into output parameters.

3. Apollo considers as parameters only inputs coming from the global arrays POST, GET and REQUEST.

5.4.4 Improvements over previous paper

This paper introduced creating test input dynamically using ASE which improved upon the ideas presented in [3].

5.5 Automated Web Application Testing Using Search Based Software Engineering [5]

5.5.1 Motivation

The main idea of this paper is how to improve automated web application testing by increasing branch coverage with reduction in testing effort. This paper introduces threes algorithms and a tool called "SWAT" and explains the positive impact of these algorithms in efficiency and

effectiveness of traditional search based techniques exploiting both static and dynamic analysis. Each improvement is separately evaluated in an empirical study on 6 real world web applications

5.5.2 Delivery Tools

The authors developed a Tool called Search based Web Application Tester (SWAT) to cover the various testing approaches. SWAT consists of two major tools: 1. Search based Tester and 2.Test Harness. The Search based Tester uses the transformed source code and the analysis data to implement the input generation described in three different Algorithms. The Test Harness uses the generated test data to run the tests on the original source code and to produce coverage and bug data. Both Search based tool and Test Harness are implemented in Perl and use the HTTP, HTML and LWP libraries.

5.5.3 Areas of Improvement

1. The author chooses six different web applications, but apart from their size, he didn't mention how these applications qualified for experimental studies. It would have been useful if the criteria for selecting application were specified.

2. The tests were conducted on a simple hardware setup and also tested locally. So it is difficult to generalize the testing results. It would have been more value added if they had tested this using medium size cluster and also setup remote services in a distributed environment.

3. All the applications chosen for the study are PHP applications. To cover major areas of web applications they could have selected other applications implemented by other languages like Java to see the overall effectiveness and efficiency of this approach.

5.6 The Seed is Strong: Seeding Strategies in Search-Based Software Testing [6]

5.6.1 Motivation

This paper considers the aspect of the initial population of the search, and presents and studies a series of techniques to improve search-based test data generation for object-oriented software. The ultimate goal is to reduce the number of test cases generated that maximize the branch coverage and at the same time have as low secondary objectives as possible, as these need manual verification.

5.6.2 Delivery Tools

In the experiments described in the paper, EVOSUITE test generation tool is used. It uses a GA to derive test suites for classes. The EVOSUITE tool operates on byte-code. In all the experiments the default settings of EVOSUITE are used.

5.6.3 Areas of Improvement

1. In this paper, the length of test suite generated is given secondary priority. This might lead to generation of larger test suits with not much increase in achieved coverage.

2. The paper does not take into account how difficult it would be to evaluate the test cases manually to test correctness of output.

3. The paper claims that seeding strategies help EVOSUITE to achieve higher code coverage. But, it is possible that there exists parameter combinations for seeding strategies and search budget that can perform better. So more parameter settings should be tried.

5.7 Genetic Programming for Reverse Engineering [7]

5.7.1 Motivation

This paper focuses on reverse engineering and the considerable potential for the development of new forms of Genetic Programming (GP) and Genetic Improvement (GI) to reverse engineering. It presents a summary of the application of SBSE to reverse engineering. It briefly reviews the relationship between the SBSE and RE publication venues and trends.

5.7.2 Commentary (Overview of the process)

At a high level, key steps likely to occur in a software transplant algorithm to add feature F from source System D (the donor) to destination System H (the host):

1) Localize: Identify and localize the code that implements F (this might use, for example, concept and feature location

2) Abstract: Construct an abstraction AF from DF, retaining control and data flow directly related to F in the donor but abstracting references to D-specific identifiers so that these become parameterized.

3) Target: Find locations HF in the host, H, where code implementing F could be located.

4) Interface: Construct an interface, I and add it to the host, H, allowing the resulting combination H [I] to act as a 'harness' into which candidate transplants can be inserted and evaluated.

5) Insert: Instantiate and concretize a candidate transplant (concretized from AF) at HF.

6) Validate: Validate the resulting transplanted system.

7) Repeat: Repeat the above steps until a suitably well tolerated transplant is found.

5.7.3 Areas of Improvement

1. The paper mentions very few practical examples of where their techniques were used and their results.

2. The paper does not mention open avenues of future work or how this work may be used to advance the study of Reverse Engineering

3) The results have not been collected and tabulated suitably so that they can be analyzed and conclusions be drawn

5.8 SBSE for software product line (SPL) engineering: a survey and directions for future work [8]

5.8.1 Motivation

There is a tremendous upsurge in SBSE for SPL. The authors did a comprehensive work on topics documenting recent advances in genetic engineering, search based branch merge and graft genetic improvement. Based on their exhaustive study they also highlighted the direction for future work. They mainly focused on genetic improvements showing how these improvements could be exploited by SPL researchers and practitioners.

5.8.2 Sampling Procedure

Real world feature models typically involves many constraints. Tracing variability information between problems (requirements) and solutions (products) is challenging. Search based feature model selection is isomorphic to the previously studied search based requirements selection problem. Search based requirements

selection seeks to find requirement selections, while respecting constraints. There are many researchers and practitioners formulating procedures and constraints to select a sampling models and features.

5.8.3 Areas of Improvement

1. The paper explains some of the topics in detail and explains the other topics in an extremely succinct manner, which leave many questions unanswered. It could have been better if the authors' team had treated all the topics in a more uniform manner.

2. Although the paper gives some excellent details about SBSE history, it lacks one of the important element e.g. study instrument to suffice more depth in their study.

3. The author has used so many references instead of giving complete explanation for the different topics. This leaves an incomplete understanding of the some of the topics leading to more questions.

6. Future Work

The following areas are considered to be more promising and are potential topics for future work in Automated Web Application Testing in SBSE.

To incorporate the success of genetic programming in software engineering, search based software engineers have turned to a technique that has come to be known as "Genetic improvement". Genetic Improvement can be used to provide Pareto frontier of programs. This Pareto program surface contains a large number of different programs (‘products’ in SPL nomenclature), each of which share the same functionality, yet all of which differ in their non-functional properties.

When the feature model grows exponentially, that will lead to an unmanageable number of different product variants. This situation is called ‘branchomania’. Using SBSE developed techniques to control branchomania by identifying branch similarities, extracting parameters and subsequent searching for suitable tunings yield individual products. A set of child branches of a shared parent could thereby be merged into a single modified parent with an additional set of parameters that capture the variability previously present in the children. In this way, a combination of parameter extraction and tuned parameter instantiation could merge several products into a single parameterized product.

To exploit the current improvements in hardware specifically multicore processors, search based software engineers have realized the possibility of parallelization as a route to scalability using computing clusters. Parallel SBSE may prove to be particularly important in scaling computational search algorithms to handle large-scale software product lines.

The other promising area in Automated Web Application Testing is FSM recovery step, in order to automatically infer proper abstraction functions. Experimenting with alternative search based algorithms and applying them to a larger benchmark of Ajax applications could be another area where improvement is possible. It includes the investigation of the role of input selection and infeasible paths in the FSM during test case generation.

7. Conclusion

We reviewed 8 years of work done in the field of Web application testing using Automated Software Engineering. Throughout the review we observed the growth of techniques that are used for this purpose.

We began with applying SBSE in dynamically generating test data to be used in testing of web applications in a PHP based environment. We then looked over techniques to precisely identify interfaces to improve testing and analysis for web applications. Following this we looked at papers that utilized these techniques in specific contexts like AJAX based applications.

Towards the end we looked at some of the newer uses of such techniques with concepts of reverse engineering. Through this study, we have highlighted that SBSE has great potential in the field of analyzing and finding bugs in web applications.

8. Acknowledgement

We as a team, thank Prof. Tim Menzis for providing this learning opportunity to dive deeper into some of the areas of the SBSE and round off our knowledge on the subject.

9. References

[1] Wassermann, Gary, et al. "Dynamic test input generation for web applications." Proceedings of the 2008 international symposium on Software testing and analysis. ACM, 2008.

[2] Halfond, William GJ, Saswat Anand, and Alessandro Orso. "Precise interface identification to improve testing and analysis of web applications."Proceedings of the eighteenth international symposium on Software testing and analysis. ACM, 2009.

[3] Marchetto, Alessandro, and Paolo Tonella. "Search-based testing of Ajax web applications." Search Based Software Engineering, 2009 1st International Symposium on. IEEE, 2009.

[4] Artzi, Shay, et al. "Finding bugs in web applications using dynamic test generation and explicit-state model checking." Software Engineering, IEEE Transactions on 36.4 (2010): 474-494.

[5] Alshahwan, Nadia, and Mark Harman. "Automated web application testing using search based software engineering." Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011.

[6] Fraser, Gordon, and Andrea Arcuri. "The seed is strong: Seeding strategies in search-based software testing." Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. IEEE, 2012.

[7] Harman, Mark, William B. Langdon, and Westley Weimer. "Genetic programming for reverse engineering." Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, 2013.

[8] Harman, Mark, et al. "Search based software engineering for software product line engineering: a survey and directions for future work. “Proceedings of the 18th International Software Product Line Conference-Volume 1. ACM, 2014.

[9]Rolph E. Anderson and Srini S. Srinivasan. E-satisfaction and e-loyalty:

A contingency framework. Psychology and Marketing, 20(2):123–138,

2003

[10]Mark Harman and Bryan F. Jones. Search based software engineering.

Information and Software Technology, 43(14):833–839, December 2001

[11] F. Ricca and P. Tonella. Analysis and testing of web applications. In Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), pages 25–34, 2001

[12] D. Kung, C. H. Liu, and P. Hsia. An object-oriented web test model for testing web applications. In 24th International Computer Software and Applications Conference (COMPSAC 2000), pages 537–542, 2000.

[13] Wasif Afzal, 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.

[14] Phil McMinn. Search-based software test data generation: a survey. Software Testing, Verification and Reliability, 14(2):105–156, 2004.

[15] Wasif Afzal, 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.

[16] [3] Mohammad Alshraideh and Leonardo Bottaci. Search-based software test data generation for string data using program-specific search operators.Software Testing, Verification and Reliability, 16(3):175–203, 2006.

[17] William Halfond and Alessandro Orso. Automated identification of parameter mismatches in web applications. In SIGSOFT '08/FSE-16, pages 181–191, 2008

[18] Nadia Alshahwan and Mark Harman. Automated session data repair for web application regression testing. In ICST '08, pages 298–307, 2008.

[19] N. Tracey, J. Clark, K. Mander, and J. McDermid. An automated framework for structural test-data generation. In ASE '98, pages 285–288, 1998

[20] M. Emmi, R. Majumdar, and K. Sen. Dynamic test input generation for database applications. In ISSTA, pages 151–162, 2007.