Review of developments in the applications of automated software engineering

Deepak Nanihalli
North Carolina State University
dnandih@ncsu.edu

Madhura Rajopadhye
North Carolina State University
mrajopa@ncsu.edu

Nithanth Kudige
North Carolina State University
nkudige@ncsu.edu

ABSTRACT

In this paper we review the recent developments in the applications of automated software engineering, specifically in the domain of web application based testing. We have reviewed literature pertaining to this domain published in the range of 2008 to 2015. We examine how the literature proceeds in describing the changes that have occurred in how we apply automated software engineering and its components to optimize several aspects of web applications ranging from parameter testing to testing an application's non-functional properties.

Keywords

Web application testing, search based software engineering, automated software engineering, test case generation

1. INTRODUCTION

Web application is a system which typically is composed of a database (or the back-end) and Web pages (the front-end), with which users interact over a network using a browser. A Web application can be of two types – *static*, in which the contents of the Web page do not change regardless of the user input; and *dynamic*, in which the contents of the Web page may change depending on the user input, user interactions, sequences of user interactions, etc.[9]

Web applications are at the forefront of every industry and online retail sales are skyrocketing as showcased by the authors of the paper 'A profile of the Internet Shopper'. Even though the figures are from 2008, we can surely say that all over the world, the internet shoppers have risen in quantity and exhibit similar features making web applications an important aspect in any industry and not just retail.

Given this rise, the need for highly tested robust applications has risen. Given the large amount of users, the various situations these developed applications will be put under has become broader and this calls for a larger spectrum of test cases. The properties of these applications (functional as well as non-functional) need to be made sure are as expected. Broadly, testing these applications is a huge area that needs to be dived into.

There are several aspects of automated software engineering such as search based software engineering which are making their way into web application development phases like test case generation, identification of parameter mismatches. Researchers have found ways to create test cases, identify faults or cracks in the flow mechanism of the applications through optimization techniques and use of evolutionary algorithms. This practice has become so widespread that comparing various approaches and enlisting their suitability/unsuitability for different application areas has become inevitable.

Hence, we have also reviewed and documented few papers which present a review of the advances that have occurred in this domain. The idea behind collating such surveys is that we get a broader variety of literature and its review at our disposal. A larger degree of exposure is achieved while comparing various methods. We also get a larger view of the domain which helps us in understanding how the areas of improvement highlighted in the earlier papers were addressed by works published later.

We first present some related work of such nature in the next section where we define our domain and highlight some highly cited published papers which describe this area of interest. Next we move on to observe the progress of this field of automated software engineering and how the areas which stunted earlier authors or the aspects which the earlier authors desired to be addressed were addressed. Finally, we present our recommendation and opinion of these works and what we think should be done next in order for the domain to advance itself.

2. MOTIVATION

Given the huge demand for web application and the subsequent pressure on the systems to perform accurately, we need to take care that each and every aspect of the web application is tested thoroughly. Manual testing has its limitations and does not always inculcate all the different sides of an application like interoperability, security and dynamics. [9]

Also, there can be various programs that can be domain-specific which can be developed to test the output for one given application. However, such domain specific testing is non-reproducible and naturally, not reusable. In the age of dynamic development and progress, there cannot be much time spent on any process that can be automated. Computers can be designed to carry out the grunt work while the expensive human labor can be reserved for intelligent design work. This is where the idea of non-domain specific testing framework, which forms the backbone of automated software engineering, comes into play. We know that once a technique or algorithm is developed, it can easily be transformed into domain specific language. This process, we imagine, would involve a one-time effort to create a conversion program. Once, we have this in place, there can obviously be a concentration of efforts towards the development of non-domain specific algorithm for automation of testing.

The great advantage of such automation and utilization of optimization techniques, search based software engineering and abstraction of the application, according to us, is that one would have the entire world of automated software engineering at their disposal without having to worry about the nitty-gritties of one's application. Any advancement in the field would mean advancement in the testing procedures for the developer.

3. RELATED WORK

In the history of works published, the earliest of the works that was examined by us was that by authors Gary Wassermann, Dachuan Yu, Ajay Chander, Dinakar Dhurjati, Hiroshi Inamura and Zhendong Su in 2008: Dynamic test input generation for web applications. This is a highly cited paper which highlights the need for automated test case generation and dynamic testing. They emphasize that manual testing can be a laborious task which is still not efficient. They draw attention to three areas of web application testing which are - test case generation, web application testing and, static analysis of web applications. They have used methods like concolic testing and constraint generation as a part of automated test generation.

A related entry in this domain, is by James M. Clarke in as early as 1998. This is titled 'Automated Test Generation from a Behavioral Model'. This shows us that automating the process of testing is not a new pursuit but was pursued by many others earlier in domains other than web applications. Many take inspiration from this paper as this highlights the use of a behavioral model. The behavior of the application under test is used to establish which automated test suite is the most suitable for carrying out its testing. A similar behavioral model has later been utilized to establish which of the available techniques produced by automated software engineering can find themselves suitable for which domain within web applications.

Automated web application testing is not limited to dynamic test case generation. We also need an automated framework which will enable us to identify faults. There are two notable works in this area. First is by William G.J. Halfond and Alessandro Orso on the Automated Identification of Parameter Mismatches in Web Applications. The components of a web application communicate extensively in order to provide a feature-rich environment. These components integrate content and data from multiple data sources. The complexity of inter-component communication, generation of interface invocations and definition of accepted interfaces occur at runtime, makes these errors more probable. Such errors can be serious and can cause a component to fail unexpectedly. And hence, identifying parameter mismatches is an idea of high importance. Second piece of literature mentioned earlier is by Artzi, S. IBM Thomas J. Watson Res. Center, Hawthorne, NY, USA Kiezun, A., Dolby, J., Tip, F. ; Dig, D., Paradkar, and A. ; Ernst, M.D. The piece is titled Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking. This addresses exactly the part about finding bugs. The authors have developed a technique which automatically generates tests and runs them while inculcating the necessary constraints on the inputs. The conditions on inputs are gradually minimized in order to identify specific faults. They have implemented this technique with a tool called Apollo for PHP.

Finally, there are several surveys and systematic literature reviews which analyze all the methods that researchers around the world are using to imbibe optimization techniques to automate the testing of their web applications. We have reviewed a few of these to get better exposure and understand the snapshot of each at a glance. The number of such systematic reviews has been increasing with time given the amount of publishing that is carried out in the domain.

4. SAMPLING PROCEDURES

We started with reviewing the works published in 2011 which were related to the domain of automated software engineering. We wanted to select a highly cited paper from which we could work backwards and then, at a later point, forwards for our analysis.

We glanced through various papers manually where we came across a wide range of applications where automated software engineering was used a technique for various optimizations purposes. The arena of web applications piqued the group's interest and decided to pursue papers related to it henceforth. We started with 'Automated Web Application Testing using Search based Software Engineering' published in 2011 by Nadia Alshahwan and Mark Harman. We then started to review works published and cited by these authors in their publication backwards in time. Four such works were reviewed. Next, we worked forward in time and reviewed papers that cited the above mentioned publication. This drew a very clear picture in front of us about how the progress went on the field. Since, most of the work reviewed by us that cites the above mentioned publication is mostly systematic literature reviews, we get a clearer picture of how it all ties together.

This paper presents a high level summary of it all and also some other publications of interest that are related.

5. RESULTS

The arena for automating the testing of web applications is quite rich and broad. Search based testing is a significantly strong method, that struck our fancy, which is used in evaluating various system properties, functional properties, non-functional properties and test case generation. This technique is the one of the most up and coming techniques for the automation of test case generation. There are several other techniques like model and graph based, mutation testing, random testing and, scanning and crawling methods which are used for this purpose.

We provide an overview of how automated software engineering fits in the picture in the following areas

1. Application testing as a whole
2. Testing of non-functional properties
3. Test case generation

Each of these is described below in the following sections. It appears from all the literature reviews and surveys conducted that extensive research has been taking place in the past two decades in this arena with much more to come. Given the exceptional advances occurring in computing power, we can inculcate more and more complex algorithms and ideas for fitness functions to drive better efficiency in the demonstrated results.

5.1 Application testing

An early review by Alalfi et al. focused on modeling aspects of web application testing was mentioned in the paper by Yuan-Fang Li , Paramjit K. Das and David L. Dowe in their publication ‘Two decades of Web application testing-A survey of recent advances’. Another notable mention by the authors which proved to be a motivation for them is the work by Di Luca and Fasolino on general approaches to web application testing. The authors dive into the domain and look at various approaches that are valued and compared. The different approaches evaluated in the above mentioned publication are:

5.1.1 Model and Graph based testing

These techniques are not yet automated and need some input from the user regarding the design of the model of the application. The idea is to create a design at an abstract level in the form of a model or a graph that captures the essence of the application. Cost effectiveness and code coverage are the main evaluation methods which are used for the purpose of scoring. The input to be provided by users can be a model of the application in which case the framework is output a regular expression from which the test cases can be created. Also, a lower level finite state machine can be alternatively used as an input for such techniques and the output that will be produced by the framework will be an application level finite state machine from which we can generate test cases. As observed, the entire procedure requires manual effort but there is definitely scope observed to automate it in the future.

The disadvantage of model or graph based testing in our view is the high amount of efforts it requires to imbibe application specific elements. Another disadvantage is the stopping criterion. There is no well-defined stopping criterion which can be widely used but depends on the program/application.

5.1.2 Mutation testing

Unlike model and graph based testing, mutation testing can be automated. The main area where mutation testing is applied is to find out the most rare as well as the most common errors in the application by changing the lines of the source code through some predefined mechanism. Density of faults detected and code coverage have proven to be good evaluation techniques for mutation testing. An example input for such a testing procedure would be a program and a mutation operator. The result would be mutated program that has been mutated according to the mutation operation provided.

The best part about this is that we know exactly when to stop. The mutation operation once finishes its mutation, the program stops.

There is an entire paper published by Upsorn Praphamontripong and Jeff Offut from George Mason University, Fairfax on Applying Mutation Testing to Web Applications where they defined eleven web mutation operators and implemented them in a tool called webMuJava. It is interesting to see the results of the case study and observe how mutation testing effectively helps in finding faults.

5.1.3 Search based testing

This testing technique is the core of automated software engineering. It can be automated. It is used to test as many branches as possible in an application via the use of heuristics to guide the search. Various evaluation methods can be utilized such as cost-effectiveness, density of faults detected and coverage.

A typical input to such a framework would consist of mutation of application inputs as generated by the different heuristics such as hill climbing, simulated annealing and evolutionary algorithms. The output would be a test suite with the aim of maximizing the branch coverage. The stopping condition for this is arbitrarily defined by the programmer as a fixed number of executions or iterations.

We would like to look more into search based testing for web applications through mentioning a publication by Serdar Doğan , Aysu Betin-Can , Vahid Garousi called Web application testing: A systematic literature review. This was carried out in May 2014 and hence is a pretty recent work. They started out with a bunch of research questions aimed at finding out what types of test models, fault models and tools have been proposed for this technique, how the empirical studies in web application testing are reported and, what is the state of empirical evidence in web application testing.

They reviewed several publications from online paper repositories and other secondary studies. The results of their literature review are enlisted as follows:

1. A large number of studies (40, 42%) proposed test techniques which used certain models as input or inferred (reverse engineered) them. The models can be classified as navigation models, control and data flow models and, DOM models, etc. Navigation models were found to be the most popular followed by DOM models
2. Different bug/fault taxonomies related to web applications were found to be proposed in various primary studies like authentication/permission, internationalization, functionality, database persistence
3. Various studies reference tools that support web application testing (54%). The more recently published studies (2008 and later) reference tools that are available for download which the authors reckon is a good sign. Some of the tools presented were MBT4Web, MUTANDIS, ATUSA, Crwaljax, etc which are available for download
4. Different metrics used for assessing cost and effectiveness of WAT techniques are - cost metrics: effort/test time, test-suite size, memory space and others related to complexity) and effectiveness metrics: code coverage, detecting injected faults model-based requirements coverage, etc.
5. Sometimes multiple metrics or pairs of metrics were found to be used
6. Various validity threats are found to question the validity of these research studies. External validity was found to be most common threat identified by various papers. Validation research facet of empirical studies succeeded to identify only 51% of the threats posed in their studies
7. Most of the empirical studies have outlined the context of their studies where a reader can compare it with other studies and the study design has been explained well
8. Most of the studies were performed in a laboratory setting while some in industrial setting. However, none of the studies used methods relevant to the practitioners.
9. Three of the empirical studies studied scalability and reported corresponding evidence
10. Empirical studies reported carrying out comparisons with other studies/tools. Highest of these kind of comparison was session based testing.
11. 62 studies target server side application, 45% target client side application and only 5% provide empirical results of both sides of a web application

As we can see there is extensive work in the area of search based software engineering applied to web application testing. Overall, this gives us an overview of what kinds of studies are being carried out and what aspects are being addressing.

5.1.4 Scanning and Crawling Methods

There are certain procedures which implement scanning and crawling methods which can be automated. The main purpose of these are to detect faults in web applications via injection of unsanitised inputs and invalid SQL injections in user forms, and browsing through a Web application systematically and automatically.

A typical input to such a framework would be unsanitised user inputs to crash the Web application (e.g., the database or force the user to enter unsafe Web sites). It would output type of defects and number of defects. A good stopping criterion for this would be when all the forms for a given Web application is injected with different forms of unsanitised user input.

There are certain others which cannot be automated. These are used for detecting navigation and page errors by systematically exploring pages and filling out forms. The evaluation methods used by these are cost-effectiveness, coverage and density of faults detected.

A typical input to the framework would be a starting web URL and it would output the number of defects and type of the defects found. A good stopping condition is a predefined maximum depth of exploration. Once this depth is reached, the running is stopped.

Scanning and crawling methods don't seem to be highly popular as no noteworthy literature addressing this was found. This is probably due to the complexity of scanners and crawlers.

5.1.5 Random Testing

This is a technique which can be automated. It resembles in some way to search based testing minus a fitness function.

This technique is mainly used in detecting errors using a combination of random input variables and assertions. Cost-effectiveness, density of faults detected and coverage are all evaluation strategies used to establish the goodness of an implementation.

Some sample inputs and corresponding outputs are shown below in the Table 1.

Table 1. Various input and outputs of applications that inculcate random testing methods

Input	Output
WS-CDL Web service choreography specifications	Test oracles (a test suite including assertions)
State machine models of WSDL Web service descriptions	A test suite, and its execution and visualization
A JavaScript program	A set of test cases and their execution results
A JavaScript program	Test oracles in the form of contracts and random test cases based on contracts

The stopping condition can vary depending on the application. For example, it could be when a predetermined number of executions is reached or when the stop instrument in the control flow graph is reached or all contracts are exhausted.

Random testing is one of the oldest methods used for the purposes of testing and all the new methods that are developed are compared with random testing.

One notable publication in the field of random testing was carried out by Justin E. Forrester and Barton P. Miller from the Computer Science department of University of Wisconsin in 2000 titled 'An Empirical Study of the Robustness of Windows NT Applications Using Random Testing'. They tested over 30 GUI-based applications by subjecting them to two kinds of random inputs. Even though the study is aimed at validating the effectiveness of Windows NT with respect to older version of Windows, it provides a good example of random testing can be used.

5.1.6 Fuzz testing

Fuzz testing is automated and is used to test the application by passing in random, boundary or invalid input parts. Like random testing, this type of testing is evaluated through its cost-effectiveness, density of faults detected and coverage.

An input given to such a framework could be Random user inputs which test rare or unlikely system behavior (e.g., values near the boundaries). The output would be test cases with oracles. In such a case, the stopping criterion would be application specific.

Another type of input which can be provided is JavaScript program and a benign input. This would result in a test suite and the identified potential client-side validation vulnerabilities. A stopping condition in this case would be when all identified data flows are exhausted.

Lastly, we can also provide a program, a grammar and an initial input to the framework and expect a test suite with identified defects as the result. A stopping condition would be when all generated inputs are exhausted.

There is one piece of literature which directly addresses fuzz testing for web applications. It is by Rune Hammersland and Einar Snekkenes titled 'Fuzz testing of Web Applications' where they take a deeper look into this type of testing for web applications. Although not a very thorough experiment was conducted, it provides an example use case for future work.

5.1.7 Concolic testing

Concolic testing is automated for most parts. The main purpose is to test as many branches as possible by venturing down different branches through the combination of concrete and symbolic execution. Cost-effectiveness, density of faults detected and coverage can are all widely used evaluation methodologies.

Concrete inputs from symbolically solved previous iteration, start with random initial input can be provided to a framework that makes use of concolic testing and we can expect path constraints at iteration as output. A good idea for a stopping condition would be when all the constraints in the program are exhausted.

A good use case is demonstrated by Saswat Anand, Mayur Naik and Hongseok Yang in 'Automated concolic testing of smartphone apps' where they demonstrated their developed algorithm on 5 smartphone apps.

5.1.8 User session based testing

This approach can be automated. It is typically used when we want to test the web application by collecting a list of user sessions and replaying them or to reduce the test suite size. In each case the user session would serve as an input. As output we can expect a combination of URL and the parameters to be passed to the server or an updated concept lattice and an updated test suite.

We can evaluate the effectiveness of this technique by using various measures like cost-effectiveness, density of faults detected and coverage.

The stopping condition can depend on whether or not certain selected user sessions have been tested, or if reasonable coverage has been achieved. We can even continue the execution till all the user sessions to be tested have been exhausted and the test suite and the lattice cannot be modified anymore.

A notable publication on user session based web application testing is 'Improving web application testing with user session data' by Sebastian Elbaum, Srikanth Karre and Greg Rothermel from 2003. They show that user session data can produce test suites as effective overall as those produced by existing white-box techniques, but at less expense.

Thus, we can see that there are various approaches each with different use cases that can be applied towards web application testing.

5.2 Testing of non-functional properties

Web application testing mainly addresses functional parts or properties of the web application. We now explore how non-functional properties are tested.

Here we examine the paper by Wasif Afzal, Richard Torkar and Robert Feldt titled 'A systematic review of search-based testing for non-functional system properties'. Here they look into how search based testing can be applied to testing of the non-functional properties of web applications.

The non-functional properties are the ones that relate to execution time, safety, quality of service, etc. which we can all agree are fundamental to a good user experience and in turn fundamental to the popularity of a web application.

Several non-functional properties and the results enlisted by the authors regarding the literature surrounding it are enlisted below.

5.2.1 Execution time

Execution time is tested using various metaheuristics like genetic algorithm, extended genetic algorithm and simulated annealing with genetic algorithm being the most popular one. The most widely used fitness functions within these metaheuristics is execution time in terms of processor cycles or other time units or coverage of code annotation.

5.2.2 Quality of Service

The only algorithm which is used for testing this non-functional property of web application is genetic algorithm. The fitness function is based on he maximization of desired QoS attributes while minimizing others, including a static or dynamic penalty function, Combination of distance based fitness that rewards solutions close to QoS constraint violation with a fitness guiding the coverage of target statements

5.2.3 Security

Security is a highly important aspect and probably the most important of all non-functional properties of web applications. Naturally, the related work spreads across a broad range with researchers utilizing many different metaheuristics like genetic algorithm, grammatical evolution, linear genetic programming and particle swarm optimization. Genetic algorithm is the most widely used metaheuristic among all.

The fitness function is based on the configuration of registers and stack, fitness functions covering vulnerable statements, maximum nesting level and buffer boundary, etc.

5.2.4 Usability

Testing the usability of a web application can be quite tricky especially when we want to automate the process.

Different metaheuristics used by researchers for this purpose are tabu search, simulated annealing, Hill climbing, genetic algorithm and ant colony algorithm. The most commonly used is the simulated annealing.

Fitness function within the metaheuristic is defined through the number of uncovered t-subsets

5.2.5 Safety

Safety too, like security is an important aspect of web applications. Genetic algorithms and algorithms based on simulated annealing have been used as metaheuristics for this purpose.

Fitness functions take into consideration costs related to the violation of the safety property and achievement of dangerous situation. They evaluate different branch predicates with zero cost if the safety property evaluates to false and positive cost otherwise.

Thus, we have good amount of literature defined for testing out non-functional properties of web-applications.

5.3 Test case generation

Test cases are the backbone of testing applications. The richness of test cases defines the robustness of the web application. Various methods have been used to dynamically generate test inputs.

Gary Wassermann, Dachuan Yu, Ajay Chander, Dinakar Dhurjati, and, Hiroshi Inamura and Zhendong Su published a paper titled ‘Dynamic Test Input Generation for Web Applications’ in 2008. They talk about the various methodologies used for this purpose. According to them, to achieve correctness of web application, testing plays an important role. There is an increasing amount of literature which inculcates measures to automatically test the web application. They address a related idea of automatically generating test cases and using these to test the web application.

Another piece of literature by Saswat Anand , Edmund K. Burke , Tsong Yueh Chen , John Clark , Myra B. Cohen , Wolfgang Grieskamp , Mark Harman , Mary Jean Harrold and , Phil Mcminn called ‘An orchestrated survey of methodologies for automated software test case generation’ summarize and compare the various different approaches used by researchers from the domain to address the automation of test cases.

5.3.1 Symbolic execution

Symbolic execution is a program analysis technique that analyzes a program’s code to automatically generate test data for the program.

Test case generation using this method uses program analysis and constraint solvers. This method can be used in combination with other methods. This technique has various open problems - path expulsion, path divergence and complex constraints. Though solutions ot these problems exist, more research is needed so that this technique can applied to real-world problems.

5.3.2 Model based testing

Model-based testing (MBT) is a light-weight formal method which uses models of software systems for the derivation of test suites.

MBT, unlike traditional methods, does not use formal models for verification; it makes use of insights in the correctness of a program. It encompasses various approaches like axiomatic approaches, FSM approaches and labelled transition system approaches. A variety of notations are used for describing models like scenario-oriented, process-oriented and state-oriented. There are various tools available for carrying out model-based testing like Conformiq designer, Smartesting Certifylt and Spec Explorer.

5.3.3 Combinatorial testing

It consists of selecting a sample of input parameters (or configuration settings), that cover a prescribed subset of combinations of the elements to be tested.

CIT has traditionally been used as a specification-based, system testing technique to augment other types of testing. It is meant to detect one particular type of fault; those that are due to the interactions of the combinations of inputs or configuration options. There has been an increase in the number of algorithms that generate CIT samples. More and more applications make use of this technique. The authors feel that this is a promising area of research for automated test case generation.

5.3.4 Adaptive random testing

It refers to the family of testing methods in which test cases are random and evenly spread across the input domain.

This is the only feasible technique present if the source code is unavailable and the specifications are incomplete. Adaptive random testing is an enhancement of random testing. There are many approches that have been developed to implement the concept of ART (spreading of test cases over the input domain) like selection, exclusion, partitioning, test profiles, metric-driven, etc. Measures of effectiveness for ART include P-measure, F-measure, time to detect first failure, etc. These use more computation time and memory as compared to random techniques.

5.3.5 Search based testing

We have already taken a look at search based testing while looking over testing web application as a whole. But to revise, Search based software testing (SBST) is a branch of search based software engineering (SBSE), in which search algorithms are used to automate the process of finding test data that maximizes the achievement of test goals, while minimizing testing costs.

The techniques make use of various search algorithms to come up with test cases and evaluate them using a fitness function. The definition of this fitness function is a major concern. This approach is widely used due to its applicability to any test objective.

6. AREAS OF IMPROVEMENT

We know that extensive research is ongoing in the fields and there is little scope to define areas of improvement. However, the authors of various publications reviewed themselves point of certain instances where their research is lacking.

One area that was highlighted in almost all the papers we reviewed before 2011 was the need for empirical results which compared different methods available for testing out the different aspects of web applications. This has been partially addressed in the literature post 2011 where the appropriate comparisons have been made and we take a look at how various approaches are suitable under what circumstances.

Another area of improvement highlighted in the paper by Yuan-Fang Li , Paramjit K. Das and David L. Dowe is the need for testing strategies which take into account concurrent user requests keep track of web application states.

Also, more focus can be made on evaluation methods for the purpose of comparison between each of the methods enlisted. We can weigh how each evaluation method can be utilized for different domain applications and make a call accordingly while developing.

7. CONCLUSION AND FUTURE WORK

We have reviewed vast amount of literature related to the topic of how automated software engineering can be used to automate the process of web application testing. In all, it was observed that search based testing is a highly recommended approach which is widely used to optimize as well as implement various aspects of web application testing like non-functional properties, test case generation, etc.

In that genetic algorithm is the poison of choice for most of the researchers which is obvious given its simplicity and ability to be applied to absolutely any domain.

What we found surprising is that simulated annealing was found to be a highly explored area for web application testing. In terms of what we would like to see in the future would be to have some work done on inculcating simulated annealing as a process for web application testing. We expect that it would prove to be highly efficient due to its strategy for avoiding local minima.

Our work is limited to systematic reviews and subsequent enlisting of processes that are applied for the purpose of web application testing. However, what can be done is to include empirical results. Also, a comparative study can be done where the empirical results can be used to showcase why a certain method is better than the other.

We also recommend diving into other processes within search based testing besides the ones listed by us and carrying out a comparative study for them and as to why they aren’t as widely used.

In conclusion, we think search based testing is the present and future of web application testing due to its flexibility and non-domain specific application. It can be molded according to any industry as well as domain.

7. ACKNOWLEDGMENT

We thank Prof. Timothy Menzies for giving us the opportunity to enrich our scope of knowledge in automated software engineering and how it is applied in web application testing

8. REFERNECES

[1] Elbaum, Sebastian, Srikanth Karre, and Gregg Rothermel. "Improving web application testing with user session data." In *Proceedings of the 25th International Conference on Software Engineering*, pp. 49-59. IEEE Computer Society, 2003.

[2] Forrester, Justin E., and Barton P. Miller. "An empirical study of the robustness of Windows NT applications using random testing." In *Proceedings of the 4th USENIX Windows System Symposium*, pp. 59-68. 2000.

[3] Artzi, Shay, Adam Kieżun, Julian Dolby, Frank Tip, Danny Dig, Amit Paradkar, and Michael D. Ernst. "Finding bugs in web applications using dynamic test generation and explicit-state model checking." *Software Engineering, IEEE Transactions on* 36, no. 4 (2010): 474-494.

[4] Saswat Anand, Mayur Naik, Hongseok Yang, Mary Jean Harrold *Automated Concolic testing of smartphone apps*

[5] Wassermann, Gary, Dachuan Yu, Ajay Chander, Dinakar Dhurjati, Hiroshi Inamura, and Zhendong Su. "Dynamic test input generation for web applications." In *Proceedings of the 2008 international symposium on Software testing and analysis*, pp. 249-260. ACM, 2008.

[6] Rune Hammersland and Einar Snekkenes. *Fuzz testing of applications*. Faculty of Computer Science and Media Technology Gjøvik University College, Norway

[7] Andrews, Anneliese A., Jeff Offutt, and Roger T. Alexander. "Testing web applications by modeling with FSMs." *Software & Systems Modeling* 4, no. 3 (2005): 326-345.

[8] Clarke, James M. "Automated test generation from a behavioral model." In*Proceedings of Pacific Northwest Software Quality Conference. IEEE Press*. 1998.

[9] Alshahwan, Nadia, and Mark Harman. "Automated web application testing using search based software engineering." In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pp. 3-12. IEEE Computer Society, 2011.

[10] Brashear, Thomas G., Vishal Kashyap, Michael D. Musante, and Naveen Donthu. "A profile of the Internet shopper: Evidence from six countries."*Journal of Marketing Theory and Practice* 17, no. 3 (2009): 267-282.

[11] Halfond, William GJ, and Alessandro Orso. "Automated identification of parameter mismatches in web applications." In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 181-191. ACM, 2008.

[12] Afzal, Wasif, Richard Torkar, and Robert Feldt. "A systematic review of search-based testing for non-functional system properties." *Information and Software Technology* 51, no. 6 (2009): 957-976.

[13] Dogan, Serdar, Aysu Betin-Can, and Vahid Garousi. "Web application testing: A systematic literature review." *Journal of Systems and Software* 91 (2014): 174-201.

[14] Li, Yuan-Fang, Paramjit K. Das, and David L. Dowe. "Two decades of Web application testing—A survey of recent advances." *Information Systems* 43 (2014): 20-54.

[15] Anand, Saswat, Edmund K. Burke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, and Phil McMinn. "An orchestrated survey of methodologies for automated software test case generation." *Journal of Systems and Software* 86, no. 8 (2013): 1978-2001.

[16] Di Lucca, Giuseppe A., and Anna Rita Fasolino. "Testing Web-based applications: The state of the art and future trends." *Information and Software Technology* 48, no. 12 (2006): 1172-1186.

[17] Alalfi, Manar H., James R. Cordy, and Thomas R. Dean. "Modelling methods for web application verification and testing: state of the art." *Software Testing, Verification and Reliability* 19, no. 4 (2009): 265-296.