Search-based software testing of Web applications

A systematic literature review

Anand Bora
North Carolina State University
Raleigh, NC
Email: abora@ncsu.edu

Nirmesh Khandelwal
North Carolina State University
Raleigh, NC
Email: nbkhande@ncsu.edu

Ravi Singh
North Carolina State University
Raleigh, NC
Email: rpsingh3@ncsu.edu

ABSTRACT

This report presents a systematic literature review of the recent (2008-2015) developments in methodologies to automate web application testing. Today, web applications are not only interactive and dynamic but also asynchronous. As our reliance on web applications grows, the need to ensure the quality, security and correctness of web applications grows with it. Testing is a widely-used technique for validating Web applications. As, manual testing is error prone and time consuming, lot of efforts have been put to automate the testing procedures.

This paper reviews the impact of search based software approach on web application testing, automated test input generation and compares search based software testing with concolic testing.

Keywords

Search-based software testing; Concolic testing; Random testing; Test data generation.

1. INTRODUCTION

What is the best way to automate testing of web applications? Search based software engineering (SBSE) is one of the well-known solution to many of software engineering's problems. SBSE proposes that by reformulating a problem as a search–based optimization problem, an optimized solution to the problem may be found.

Web application is a client-server software application in which the client runs in a web-browser. Today, these web applications play an important role on society ranging from corporate world to entertainment world. Web applications are error-prone and are difficult to test, causing serious dependability threats. With the increase in the importance of web application in our lives, testing a web application has become crucial. Manual testing of web applications is one of the approaches to find out bugs and ensure the quality of webs. But, as the manual testing is error prone and tedious, role of automation of testing has become significant.

Web development can be done in different languages each has different characteristics posing challenges while testing web applications. Web technologies like AJAX require testing approaches to tackle the novel problems.

Many researchers have proposed different approaches to automate the testing of web applications. There have been different approaches to achieve the automation of web testing. One of the approaches, Search-based software testing, is the area of focus in this paper. In this study, we present a systematic literature review of the web application testing with the focus on use of search based software testing.

2. BACKGROUND

2.1 Web Application testing

Web application testing is software testing that focuses on web application. The testing is done to verify browser compatibility, operating systems compatibility, web security apart from the normal functionality testing, stress testing, performance testing etc. Different challenges are observed while testing a web application.

2.2 Search-based Software Engineering

The field of search-based software engineering deals with transforming problems into optimization search problems. The 'search' in SBSE is different from the 'search' from other contexts such as textual or hyper textual search.

For SBSE, a search problem is one in which optimal or near optimal solutions are sought in a search space of candidate solutions, then using fitness function the better solutions are distinguished from the worst solutions. Though this approach does not guarantee to get the optimal solution, we can expect to get a near optimal solution.

There is another field related to Search-based domain, i.e., Search-based software testing which is relevant to our study.

2.3 Search-based Software Testing

The field of search-based testing has been systematically review by Afzal et al.[11]. In their systematic literature review it is defined as, "Search-based software testing (SBST) is the application of metaheuristic search techniques to generate software tests. The test adequacy criterion is transformed into a fitness function and a set of solutions in the search space are evaluated with respect to the fitness function using a metaheuristic search technique" [11]. Thus, one can infer that search-based testing is based on the following three components:

- metaheuristic search techniques,
- fitness functions, and
- test suite generation.

2.4 Genetic Algorithm

Genetic algorithms were invented by John Holland in the 1960s. Holland's original goal was not to design application specific algorithms, but rather to formally study the ways of evolution and adaptation in nature and develop ways to import them into computer science. Holland [1975] presents the genetic algorithm as an abstraction of biological evolution and gives the theoretical framework for adaptation under the genetic algorithm.

In order to explain genetic algorithms, some biological terminology needs to be clarified. All living organisms consist of cells, and every cell contains a set of chromosomes, which are strings of DNA and give the basic information of the particular organism. A chromosome can be further divided into genes, which in turn are functional blocks of DNA, each gene representing some particular property of the organism. The different possibilities for each property, e.g. different colors of the eye, are called alleles. Each gene is located at a particular locus of the chromosome. When reproducing, crossover occurs: genes are exchanged between the pair of parent chromosomes. The offspring is subject to mutation, where single bits of DNA are changed. The fitness of an organism is the probability that the organism will live to reproduce and carry on to the next generation. The set of chromosomes at hand at a given time is called a population. Genetic algorithms are a way of using the ideas of evolution in computer science. When thinking of the evolution and development of species in nature, in order for the species to survive, it needs to develop to meet the demands of its surroundings. Such evolution is achieved with mutations and crossovers between different chromosomes, i.e., individuals, while the fittest survive and are able to participate in creating the next generation.

In computer science, genetic algorithms are used to find a good solution from a very large search space, the goal obviously being that the found solution is as good as possible. To operate with a genetic algorithm, one needs an encoding of the solution, i.e., a representation of the solution in a form that can be interpreted as a chromosome, an initial population, mutation and crossover operators, a fitness function and a selection operator for choosing the survivors for the next generation.

2.4 Fitness Function

A fitness function is a particular type of objective function that is used to summarize, as a single figure of merit, how close a given design solution is to achieving the set aims. In a search-based approach, the fitness function measures the quality of an individual I within a population P, and is essential to guide the evolution of individuals towards the desired solution.

3. CHALLENGES IN WEB APPLICATION TESTING

3.1 Automated Test Data Generation

One of the key aspects in automating the web application testing is to generate test input data automatically. In paper [2] "Dynamic Test Input Generation for Web Applications", the authors presented an approach for analyzing web applications by generating test inputs for them automatically using information from previous execution.

3.2 Interface Identification

With the rise in widespread, sophisticated, and complex web applications, there is a need of automated techniques to ensure quality for such applications. One fundamental key in these techniques is to identify interface accurately, as the components of a web application communicate extensively via implicitly-defined interfaces to generate customized and dynamic content. In Paper [3] "Precise Interface Identification to Improve Testing and Analysis of Web Applications", the authors proposed a methodology to identify interface using symbolic execution. There had been some other approaches on accurate interface identification. A web application modeling technique by Deng, Frankl, and Wang [9] scans the bytecode of a web application and identifies the names of the IPs accessed by the

application. Web crawling is also used to identify accepted interfaces.

3.3 Seeding Strategies in SBSE

One of the key factors for the efficiency of search-based techniques to generate tests is seeding strategy. In the context of search-based software testing (SBST), the most common case of seeding regards the case when testing targets (e.g., branches to cover) are sought one at a time, as for example in [10]. In paper [8], the authors analyzed the effects of different seeding techniques in the context of search-based testing for object-oriented languages. Their results provide evidence that a good choice of seeding techniques can lead to an overall improvement of the search results. Paper [1] has discussed two approaches for seeding strategy - Static Constant Seeding (SCS) and Dynamically Mined Value (DMV) Seeding used in the search process for Search-based software testing.

3.4 Achieving Greater Code Coverage

Code coverage is a measure of how well the testing has been done. It measures the number of lines, tested or covered through the testing, of code written as part of applications development. In paper [4] titled "Automated Test Data Generation for Coverage: Haven't We Solved This Problem Yet?" the authors have focused on generation of test cases automatically to achieve the greater code coverage. They have analyzed concolic testing and search-based testing strategies along with random testing to find out the best method among them with respect to code coverage.

3.5 Dynamic updating of web pages and dynamic typing

Web development languages such as Python, PHP, and Ruby are dynamically typed where same variables can be treated as string or numeric in the same script. It becomes difficult to decide the type of variables involved in a predicate, posing a problem when deciding which fitness function to use. Paper [1] solved this problem by checking types of variables dynamically at run-time using built-in PHP functions and then directing to the appropriate fitness function.

Ajax is a grouping of technologies which are used web applications. It provides ability to dynamically update web pages partially which makes difficult to test AJAX web applications. In paper [5], termed "Search-Based Testing of AJAX Web Applications", the authors proposed a search-based test case derivation technique for Ajax (called HILL), based on the hill climbing algorithm.

3.6 Search-based Security testing for detecting vulnerabilities in web applications

Web applications are easy to access and easy to deploy but they are also easy to attack. SQL injections are one of the most common web application vulnerabilities. In paper [13], the authors presented a technique, named BIOFUZZ, to automatically detect such vulnerabilities through targeted test generation. They used search-based testing to systematically evolve inputs to maximize their potential to expose vulnerabilities. One other paper which made use of genetic algorithms in the context of SQL injections is by Liu et al. [14]. Their approach was quite different from BIOFUZZ, though, since they aimed to prevent SQL injections, and they adopted search-based techniques only to validate user inputs.

4. LITERATURE REVIEW OF RELATED PAPERS

The papers reviewed are listed in chronological order from 2008 to 2015.

First section consists of the papers published before seed paper, i.e., "Automated web application testing using search-based software engineering" [1] by Nadia Alshahwan and Mark Harman.

BEFORE SEED PAPER (2011)

4.1 Dynamic Test Input Generation for Web Applications

4.1.1 Keywords

a) Concolic Testing

It is a software verification technique that performs symbolic execution, a classical technique that treats program variables as symbolic variables, along a concrete execution (testing on particular inputs) path.

b) Symbolic execution

It is a means of analyzing a program to determine what inputs cause each part of a program to execute.

c) Directed Random Testing

It is a black-box software testing technique where programs are tested by generating random, independent inputs. Results of the output are compared against software specifications to verify that the test output is pass or fail. In case of absence of specifications the exceptions of the language are used which means if an exception arises during test

execution then it means there is a fault in the program.

4.1.2 Motivational Statements

Manual testing requires extensive human effort, which comes at significant cost. Within the domain of automated web application testing, this paper focuses on automatic test case generation, automatic test input generation by modelling string operations, and checking string values against existing policies to prevent SQL injection attacks. Paper discusses techniques and coding examples for PHP applications.

4.1.3 Related work

Papers [12, 13, 14] discuss on generation of random input values.
Papers [15, 16, 17] are some of the previous works on PHP application testing.

4.1.4 Approach

Constraint generation
The testing framework records a symbolic constraint for each conditional expression that appears in the program's execution.

Constraint resolution
Finite State Machines (FSMs) were used to invert string operations.

Test Oracles
For automatic test input generation to be useful, it test oracle are required that will give feedback on each execution of the program. Typically this feedback takes the form of pass or fail.

Selective Constraint Generation
An iterative approach is proposed to narrow the focus of constraint generation to constraints that are relevant to possible failures.

4.1.5 Future work and recommendations

a) Constraint resolution algorithm discussed in the paper could be enhanced to include multivariate constraints in some cases.
b) The approach suggested in the paper is not fully automated. The web page must be manually loaded (e.g., by clicking "go"), the analyzer must be manually invoked, and analyzer writes the next inputs to a file, so they must be manually provided to the URL. An additional step of automation could be handy to overcome this drawback.

4.2 Search-based testing of Ajax web applications

4.2.1 Keywords

a) AJAX Applications
Short for asynchronous JavaScript and XML. With Ajax, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page.

b) Hill-Climbing Algorithm
It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

c) Finite State Machine (FSM)
It is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

4.2.2 Motivational statements

Author's previous work investigated a state-based testing approach, based on semantically interacting events. The main drawback of this approach is that exhaustive generation of semantically interacting event sequences limits quite severely the maximum achievable length, while longer sequences would have higher fault exposing capability. This paper investigates a search-based algorithm for the exploration of the huge space of long interaction sequences, in order to select those that are most promising, based on a measure of test case diversity, on the hill climbing algorithm.

4.2.3 Related work

Paper [18] discusses on model-based testing of web applications.
Previous work on web application testing has focused on static webpages and the loosely structured control flow between them [19].

4.2.4 Future work and recommendations

a) Experiment with alternative search based algorithms should be done and applied to a larger benchmark of Ajax applications.
b) FSM algorithm could be improved further.

4.3 Precise interface identification to improve testing and analysis of web applications

4.3.1 Keywords

a) Domain Constraining Operations

Certain types of operations that we call domain-constraining operations, implicitly constrain the domain of an IP. Examples of these operations are functions that convert an IP value into a numeric value or comparisons of the IP value against a specific value.

b) Interface Domain Constraint (IDC)

The set of constraints place on an accepted interface along a specific execution path an interface domain constraint (IDC). An accepted interface may have more than one IDC associated with it, if different domain-constraining operations are performed on its IPs along different paths.

c) Symbolic execution

It is a means of analyzing a program to determine what inputs cause each part of a program to execute.

4.3.2 Motivational statements

As web applications become more widespread, sophisticated, and complex, automated quality assurance techniques for such applications have grown in importance. Accurate interface identification is fundamental for many of these techniques, as the components of a web application communicate extensively. Current techniques for identifying web application interfaces can be incomplete or imprecise. To address these limitations, paper present a new approach for identifying web application interfaces that is based on a specialized form of symbolic execution.

4.3.3 Related work

Papers [42, 43] discuss on sophisticated heuristics approach to identify interface.
Other research papers [18, 20, 21] deals with developer-provided interface specifications

4.3.4 Approach

Symbolic Transformation

The goal of this step is to transform a web application so that its symbolic execution will provide information about accepted interfaces and interface domain constraints. This is twostep process – introducing symbolic values and replacing domain constraining operations.

Generating Path Conditions

In the second step, their approach generates the web application's PCs by symbolically executing the transformed web application.

Interface Identification

In the third step, their approach identifies accepted interfaces and IDCs by analyzing the PCs and symbolic states generated in previous step.

Selective Constraint Generation

An iterative approach is proposed to narrow the focus of constraint generation to constraints that are relevant to possible failures.

4.3.5 Results

Test input data was generated for five different applications after applying interface identification approach suggested by authors.

Subject	# Size of test suite			
	W_{df}	W_{ee}	*Spi.*	DFW
Bookstore	258,565	10,634	68,304	33,279
Classifieds	47,352	3,968	7,238	10,732
Employee Dir.	627,820	3,772	46,099	54,887
Events	36,448	1,735	4,145	5,566
Average	242,546	5,027	31,447	26,116

Table 1. Test Suit Size

Subject	Block (%)				Branch (%)				Cmd-form (abs.)			
	W_{df}	W_{ee}	*Spi.*	DFW	W_{df}	W_{ee}	*Spi.*	DFW	W_{df}	W_{ee}	*Spi.*	DFW
Bookstore	84.1	87.3	75.6	68.7	55.2	59.7	42.1	34.8	88	737	63	54
Classifieds	81.6	83.7	76.0	66.3	51.3	54.8	41.7	32.3	96	366	99	19
Employee Dir.	83.0	84.6	76.4	69.3	52.9	56.1	42.4	34.9	30	351	22	16
Events	83.5	84.8	76.8	68.2	55.3	57.2	43.9	34.5	37	186	22	16
Average	83.0	85.1	76.2	68.1	53.7	56.9	42.5	34.1	63	410	52	26

Table 2. Test-input generation and coverage

4.4 Automated Test Data Generation for Coverage: Haven't We Solved This Problem Yet?

4.4.1 Keywords

a) Concolic Testing

It formulates the test data generation problem as one of finding a solution to a constraint satisfaction problem, the constraints of which are produced by concolic execution of the program under test. Concolic execution combines symbolic and concrete execution.

b) AUSTIN

A search based tool to generate test data

c) *CUTE*

A concolic tool to generate test data.

4.4.2 Motivational statements

Little work had been done (at the time of writing that paper) to realize the effectiveness of concolic testing and search based testing with complete real world software-applications.
RQs
i) How effective are concolic and search based tools when applied to real world software applications?
ii) How long does it take for concolic and search based tools to achieve certain levels of coverage?

4.4.3 Related work

Authors have mentioned some of the tools based on random testing, concolic testing and search based testing. [44] DART (by Godefroid et al.) , a random testing tool, is different from CUTE as DART does not attempt to solve constraints involving memory locations.
Concolic testing has also been used to search for security vulnerabilities in large Microsoft applications as part of the SAGE tool. [24]

4.4.4 Patterns

While comparing two approaches, one of the best practices to be followed is the comparison should be fair.
Authors have given the importance to that and mentioned how they have tried to avoid any threats to validity of their findings. To address internal validity threats to the experiments, they have used default settings and if not possible, then reasonable values have been used. To address external validity threats, the authors have used a variety of programming styles and sources.

4.4.5 Results

The results show that there are many challenges remaining in making automatic test data generation tools robust and of a standard that could be considered 'industrial strength'.
This is because with the exception of one of the test subjects chosen, neither tool managed to generate test data for over 50% of the branches in each application's code.
Two main challenges pointed by authors are - the tools need to be able to prevent or recover from segmentation faults, so that they may continue the test data generation process to any effect. Secondly, test data generation tools need to become much more heterogeneous in nature.

4.4.6 Future work and recommendations

The authors selected random 12 functions from the 4 test objects to test the effectiveness of two approaches. It could have been better if apart from random 12 functions, they could have tested on some known functions on which both approaches were expected to perform better. It would have given additional evidences of the effectiveness of search based and concolic testing to generate input test data.

AFTER SEED PAPER (AFTER 2011)

4.5 Dynamic Adaptive Search Based Software Engineering

4.5.1 Keywords

a) Hyper heuristic approach

Normal heuristics operate on search space of potential solutions. Hyper Heuristics operate on search space of heuristics.

b) Self adaptive software systems

Software systems that have SBSE integrated within the systems that allows them to fix faults and cope with anomalies, provides on-line additivity to meet new challenges, environments and platforms.

4.5.2 Motivational statements

The main motivation behind this paper is to present a proposal of applying Search based software engineering practices to create self-adaptive software systems. The author starts by referring many examples where SBSE is proved useful in the past: Requirements optimization, Predictive Modelling, search based testing for Non-Functional Properties, Program Comprehension, search–based software design and search based automated Testing. - The main challenge according author is following: Although SBSE has been successfully applied in isolation to the above mentioned phases of software development process; there is a need to develop holistic SBSE approach which can allow all of these techniques to apply in co-ordination. The programmers will not need to devise the search algorithms for different phases.
Instead of designing bespoke optimization algorithms for specific instances, we advocate the design of 'reasonably good' hyper heuristic optimizers that have the generality to be applied more readily 'out of the box'.

Self-additivity has been a goal of software and systems engineering research for some time and author believes that SBSE will allow adding some value to this grand challenge.

4.5.3 Patterns

The authors provide a great insight into the role of Experimentation vs Empirical results in relation to computer science. Experimentation is something that scientists do "under the laboratory conditions" in order to observe cause and effect of different quantities. On the other hand, empirical results are any statement about the world that is related to observation or experience.
In order to make experimentations in software engineering, people often use synthetic data. Author makes a cautious note that such data should be used only for finding answers that are not clear from real data. It should not be a "substitute", rather it should be "augmentation" to real data. In empirical software

engineering we need both laboratory controlled data and data based on real world empirical experimentation, not one or the other.

4.5.4 Future work and recommendations

a) No concrete examples present in paper to support the author's claims about Adaptive Automated Software Engineering. Although the purpose of the paper is to introduce this new idea that can be worked upon by other researchers in the future, it would have been nice to give some pointers on the required work done by the authors.
b) Not clear how the author plans to achieve hypothesis. Author provides details about the work done so far and related work. But there is no mention on techniques to consolidate that work into holistic system. This question is somewhat kept open ended by authors.

4.6 The Seed is Strong: Seeding Strategies in Search-Based Software Testing

4.6.1 Keywords

a) Genetic Algorithm

It is an evolutionary algorithm (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection and crossover.

b) Seeding

Seeding refers to any technique that exploits previous related knowledge to help solve the testing problem at hand.

c) EVOSUITE

An advanced tool based on Genetic Algorithm, featuring for example the whole test suite optimization approach to test data generation, bloat control techniques and effective assertion generation through mutation testing.

4.6.2 Motivational statements

Authors of the paper mentioned that search-based techniques had been shown to be a promising approach to tackle many kinds of software engineering tasks, particularly software testing. They claimed that those techniques were not adopted by practitioners because of limitations like efficiency and applicability. So, investigating those techniques was of practical value.

4.6.3 Related work

The authors discussed few papers on seeding strategies to improve the search. Some of the papers/work they mentioned was:

a) Angdon and Nordin studied a seeding strategy in order to improve the ability of a classifier/repressor to generalize. [46]

b) In the context of testing real-time systems to find worst case execution times, Tlili et al. applied seeding strategies.
They also discussed some of the work related to GA. Like, a) Miraz et al. created the initial population by selecting the best individuals out of a larger pool of randomly generated individuals. [47]

4.6.4 Future work and recommendations

Authors considered several seeding strategies, and applied them to the context of testing object oriented code in terms of the EVOSUITE tool. Authors mentioned that further seeding strategies are possible, and these as well as investigations of how individual seeding strategies interact will be part of their future work.
a) The paper claims that seeding strategies help EVOSUITE to achieve higher code coverage. But, it is possible that there exist parameter combinations for seeding strategies and search budget that can perform better. So, different parameter settings should be tried.
b) Not clear how the author plans to achieve hypothesis. Author provides details about the work done so far and related work. But there is no mention on techniques to consolidate that work into holistic system. This question is somewhat kept open ended by authors.

4.7 Automated design of algorithms and genetic improvement: contrast and commonalities

4.7.1 Keywords

a) Automated Design of Algorithms

A methodology for using SBSE for discovering and updating the algorithms for computational problems

b) Genetic Programming (GP)

Specialization of genetic algorithms where each individual is a computer program. Computer programs are represented as Abstract Syntax trees and these trees are mutated to generate new programs.

c) *Genetic Improvement (GI)*

GI is methodology where we directly apply SBSE techniques on the source code. It treats software code as genetic material and tries to mutate it to improve the program for a given objective. These operations may for example consist of copying, deleting and swapping lines of code.

4.7.2 Motivational statements

The main motivation behind the paper is to compare and contrast the ADA and GI. Both of these techniques have been used to design the new algorithms automatically using SBSE. Although these techniques have some overlap, there is little fundamental difference in the approaches. The author wants to give us more insight for both of these techniques so that it will clear things out for future research.

4.7.3 Patterns

The authors provide a good pattern for Altering and adjusting the algorithms to solve new problems. These techniques can be used as general pattern for automatic design of Algorithms. Replace the components of existing algorithm by the component from some fixed set reordering: Depending on the restrictions, algorithms can be reordered for initializing, return statements. Parameter tuning: Changing the amount of any or each component. This should be used in association with the above two techniques, so that newly generated algorithm is tested against tuned parameters. When the algorithm being designed is a search or an optimization algorithm the main objective of the fitness function is typically to improve a functional property that is evaluated as the fitness of the best solution found by the evolved algorithm

4.7.4 Future Work and recommendations

In the last section of paper, author provides a compare and contrast between ADA and GI. The author mainly describes the types of problem that both algorithms try to solve, and which one to choose for a gives type of problem. This can act as a base for the future work in the domain.
The most notable difference is that GI is applied in-situ or directly to the source code while ADA works ex-situ, i.e. evolves a function that is injected into the original code.
GI makes small changes, sometimes many small changes that do not have to be constrained to a small region of the source code. ADA's improvement of an existing program on the other hand is a replacement of a certain call or statement in the source and is thus limited to the places where that call is made.

5. CONCLUSIONS AND FUTURE WORK

Software quality is an expensive problem, and the need for automated techniques for defect repair is pressing. Existing work in the area is promising, receiving attention from various research communities. In this paper we review a set of related search based testing algorithms, adapted for web application testing and various other applications of SBSE in the field of testing and debugging. As a relatively new area of research, SBSE does not yet experience broad industry acceptance. In the context of SBSE use in fixing or improving programs, developers need to be confident that any automatically produced modification does not generate unexpected behavior outside the scope of a system's requirements and testing environment. Considering that fully automated programming has yet to be achieved, a desirable property of such modifications would be that they need to be easily understood by humans to support maintenance activities.

The research results summarized in this paper, suggest that the prospects for automated software repair are promising. Transforming research results into practicality, however, raises important challenges, which we believe should guide future work.

Future work would comprise of: [25]

1. Real-world practicality: How can we transform automatic software repair research into widely used real-world software maintenance techniques?

2. Theory of unsound repair methods: The success of existing approaches, particularly those that are unsound or stochastic. The empirical results raise questions about the nature of extant software, its robustness in the face of random modifications, and how robustness and evolvability can be leveraged and enhanced.

3. Automatically finding or generating code that is likely to repair software defects: Beyond the challenge of identifying good repair locations, it is also desirable to understand, formalize, and automatically predict how best to make a repair, addressing the fix localization problem. Progress in this area could increase the applicability and scalability of automated repair techniques, and it might improve our understanding of bug repair in general.

4. Automatic generation of full test cases: Repair techniques use testing to guide a search or measure acceptability. Test suites, found much more commonly in practice than formal specifications, serve as a proxy for complete specifications and are used by many program analysis techniques.

5. Fully automated software development: Beyond the goal of automated software repair lies the challenge of fully automating the entire software development process, including synthesizing complex programs from scratch using unsound techniques such as genetic programming.

6. ACKNOWLEDGMENTS

Our thanks to Dr. Menzies and teaching assistant Rahul Krishna for their guidance and support.

7. REFERENCES

[1] Alshahwan, N., & Harman, M. (2011, November). 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.

[2] Wassermann, G., Yu, D., Chander, A., Dhurjati, D., Inamura, H., & Su, Z. (2008, July). Dynamic test input generation for web applications. In *Proceedings of the 2008 international symposium on Software testing and analysis* (pp. 249-260). ACM.

[3] Halfond, W. G., Anand, S., & Orso, A. (2009, July). Precise interface identification to improve testing and analysis of web applications. In *Proceedings of the eighteenth international symposium on Software testing and analysis* (pp. 285-296). ACM.

[4] Lakhotia, K., McMinn, P., & Harman, M. (2009, September). Automated test data generation for coverage: Haven't we solved this problem yet?. In *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009. TAIC PART'09.* (pp. 95-104). IEEE.

[5] Marchetto, A., & Tonella, P. (2009, May). Search-based testing of Ajax web applications. In *Search Based Software Engineering, 2009 1st International Symposium on* (pp. 3-12). IEEE.

[6] Haraldsson, S. O., & Woodward, J. R. (2014, July). Automated design of algorithms and genetic improvement: contrast and commonalities. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion* (pp. 1373-1380). ACM.

[7] Harman, M., Burke, E., Clark, J., & Yao, X. (2012, September). Dynamic adaptive search based software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on* (pp. 1-8). IEEE.

[8] Fraser, G., & Arcuri, A. (2012, April). The seed is strong: Seeding strategies in search-based software testing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (pp. 121-130). IEEE.

[9] Deng, Y., Frankl, P., & Wang, J. (2004). Testing web database applications. *ACM SIGSOFT Software Engineering Notes*, *29*(5), 1-10.

[10] Wegener, J., Baresel, A., & Sthamer, H. (2001). Evolutionary test environment for automatic structural testing. *Information and Software Technology*, *43*(14), 841-854

[11] Afzal, W., Torkar, R., & Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, *51*(6), 957-976.

[12] Cadar, C., Ganesh, V., Pawlowski, P. M., Dill, D. L., & Engler, D. R. (2008). EXE: automatically generating inputs of death. *ACM Transactions on Information and System Security (TISSEC)*, *12*(2), 10.

[13] Lei, Y., & Andrews, J. H. (2005, November). Minimization of randomized unit test cases. In *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on* (pp. 10-pp). IEEE.

[14] Pacheco, C., & Ernst, M. D. (2005). *Eclat: Automatic generation and classification of test inputs* (pp. 504-527). Springer Berlin Heidelberg.

[15] Jovanovic, N., Kruegel, C., & Pixy, E. K. (2010). A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In *Proceedings of the 2006 IEEE symposium on Security and Privacy, Washington, DC, IEEE Computer Society* (pp. 258-263).

[16] Xie, Y., & Aiken, A. (2006, July). Static Detection of Security Vulnerabilities in Scripting Languages. In *USENIX Security* (Vol. 6, pp. 179-192).

[17] Costa, M., Castro, M., Zhou, L., Zhang, L., & Peinado, M. (2007, October). Bouncer: Securing software by blocking bad input. In *ACM SIGOPS Operating Systems Review* (Vol. 41, No. 6, pp. 117-130). ACM.

[18] Ricca, F., & Tonella, P. (2001, July). Analysis and testing of web applications. In *Proceedings of the 23rd international conference on Software engineering* (pp. 25-34). IEEE Computer Society.

[19] Kung, D. C., Liu, C. H., & Hsia, P. (2000). An object-oriented web test model for testing web applications. In *Quality Software, 2000. Proceedings. First Asia-Pacific Conference on* (pp. 111-120). IEEE.

[20] Jia, X., & Liu, H. (2002, November). Rigorous and automatic testing of web applications. In *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002)* (pp. 280-285).

[21] Andrews, A. A., Offutt, J., & Alexander, R. T. (2005). Testing web applications by modeling with FSMs. *Software & Systems Modeling*, *4*(3), 326-345.

[22] Huang, Y. W., Huang, S. K., Lin, T. P., & Tsai, C. H. (2003, May). Web application security assessment by fault injection and behavior monitoring. In *Proceedings of the 12th international conference on World Wide Web* (pp. 148-159). ACM.

[23] Yuan, X., & Memon, A. M. (2007, May). Using GUI run-time state as feedback to generate test cases. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on* (pp. 396-405). IEEE.

[24] Godefroid, P. (2007, November). Random testing for security: blackbox vs. whitebox fuzzing. In Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007) (pp. 1-1). ACM.

[25] Le Goues, C., Forrest, S., & Weimer, W. (2013). Current challenges in automatic software repair. *Software Quality Journal*, *21*(3), 421-443.