International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-12, October 2019

MUTWEB- A Testing Tool for performing Mutation Testing of Java and Servlet Based Web Applications

S. Suguna Mallika, D. Rajya Lakshmi

Abstract: ***Mutation testing is one of the oldest and unique testing techniques to perform white box testing of software applications. Code coverage becoming an increasing concern in the testing cycle of software, mutation testing technique aids in achieving higher code coverage and unearthing more number of errors at the testing site itself. The parameters like the database connectivity, session management, cookie management, are the beginning point of web application testing failures given the heterogeneity aspects associated with the development of a web application. A detailed account on list of available testing tools for performing mutation testing are presented here. A big bundle of mutation testing tools are still available, however they are not focussing on some of the crucial web vulnerabilities like session and cookie management in web apps. In the current work, a tool to perform mutation testing of web applications is developed and tested to see if desired results are occurring. An architecture of the tool is designed is discussed and presented. A brief analysis on results is presented.***

Key Words: mutation testing, automated testing tool, web application testing

I. INTRODUCTION

Web Applications are growing in tandem with the entire e-commerce giant leaping up exponentially every day[1][17]. A large amount of customer retention happens with the credibility of the web application in use. And hence a proper testing is the need of the day when especially small to medium to large businesses are banking on a web application for growing in their business. Typically a web application also has grown from mere information presenting lopsided html pages to a more dynamic software where information flows through between the clients and the server. The incredible discriminating features of a web app from regular software are what make them unusual for the regular kind of testing. Mutation testing, the other negative testing finds an interesting space when it comes to digging the deeper overlooked faults made unknowingly by the developers who end up fixing those vulnerabilities at a whopping cost much later[15].

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
S. Suguna Mallika*, CSE, CVR College of Engineering, Hyderabad, India. Email: suguna.kishore@gmail.com
Dr. D. Rajya Lakshmi, CSE, JNTU-UCEN, Narsaraopet, Guntur, India. Email: rajyalakshmi.cse@jntukucev.ac.in

The idea of mutation testing is to seed in buggy code at known points in the original source file and test it against the regular test suite. If the software thwarts the buggy code and gives a wrong output, then the mutant is said to be killed and there is a strong test case, nevertheless assuring a secure software. However, if the software were to render results exactly the same way like the original code overlooking the faulty code, then there are two things to be ascertained. One is to enhance the test input sample size and check to see if the mutant is overthrown. If the mutant still persists then there is a need to increase the test suite with a new test case and a backward tracing of the faulty code's abnormal behavior till the actual bug is unearthed.

Usually for performing mutation testing of a software, there need to be some mutation operators which serve as appropriate changes in the source code and then the faulty software is tested either manually or through any automated tool. There needs to be some metric to evaluate the effectiveness of the mutation test suite that was applied on the software. However, previous works reveal that mutation operators were not proposed in every vulnerable area of web applications though.

II. LITERATURE SURVEY

The web applications with their heterogeneous nature have many vulnerabilities like cross site scripting[7], broken session management and authentication[16], application logic failures, database connectivity problems, cross browser compatibility[3][8] are some of the worst security related vulnerabilities that dynamic web applications often succumb to while in execution. There are tools to perform load testing and performance testing of web apps but there are not many tools which test all the heterogeneity aspects mentioned afore. In this work, an automated testing tool to perform mutation testing of javascript based web applications is presented. Novelty of this tool lies in its ability to implement mutation operators so far not defined in any of the previous mutation testing works.

There are quite a number of tools for performing mutation testing on various stand alone applications. Tools like MuClipse, PIT, Jumble perform mutation testing on Java based stand alone applications and tools like Mutpy and Cosmic Ray on stand alone python programs[2][14][18][19][20].

However, the number of tools offering mutation based testing to web apps are quite few like WebMuJava, Selenium, HP–QTP, FitNesse, Watir, testComplete, LoadRunner, TestNg, TOSCA, SilkTest, WinRunner[4][5][9][10][11][12][13].

A review of some of the automated testing tools and the type of testing supported by the tools led the survey to some interesting facts that there are not many tools available for testing the non-functional requirements of the web applications like security, performance, availability etc. Most tools focused on testing the functional requirements which otherwise could be brought down to the unit level testing.

MuClipse Tool offers different mutation operators to the tester who can choose from the list provided on the User Interface for launching corresponding test cases. The test cases are run using JUnit and tool displays a mutation score at the end of testing[24].

Jumble and PIT tool is a mutation based testing tools which performs mutation at byte code level[21][23]. Cosmic-ray, Mutpy are mutation testing tools for python based web applications[20][22]. WebMuJava is a mutation testing tool for web applications. It tests mutation operators over web applications with vulnerabilities in link transitions, state management etc[16].

III. Architecture Of Tool

In this work, an automated testing tool named MUTWEB is developed and used for mutation testing of 5 sample open source web apps. Tools architecture is as presented in Fig.1. There is a presentation layer which facilitates selection of operators, a canonical layer which mutates the original code to pieces of mutants and the logic layer which analyses the results and writes log files for the tester's understanding of the results. Further efforts are underway to feed the results of the tool to a machine learning algorithm which analyses the nature of defects and makes a prediction of defects in web applications with precision.

Table-I: Various Testing Tools Currently Available

S.No	Tool Name	Type of Testing Supported	Browser Support	Language Supported	Open Source/ Licensed
1	WATIR	Functional Testing	IE, Chrome, Safari, Firefox	All	Open source
2	Selenium	Functional Testing	IE, Chrome, Safari, Firefox	Java, .NET, Ruby, Perl, PHP	Open source
3	HP-QTP	Functional Testing	IE, Chrome, Safari, Firefox	VB Script	Licensed
4	Fitnesse	Acceptance Testing	N/A	Java, Python, C#,	Open source
5	testComplete	Functional Testing, Unit Testing	IE, Chrome, Safari, Firefox	VBScript, Jscript, Python, Delphi Script, C++ script, C# Script	Licensed
6	Load Runner	Load Testing	Chrome, Safari, IE, Firefox	Java, .NET, JavaScript, HTML scripting	Licensed
7	Test Ng	Integration Testing, Functional Testing, End-End Testing, Unit Testing	N/A	Java	Open source
8	TOSCA	Functional Testing	IE, Firefox, Chrome	Delphi, .NET including WPF, Java, swing/SWT/AWT, VB	Licensed
9	SilkTest	Functional Testing	IE, Firefox	.NET, Java, Swing, SWT, DOM	Licensed
10	WinRunner	Functional Testing	Any Browser	Any web based application	Licensed

S.No	Tool Name	Type of Testing Supported	Browser Support	Language Supported	Open Source/ Licensed
11	ApacheJMeter	Performance Testing, Load Testing	Any Browser	web service	Open source
12	NeoLoad	Load Testing	IE, Firefox, Chrome	ASP, .Net, J2EE, PHP	Licensed
13	LoadUI	Load Testing	Any Browser	Any web based application	Licensed
14	WebLoad	Load Testing	IE, Firefox, Chrome	HTTP/HTTPS (SSL, TLS), WebSocket, PUSH, AJAX, SOAP, HTML5, WebDAV and others.	Licensed
15	WAPT	Load Testing, Stress Testing	IE, Firefox, Chrome and others	Java Script	Licensed
16	Rational Performance Tester	Performance Testing	Any Browser	Any Script, XSS, SOAP	Licensed
17	Testing Anywhere	Functional Testing	IE, Firefox, Chrome	Any Web Based Application	Licensed
18	Qengine	Functional Testing	IE, Mozilla, Firefox	VBScript, Jscript, Python, Delphi Script, C++ Script, C# Script	Licensed(but End- of Sale) announced
19	MUTANDIS	Functional Testing	Any browser	Java Script	Open source
20	ATUSA	Functional Testing	Any browser	Ajax based any script crawling	Open source
21	Crawljax	Navigation Testing	Any browser	Ajax based any script crawling	Open source
22	JSART	Regression Testing	Any Browser	Java Script based any web application	Open source
23	webMate	Regression Layout Testing	IE, Firefox, Chrome and others	VBScript, Jscript, Python, Delphi Script, C++ Script, C# Script	Licensed
24	reAjax	Functional Testing	Mozilla, Firefox	Ajax based scripts	Open source
25	WebVizor	Functional Testing	Any browser	Any Language	Open source
26	Web Portal In Container Testing	Integration Testing	Any browser	Any Script	Open source
27	Veriweb Tool	Navigation Testing	Any browser	JavaScript	Open source
28	WebScarab	Security Testing	IE, Firefox, Chrome and others	Any Script, XSS, SOAP	Open source
29	Acunetix	Security Testing, Penetration Testing	Any browser	Any Script, XSS, SOAP	Licensed
30	Fortify	Security Testing	Any web browser	C#,.NET, Java, ASP	Licensed

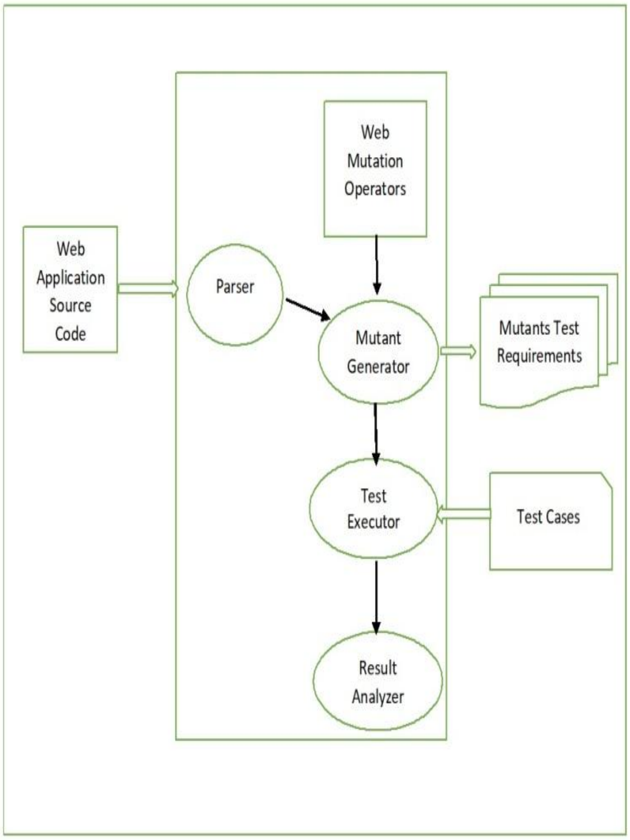

Fig. 1. Architecture of MUTWEB

IV. IMPLEMENTATION

The tool is deployed on apache tomcat server. Initially, the web application under test should be placed in the directory of the testing tool's source code. The MUTWEB's web.xml is updated with the files of the application. The main page of the application is executed. Here, the file to be mutated is given as input and the mutation operator is selected. Some operators require generation of log file before mutation and a section for doing the above process is provided. The log file generation code is inserted into the file and the mutated application must be executed to write log code into a log file. Then another section which also takes input as a file name and type of operator is provided which now applies mutation and modifies the logger code inserted previously. Figure 3 shows the home page of the MUTWEB tool which enables the tester to enter the filename under test.

The application is executed again in order to generate another log file. The file name and the type of operator selected before and applying mutation must be the same. After both the log files are generated, result analyzer compares the contents of both log files is executed. And the status of the mutant is displayed (Live or Dead). After this, the contents of both the log files are cleared. Before mutation is applied, a copy of that file is created and after executing the log checking servlet, the contents of mutated file are updated with its original contents.

The mutation operators applied on the web applications for testing using the current tool are presented in Table-II.

Table-II: Mutation Operators Implemented in MUTWEB

S.No.	Name of the Operator	Description	Category
1	DSID	If a profile URL is tried to be accessed even after logging out of a web application, the user information has to be inaccessible and be redirected to login page again.	Incorrect Session Management
2	DACD	Add Cookie Method Deletion - This operator simply deleted the cookie method from the source code.	Incorrect Cookie Management
3	DHBR	HTTP Boolean Replacement- This will not throw any error but problems might occur while validating the current session	Incorrect Session Management
4	DFIR	Forward Include Replacement - This operator will replace 'forward' with 'include' and vice versa in the following code. But with respect to servlets this operator has not been validated.	Incorrect Session Management
5	DRDUR	Request Dispatcher URL Replacement- Modifying the URL in the code and checking the result with the original code execution. Request Dispatcher method has not been checked for in the previous works so far.	Incorrect Session Management
6	DCD	Close Method Deletion – the conn.close() method is responsible for claiming back the connection resources offered to the client. However if the conn.close() method is removed then the resources continue to be in use without being reallocated for a new connection thus impacting performance of a web application as the number of users accessing it increases.	Incorrect Session Management
7	DSSR	Sessions Set Attribute Name Replacement -	Incorrect Session Management

8	DGSR	Session Name Replacement – Replace the session name in the URL with another previous or some random value and check to see if the contents of the web application are still accessible.	Incorrect Session Management
9	DCDM	Cookie Method Modification	Incorrect Cookie Management
10	BAR	Basic Authentication Replacement	Incorrect Session Management
11	AAR	Advanced Authentication Replacement	Incorrect Session Management
12	XSSC	Cross Site Scripting Check	Cross Site Scripiting Vulnerability
13	DRUR	Modifying the URL- Modifying the Url in the code and checking the result with the original code execution. RequestDispatcherrd=request.getRequestDispatcher("Welcome.html");	Incorrect Session Management
14	DSGD	Servlet based web application's getAttribute function is deleted.	Incorrect Session Management

A. User Interface

Fig. 2 presents the front page where the user is provided with options to enter the filename for applying mutation, selecting the type of mutation operator and option to generate log file before and after mutation.

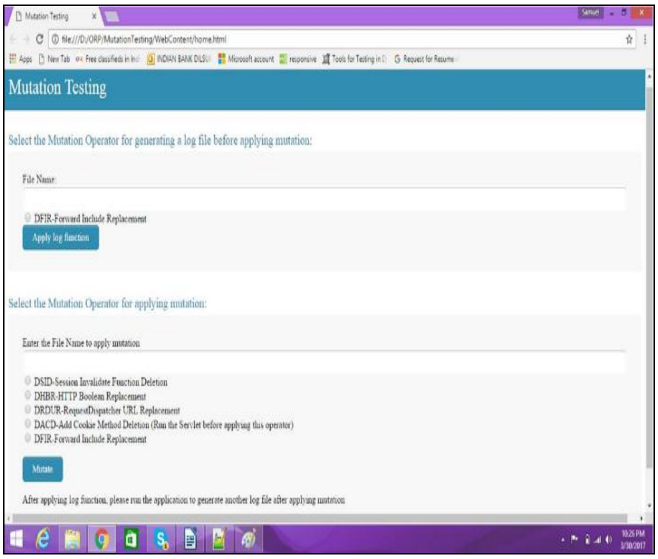

Fig. 2. Home Page of MUTWEB Tool

B. *LoggerServlet*

This Servlet will get the information such as filename and type of operator selected and will add some log code related to the selected operator in the given filename. However, this will not apply mutation but that log code is used to generate a log file which is later used for comparison. After applying log code, its redirected back to the input page. This page will compare the contents of the two log files that are generated and produce an output which tells whether the mutant applied is live or dead. After checking the contents, contents of both the log files are cleared and the mutated code is replaced back with its original code.

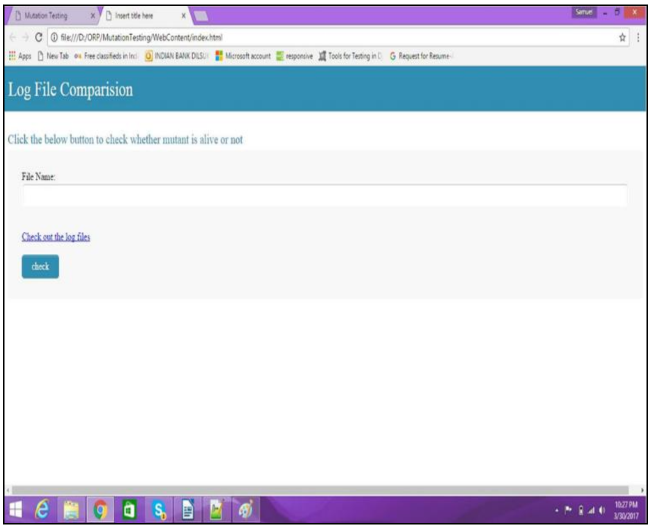

Fig. 3 Invoking Result Analyzer for checking whether mutant is live or dead.

The steps for generating log file, running a servlet before applying mutation are not similar for all operators. For example, before applying the DACD mutation operator, the servlet under test is to be executed before applying mutation. There is no need for applying a test which generated log file before the mutation process. After applying this operator, two log files are generated by the back-end code of DACD operator. After applying this code, the updated servlet is run again. Sample code for generation of log file when trying to execute the DACD mutation operator is provided in Table-III.

C. *Code Snippets*

The code snippet in Table-III presents the sample mutated code generated for the corresponding mutation operator selected by the tester against the web application under test. Similarly, other mutants also would get generated based on operator selected.

Table-III: Sample Code Snippet

```
File log= new File(fn);
Pattern p = Pattern.compile("request.getSession.");
Matcher m1;
{
FileReaderfr = new FileReader(log);
String s;
String totalStr = "";
try (BufferedReaderbr = new BufferedReader(fr)) {
while ((s = br.readLine()) != null) { m1
= p.matcher(s);
if(!m1.find()){ totalStr
+= s;
totalStr += '\n';
}
else{
totalStr += "HttpSession session=request.getSession(true);";
totalStr += '\n';
}
}
FileWriterfw = new FileWriter(log);
fw.write(totalStr);
fw.close();
```

V. RESULT ANALYSIS

Upsorn and Offutt [16] tested their mutation operators on 15 open source web applications that are made available at http://github.com/nanpj. To apply the proposed operators five web applications were picked up in the current work, which are servlet based and were subjected to mutation testing with the proposed operators.

In the current work, web apps under testing are referred to as experiments, where e_i refers to ith experiment. For testing the proposed operators by the authors, only 5 applications namely BSVoting, HLVoting, KSVoting, Conversion and computeGPA are taken into consideration. BSVoting, HLVoting and KSVoting are online voting applications which allows a student to maintain and cast their vote against other user's votes. computeGPA is an application which computes the grade point average of a particular student by accepting their credit hours and grades for the courses the students enrolled. Conversion is a simple webapp which enables users to do online conversion of measurements from one unit to another.

All these experiments are using features that include session management, cookie management, authentication, etc to test our proposed mutation operators. Table-IV lists the experiments along with details of number of lines of code in each web app under test and the total number of components each web app is comprised of.

Table-IV: Subject web apps under Test

Subjects	Components	LOC
BSVoting(E1)	11	930
computeGPA(E2)	1	1619
HLVoting(E3)	12	939
KSVoting(E4)	7	1024
Conversion(E5)	1	388

Every mutation operator does a specific code change at known points and the web application's behavior is studied after the inserted change. If the web app thwarted the change done effectively with its exception handling codes imbibed efficiently, then that particular mutant is killed. If not , we have a susceptibility exposed.

A detailed break up of mutants generated and killed by web mutation adequate tests is presented in Table-VII. The operator wise summary of mutants killed by each of the web mutation adequate tests is provided in Table -VIII.

It is apparent that not all mutation operators help in detecting faults in web apps, but some of them recommend preferred web applications features for developing a better web app and improve the standard of the application. For instance, the experiments under investigation are not employing cookies and suggest that the web developers actually use cookies for better performance of their application. Similarly, the experiments in investigation were found to be using xml files to store and retrieve data instead of a database in the backend. The authors are further exploring other open source web applications which used database connectivity to test some of the operators thereof.

Summary of mutants generated and killed by each and every web app is presented in Table -V.

Table-V: Summary of mutants generated and killed by web mutant adequate tests

Exp #	Mutants	Equivalent	Killed	Tests
E1	43	8	35	10
E2	14	6	8	4
E3	61	18	43	9
E4	54	18	36	10
E5	3	0	3	3

Moreover, the web applications in test, did not use a backend database due to which the proposed operators could not induce mutants into the code. The developers used xml files for storing and retrieving data which affects the security of the application as it is quite easy to edit the xml files by gaining access to them.

Table-VI: Summary of mutants generated by Web Mutation Adequate Tests Operator Wise

Exp	Mutants Generated																			Total
	DSID	DACD	DHBR	DFIR	DRDUR	DCD	DSSR	DGSR	DCDM	BAR	AAR	XSSC	DRUR	DSGD	DPR	DPD	DDNR	DLDR	DSSD	
E1	1	0	1	4	4	0	4	10	0	0	0	1	4	10	0	0	0	0	4	43
E2	1	0	0	0	0	0	6	0	0	0	0	1	0	0	0	0	0	0	6	14
E3	1	0	1	9	0	0	18	1	0	0	0	1	1	11	0	0	0	0	18	61
E4	1	0	1	6	7	0	11	1	0	0	0	1	7	8	0	0	0	0	11	54
E5	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	3

Table –VII: Summary of mutants killed operator wise by web mutant adequate tests

Exp	Mutants Killed																			Total
	DSID	DACD	DHBR	DFIR	DRDUR	DCD	DSSR	DGSR	DCDM	BAR	AAR	XSSC	DRUR	DSGD	DPR	DPD	DDNR	DLDR	DSSD	
E1	1	0	1	4	0	0	4	10	0	0	0	1	4	10	0	0	0	0	0	35
E2	1	0	0	0	0	0	6	0	0	0	0	1	0	0	0	0	0	0	0	8
E3	1	0	1	9	0	0	18	1	0	0	0	1	1	11	0	0	0	0	0	43
E4	1	0	1	6	7	0	11	1	0	0	0	1	0	0	0	0	0	0	0	36
E5	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	3

A mapping of the generated mutants verses the killed mutants is presented in Fig. 3. It is evident that the killed mutants represented a significant number of faults exposed due to the proposed mutation operators. Nevertheless, there are still some operators which could not be floated due to lack of usage of those particular features in the sample case studies taken. For instance, none of the web apps undertaken as case studies implemented backend database connectivity, and cookies as part of development. Sans these operators test suites were written only to test the features falling under the proposed operators' category.

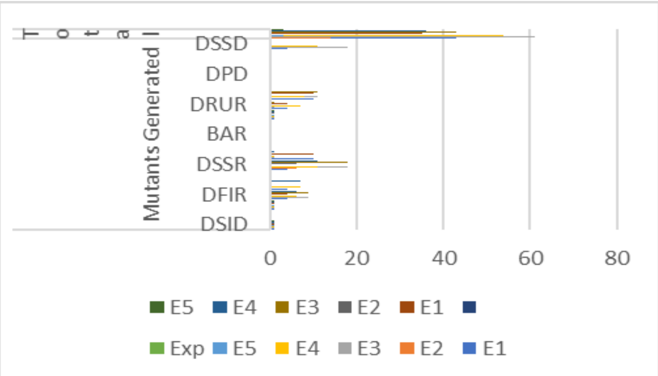

Fig. 1. Mutants Generated Vs Killed Mutants

VI. Conclusions And Future Work

WEBMUT offers a humble beginning of the proposed mutation operators by the authors. In continuance to its purpose behind design, there needs to be more mutation operators added including some more generic operators, language based operators, to make it a generic tool that can be offered across to any web based application and giving flexibility to the tester to choose a set of operators that he would like to apply for a specific application under test.

There can be some more generic operators proposed for web application vulnerabilities like missing plugins, cross browser compatibilities. Typically, one suits all kind of testing suite is the need of the hour to test any kind of web application. Sans functionality testing, integration vulnerabilities, session management, cross browser, plugins, database connectivity are some generic points of vulnerabilities in web applications irrespective of language/framework chosen for development. They need a a more generic set of test cases to be designed for testing the above vulnerabilities which have an indirect bearing on the non functional aspects of web application like performance, security, reliability etc. Further efforts should culminate into a complete and comprehensive test suite for any web application to test its non functional requirements.

The tool is working as per the expectations with which it is built. However it could be further modified to feed the results to a machine learning algorithm which upon taking feed from the tool could make defect prediction in web applications. Then the automated testing of web applications would become an end to end solution for better performance security of web apps. The tool could further be extended to compute the test suite adequacy metric in an attempt to help the testers gain deeper insights into the efficiency of the test suite being employed by them for testing the web apps.

However, other metrics pertaining to mutation testing need further exploration like mutation score computation for statistical evaluation of the web apps under test.

REFERENCES

1. Shakti Kundu. "Web Testing: Tool, Challenges and Methods". IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 3, March 2012. ISSN (Online): 1694-0814.
2. Arora A., Sinha M. "Web Application Testing: A Review on Techniques, Tools and State of Art". International Journal of Scientific & Engineering Research, Volume 3, Issue 2, February-2012 ISSN 2229-5518.
3. Moheb R. Girgis, Tarek M. Mahmoud, Bahgat A. Abdullatif, Alaa M. Zaki. "An Automated Web Application Testing System". International Journal of Computer Applications (0975 – 8887) Volume 99– No.7, August 2014.
4. Nisha Gogna. "Study of Browser Based Automated Test Tools WATIR and Selenium". International Journal of Information and Education Technology, Vol. 4, No. 4, August 2014.
5. Monika Sharma, Rigzin Angmo. "Web based Automation Testing and Tools". (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (1), 2014, 908-912. ISSN : 0975-9646.
6. J. Križani, A. Grguri, M. Mošmondor, P. Lazarevski. "Load testing and performance monitoring tools in use with AJAX based web applications". MIPRO 2010, May 24-28, 2010, Opatija, Croatia.
7. LaShanda Dukes, Xiaohong Yuan, Francis Akowuah. "A Case Study on Web Application Security Testing with Tools and Manual Testing". 978-1-4799-0053-4/13 2013 IEEE.
8. Ali Mesbah, Mukul R. Prasad."Automated Cross-Browser Compatibility Testing". ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA.
9. http://www.cs.utah.edu/~juliana/pub/veriweb-www2002.pdf as on 8/4/17.
10. http://selab.fbk.eu/marchetto/tools/ajax/reAJAX as on 8/4/17
11. Sara Sprenkle, Holly Esquivel, Barbara Hazelwood, Lori Pollock, "WEBVIZOR: A Visualization Tool for Applying Automated Oracles and Analyzing Test Results of Web Applications",
12. Valentin Dallmeier, Bernd Pohl, Martin Burger, Michael Mirold, Andreas Zeller, "WebMate: Web Application Test Generation in the Real World", ICSTW '14 Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops,Pages 413-418 March 31 - April 04, 2014.

13. Hossain Shahriar and Mohammad Zulkernine, MUTEC: Mutation-based Testing of Cross 28 Site Scripting, ICSE Workshop, IEEE, 2009.
14. David Schuler · Andreas Zeller, Javalanche: Efficient Mutation Testing for Java, ESEC-FSE,ACM, August 24–28, 2009.
15. Shabnam Mirshokraie, Ali Mesbah, Karthik Pattabiraman, Guided Mutation Testing for JavaScript Web Applications, IEEE Transactions on Software Engineering, VOL. 41, NO. 5, MAY 2015
16. Upsorn Praphamontripong, Web Mutation Testing, IEEE Fifth International Conference on Software Testing,Verification and Validation, 2012.
17. D. R. Lakshmi and S. S. Mallika, "A Review on Web Application Testing and its Current Research Directions," Int. J. Electr. Comput. Eng., vol. 7, no. 4, p. 2132, 2017.
18. Mallika, S.S.: EATOOS-testing tool for unit testing of object oriented software. Int. J. Comput. Appl. (0975–8887) 80(4), 6–10 (2013).
19. Augsornsri, P., Suwannasart, T.: An integration testing coverage tool for object-oriented software. In: International Conference on Information Science and Applications. IEEE, Seoul (2014). https://doi.org/10.1109/icisa.2014.6847360.
20. Mutpy, https://bitbucket.org/khalas/mutpy last accessed as on 1/10/19.
21. PIT, "http://pitest.org/" last accessed on 1/10/19.
22. Cosmic Ray, "https://github.com/sixty-north/cosmic-ray last accessed as on 1/10/19.
23. Sourceforge, "Jumble" http://jumble.sourceforge.net/, 2007 last accessed as on 1/10/19.
24. B.H. Smith and L. Williams, "An Empirical Evaluation of the Mujava Mutation Operators," in Proceedings of the 3rd workshop on Mutation Analysis(MUTATION '07), published with the proceedings of the 2nd testing. Academic and Industrial Conference Practice and Research Techniques (TAIC PART '07). Windsor, UK: IEEE Computer Society, 10-14 September 2007, pp. 193-202.

AUTHORS PROFILE

Mrs. S. Suguna Mallika obtained her B.Tech in Computer science and Engineering from Nagarjuna University, India, M.Tech in Computer Science from JNTU Hyderabad, India, and currently pursuing her Ph.D in Computer Science and Engineering from JNTU Kakinada, India. She is currently working as an Associate Professor in the department of Computer Science and Engineering at CVR College of Engineering, Hyderabad, India. Her research interests are currently vested in the area of Software Engineering.

Dr. D. Rajya Lakshmi obtained her B.E in Electronics. and M.Tech in Computer Science and Engineering from Andhra University, India and Ph.D from JNTU Hyderabad, India. She is currently Principal JNTU UCEN, Jawaharlal Nehru Technological University Kakinada's constituent college, University college of Engineering , Narsaraopet. Her research interests include Computer Vision, Image Processing, Software Engineering, Data Mining, Network Security and Soft computing.