ISSN 2047-3338

Web Application– A Study on Comparing Software Testing Tools

[1]Dr. S. M. Afroz, [2]N. Elezabeth Rani and [3]N. Indira Priyadarshini
[1,2]Nizam Institute of Engineering and Technology, India
[3]GuruNank Engineering College, India

***Abstract*– Web application script crashes and malformed dynamically generated web pages are common errors and they seriously impact the usability of Web applications. Static analysis tools for webpage validation cannot handle the dynamically generated pages that are ubiquitous on today's Internet. We present a dynamic test generation technique for the domain of dynamic Web application, utilizes both combined concrete and symbolic execution. The technique generates tests automatically, runs the tests capturing logical constraints on inputs, and minimizes the conditions on the inputs to failing tests so that the resulting bug reports are small and useful in finding and fixing the underlying faults. We are implementing the PHP Programming language test using Apollo. This paper presents the survey on static and dynamic testing analysis, also compare the dynamic test generation of DART and Apollo Software web Tools. Our analysis present the Apollo is effective than the existing tools.**

***Index Terms*– Web Application, Testing Tools, Intrusion and Anomaly**

I. INTRODUCTION

WEB applications are one of the increasing growths of classes of software systems in use today. These applications support a wide range of activities including business functions such as product sale and distribution, scientific activities such as information sharing and proposal review, and medical activities such as expert-system-based diagnoses. It is important that Web applications be dependable, but recent reports indicate that in practice they often are not. For example, one study of Web application integrity found that 29 of 40 leading e-commerce sites [3] and 28 of 41 government sites [2] exhibited some type of failure when exercised by a "first-time user".

Several tools for validating Web applications have been created, but most of these focus on protocol conformance, load testing, link checking, and various static analyses Such tools address problems of availability, navigability, and performance faced initially by deployed Web applications, do not directly assist in detecting the failures in meeting functional requirements that have been found to dominate in mature deployed Web applications [2]. To date, tools that do support functional validation do so only by supporting capture replay: the recording of tester input sequences for use in regression testing. Recently, a few more formal approaches for testing the functional requirements of Web applications have been proposed [8], [1], [5]. In essence, these are "white-box" testing approaches, building system models from inspection of code and identifying test requirements from those models.

Early studies have shown that these approaches can facilitate the construction of "adequate" test suites; however, the approaches can also be costly due to the human effort required to generate test cases that meet the identified test requirements. The search for a generalizable and practical approach to the functional testing of Web applications is complicated by several characteristics of those applications. First, Web application usage can change rapidly.

For example, a Web site can be caught by a search engine and suddenly receives hundreds of thousands of hits per day rather than just dozens [4]. In such cases, test suites designed with particular usage profiles in mind may be inappropriate. Second, Web applications typically undergo maintenance at a faster rate than other systems; this maintenance often consists of small incremental changes [7]. To accommodate such changes cost-effectively, testing approaches should be automatable and test suites should be adaptable. Finally, Web applications typically involve complex, multitiered, heterogeneous architectures including Web, application, and database servers, and clients acting as interpreters.

Testing approaches must be able to handle the various diverse components in these architectures. The foregoing characteristics are not unique to Web applications, but they are particularly prevalent, and their effects on testing are particularly acute in this paradigm. Unfortunately, although the recently proposed techniques [6], [1], [5] partially address the third characteristic, the first two characteristics have not yet been addressed.

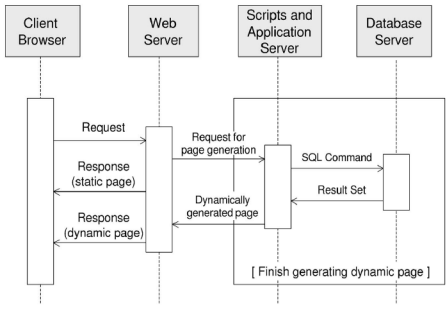

Fig. 1. Shows the seuentail diagram for web application

The goal is to find the failures in web applications in ways:

Execution failures that are manifested as crashes or warnings during program execution and HTML failures that occur when the application generates malformed HTML. Execution failures may occur, for example, when a web application calls an undefined function or reads a nonexistent file. In such cases, the HTML output contains an error message and execution of the application may be halted, depending on the severity of the failure. HTML failures occur when output is generated that is not syntactically well-formed HTML (e.g., when an opening tag is not accompanied by a matching closing tag). HTML failures are generally not as important as execution failures because Web browsers are designed to tolerate some degree of malformedness in HTML, but they are undesirable for several reasons. First and most serious is that browsers' attempts to compensate for malformed web pages may lead to crashes and security vulnerabilities. Second, standard HTML renders faster. Third, malformed HTML is less portable across browsers.

2. INTRODUCTION TO WEB APPLICATION

A web application is an application that uses a web browser as a client. The application can be as simple as a message board or a guest sign-in book on a website, or as complex as a word processor or a spreadsheet. A client-server environment is one in which multiple computers share information such as entering information into a database. The 'client' is the application used to enter the information (web browser like Google, MSN) and the 'server' is the application used to store the information.

A. Benefits of a Web Application

A web application relieves the developer of the responsibility of building a client for a specific type of computer or a specific operating system. Since the client runs in a web browser, the user could be using an IBM-compatible or a Mac. They can be running Windows XP or Windows Vista. They can even be using Internet Explorer or Firefox, though some applications require a specific web browser. Web application uses a combination of server-side script (ASP, PHP, etc) and client-side script (HTML, JAVA script, etc) to develop the application. Web Applications have been around since before the web gained mainstream popularity. For example, Larry Wall developed Perl, a popular server-side scripting language, in 1987. That was seven years before the Internet really started gaining popularity outside of academic and technology circles.

Web applications are preferred over traditional applications for two reasons:

i). Web Applications are more accessible: The HTTP protocol used in web applications is a standard protocol that can travel across corporate firewalls. The only client software a user nee is a web browser. Also, Web Applications are available on many platforms. Web browsers are packaged with most operating systems these days.

ii). Web Applications have a lower maintenance and deployment cost: Since Web applications are running in web browser, they do not depend on installing client software on each user's computer. Web applications can be maintained by modifying code that resides on a server. This reduces the time and the cost of upgrade and deployment of web applications compared to tradinal client/server applications.

B. Web Application Architecture Views

Four views of a Web Application Architecture proposed by Kruchten: Logical, Process, Physical, and Development view.

Each view captures specific decisions and all views must be examined to gain a good understanding of the whole applications. To account additional requirements such as security and requirement, additional views can be added to the Web Application Architecture View.

• *The Logical View:* The Logical view provides a high level abstraction of the system based on the domain of the problem. The application is represented by different components and the interaction between them. At the architectural level, Web Applications can be organized as 2-tiers or as 3 tiers (as shown in Fig. 2).

Presentation Logic tier shows the interaction between the components responsible for the generation of the User Interface. Components in this tier only interact with components in the *Business Logic* tier.

Business Logic tier contains all the knowledge required to modify the data components that are contained in the Database tier.

Database tier contains all the data components that are used to provide persistent storage for the application data.

The 3-tired architecture provides a good separation of concerns. In a 2-tired architecture both business logic and data access code are mixed. The advantage of the 3-tired architecture is the encapsulation of concerns. The database system used for the implementation can easily be changed without affecting the Business Logic (as would be the case for a 2-tier architecture).

• *The Development View:* The Development view focuses on the mapping of the Logical view conceptual components to the actual implementation artifacts. It presents that actual software module organization in the development environment. The Development View for Web Applications must highlight additional details such as:

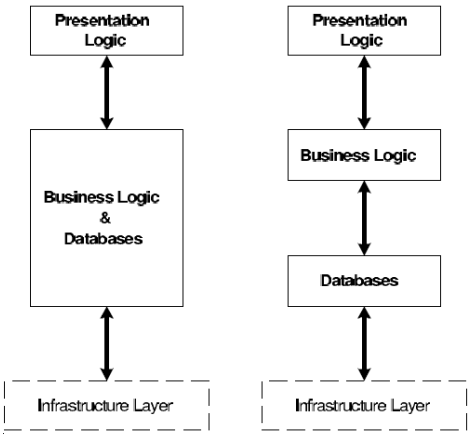

Fig. 2. High level logical view of web application

– The link structure of the application pages
– User's session management techniques
– Application page generation technology

Many web pages are linked together to form a Web Application. To avoid death links all links have to be checked regularly. Web Applications use the HTTP protocol as their communication medium. Since the HTTP protocol is a stateless protocol, the Web Applications have to use techniques such as Cookies or hidden fields in a web page to store the state for a particular session.

Application pages are a mixture of HTML tags and control code. The control code is used to personalize web pages and the HTML tags are used to format the output of the page. When an application page is requested, a web server is preprocessing all data from various resources and generates the final HTML page. The developer has two options for developing application pages: First, he can use *Code based* application pages. With this technique HTML pages are generated fully by executable programs. Some examples of this technology are: CGI and Java-Servlet. Second, the developer can use templated-based application pages.

These are written with the HTML Language and extended with tags to embed control code. Examples of this technology are PHP and Java Server Pages (JSP).

• *The Physical View:* The Physical view presents the mappings of the components in the Development view to the components in the environment. Web applications have a rich environment, which contains the following components:

– Web browsers
– Web servers
– Application servers
– Databases
– Distributes objects (i.e. Enterprise Java Beans)

The user of a Web application uses the web browser as the interface to get access to the Web applications functionality. The browser transmits the user's action to the web server, sending the requests using the HTTP protocol. The web server determines if the request can be fulfilled directly. Otherwise the applications server must be invoked. As can be seen from figure, if the user wants to retrieve some data from the database, the application server must be invoked. Finally, the web server with the possible returns from the application server generates the HTML pages that is returned to the user.

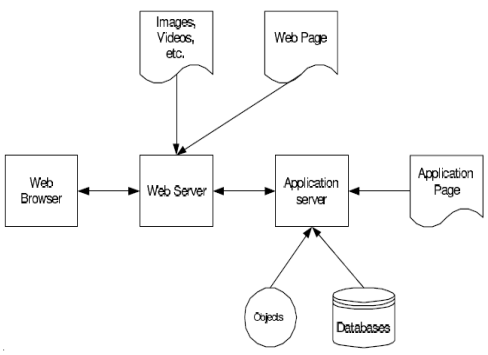

Fig. 3. Physical view of a web application

• *The Process View:* The Process view presents the concurrency and distribution of process in the application.

III. INTRUSION DETECTION SYSTEM

Intrusion detection systems (IDS) process large amounts of monitoring data. As an example, a host-based IDS examines log files on a computer (or host) in order to detect suspicious activities. Network-based IDS, on the other hand, searches network monitoring data for harmful packets or packet flows.

• *What Is Anomaly:* Anomaly detection refers to detecting patterns in a given data set that do not conform to an established normal behavior. The patterns thus detected are called anomalies and translate to critical and actionable information in several application domains. Anomalies are also referred to as outlier, surprise deviation etc.

Most anomaly detection algorithms require a set of purely normal data to train the model and they implicitly assume that anomalies can be treated as patterns not observed before. Since an outlier may be defined as a data point which is very different from the rest of the data, based on some measure, we employ several detection schemes in order to see how efficiently these schemes may deal with the problem of anomaly detection. The statistics community has studied the concept of outliers quite extensively. In these techniques, the data points are modeled using a stochastic distribution and points are determined to be outliers depending upon their relationship with this model. However with increasing dimensionality, it becomes increasingly difficult and inaccurate to estimate the multidimensional distributions of the data points. However recent outlier detection algorithms that we utilize in this study are based on computing the full dimensional distances of the points from one another as well as on computing the densities of local neighborhoods.

The deviation measure is our extension of the traditional method of discrepancy detection. As in discrepancy detection, comparisons are made between predicted and actual sensor values, and differences are interpreted to be indications of anomalies. This raw discrepancy is entered into a normalization process identical to that used for the value change score, and it is this representation of relative discrepancy which is reported. The deviation score for a sensor is minimum if there is no discrepancy and maximum if the discrepancy between predicted and actual is the greatest seen to date on that sensor. Deviation requires that a simulation be available in any form for generating sensor value predictions. However the remaining sensitivity and cascading alarms measures require the ability to simulate and reason with a causal model of the system being monitored.

An appealing way to assess whether current behavior is anomalous or not is via comparison to past behavior. This is the essence of the surprise measure. It is designed to highlight a sensor which behaves other than it has historically. Specifically, surprise uses the historical frequency distribution for the sensor in two ways: It is those sensors and to examine the relative likelihoods of different values of the sensor.

It is those sensors which display unlikely values when other values of the sensor are more likely which get a high surprise scores. Surprise is not high if the only reason a sensor's value

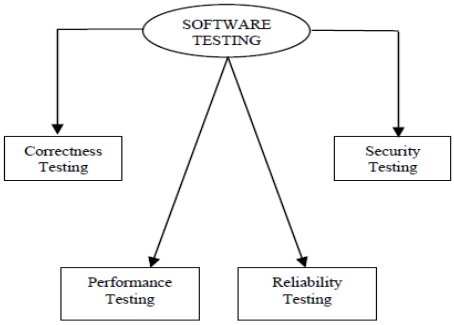

Fig. 4. Software testing techniques

is unlikely is that there are many possible values for the sensor, all equally unlikely.

IV. SOFTWARE TESTING TECHNIQUES

Software testing is a process used to measure the quality of software developed and also the process of uncovering errors in a program and makes it a feasible task. It is useful process of executing program with the intent of finding bugs. Fig. 4 represents the some of the most prevalent techniques of software testing which are classified by purpose [16].

A. Correctness Testing

Essential purpose of testing is correctness which is also the minimum requirement of software. Correctness testing tells the functionality of the system from the wrong one for which it will need some type of Oracle. Either a white box point of view or black box point of view can be taken in testing software as a tester may or may not know the inside detail of the software module under test. For e.g. Data flow, Control flow etc.

B. White Box Testing

White box testing based on an analysis of internal structure of a piece of software. White box testing is the process of giving the input to the system and checking how the system processes that input to generate the required output. It is necessary for a tester to have the full knowledge of the source code. White box testing is applicable at integration, unit and system levels of the software testing process. In white box (Fig. 5) testing one can be sure that all parts through the test objects are properly executed [15], [17].

C. Black box testing

It is an integral part of 'Correctness testing' but its ideas are not limited to correctness testing only. Correctness testing is a method which is classified by purpose in software testing. Black box testing is based on the analysis of the

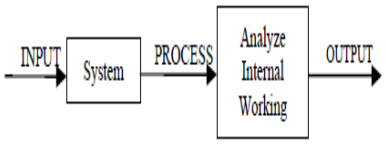

Fig. 5. White-box testing

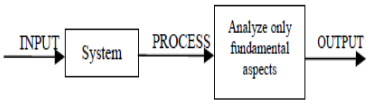

Fig. 6. Black-box testing

specifications of a piece of software without reference to its internal working. The goal is to test how well the component conforms to the published requirement for the component. Black box testing have little or no regard to the internal logical structure of the system, it only examines the fundamental aspect of the system. It makes sure that input is properly accepted and output is correctly produced.

In black box testing (Fig. 6), the integrity of external information is maintained. The black box testing methods in which user involvement is not required are functional testing, stress testing, load testing, ad-hoc testing, exploratory testing, usability testing, smoke testing, recovery testing and volume testing, and the black box testing techniques where user involvement is required are user acceptance testing, alpha testing and beta testing. Other types of Black box testing methods includes graph based testing method, equivalence partitioning, boundary value analysis, comparison testing, orthogonal array testing, specialized testing, fuzz testing, and traceability metrics [15].

D. Grey Box Testing

Grey box testing techniques combined the testing methodology of white box and black box. Grey box testing technique is used for testing a piece of software against its specifications but using some knowledge of its internal working as well [15]. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages. Grey box testing is a process which involves testing software while already having some knowledge of its underline code or logic. The understanding of internals of the program in grey box testing is more than black box testing, but less than clear box testing [18].

V. ANALYSIS OF WEB APPLICATION TESTING TOOLS

Two approaches for testing web applications: static analysis and dynamic analysis. In the context of Web applications, static approaches have limited potential because 1) Web applications are often written in dynamic scripting languages that enable on-the-fly creation of code, and 2) control in a Web application typically flows via the generated HTML text (e.g., buttons and menus that require user interaction to execute), rather than solely via the analyzed code.

Both of these issues pose significant challenges to approaches based on static analysis. Testing of dynamic Web applications is also challenging because the input space is large and applications typically require multiple user interactions. The state of the practice in validation for Web-standard compliance of real Web applications involves the use of programs such as HTML Kit5 that validate each generated page, but require manual generation of inputs that lead to

displaying different pages. We know of no automated tool that automatically generates inputs that exercise different control-flow paths in a Web application, and validates the dynamically generated HTML pages that the Web application generates when those paths are executed.

A. Static Analysis Web Software Tools

Table 1 describes some web software tools.

B. Dynamic Analysis Testing Tools

i) DART: (directed automated random testing) integration of random testing and dynamic test generation using symbolic reasoning is best intuitively explained with an example.

Consider the function h in the file below:

```
int f(int x) f return 2 * x; g
 int h(int x, int y) {
    if (x != y)
    if (f(x) == x + 10)
    abort(); /* error */
    return 0;
 }
```

The function h is defective because it may lead to an abort statement for some value of its input vector, which consists of the input parameters x and y. Running the program with random values of x and y is unlikely to discover the bug. The problem is typical of random testing: it is difficult to generate input values that will drive the program through *all* its different execution paths. In contrast, DART is able to dynamically gather knowledge about the execution of the program in what we call a *directed search*.

Starting with a random input, a DART-instrumented program calculates during each execution an input vector for the next execution.

Table 1: Analysis of web software tools

Product	Comments
AccVerify/AccRepair	Verify, Correct, Monitor and Manage your Web Site and Web Based Applications for W3C and Section 508 compliance.
CSE HTML Validator	HTML, XHTML, CSS, link, spelling, and accessibility checker available. Windows application
Cyber Spyder Link Test	Web site management program to be used for verifying that the URLs on a site are not broken and for analyzing site content. Shareware.
HTML PowerTools	Suite of Windows tools for HTML checking, spelling, etc.
HTML Tidy	A very nice tool that fixes common errors and pretty prints HTML. Free
Link Checker Pro	Link Checker Pro is a link checking tool for websites and has been tested on sites containing more than 100,000 links. It can export results in a number of formats and provide a graphical view of the website structure.
Link Validator	Link checker and site management tool for webmasters to check links for accuracy and availability, find broken links and links containing syntactic errors. A free "lite" version is also offered.
Truwex website QA tool	Free online tool for evaluating website accessibility, privacy, performance, quality, broken links. Shows issues on a web page map and in the HTML code.
WebQA	Report on over 40 errors including, Search engine optimization, Site inventory, Web accessibility (now includes Bobby reports, Section 508, W3C's WCAG), Corporate standards. Can also manage meta data and create automated test scripts to test site functionality.

This vector contains values that are the solution of symbolic constraints gathered from predicates in branch statements during the previous execution. The new input vector attempts to force the execution of the program through a new path.

By repeating this process, a directed search attempts to force the program to sweep through all its feasible execution paths.

For the example above, the DART-instrumented h initially guesses the value 269167349 for x and 889801541 for y. As a result, h executes the then-branch of the first if-statement, but fails to execute the then-branch of the second if-statement; thus, no error is encountered. Intertwined with the normal execution, the predicates x0 ≠y0 and 2, x0 ≠ x0 + 10 are formed on-the-fly according to how the conditionals evaluate; x0 and y0 are *symbolic variables* that represent the values of the memory locations of variables x and y. Note the expression 2.x0, representing f(x): it is defined through an interprocedural, dynamic tracing of symbolic expressions.

The predicate sequence hx0 ≠ y0; 2, x0 ≠x0 + 10i, called a path constraint, represents an equivalence class of input vectors, namely all the input vectors that drive the program through the path that was just executed. To force the program through a different equivalence class, the DART-instrumented h calculates a solution to the path constraint hx0 ≠ y0; 2.x0 = x0 + 10i obtained by negating the last predicate of the current path constraint. A solution to this path constraint is (x0 = 10; y0 = 889801541) and it is recorded to a file. When the instrumented h runs again, it reads the values of the symbolic variables that have been solved from the file. In this case, the second execution then reveals the error by driving the program into the abort() statement as expected.

ii) Apollo: Apollo first executes the Web application under test with an empty input. During each execution, Apollo monitors the program to record path constraints that reflect how input values affect control flow. Additionally, for each execution, Apollo determines whether execution failures or HTML failures occur (for HTML failures, an HTML validator is used as an oracle). Apollo automatically and iteratively creates new inputs using the recorded path constraints to create inputs that exercise different control flow. Most previous approaches for concolic execution only detect "standard errors" such as crashes and assertion failures. This approach detects such standard errors as well, but also uses an oracle to but which are interactively supplied by the user (e.g., by clicking buttons in generated HTML pages).

C. Comparative Study

Comparing the web application testing tools such as DART, and Apollo. DART tool generates test cases by executing the web application on concrete user inputs. This tool is best suitable for testing static web sites and is not suitable for dynamic web applications. The DART needs user inputs for generating the test cases. It is most difficult thing for the human being to provide dynamic inputs for all the possible cases.

Apollo works on dynamic web applications. It can generate dynamic test cases for the dynamic web applications (PHP, ASP).This approach focus on Server-Side-Code and some of client-side through web forms and aims to identify two kinds

of failures of web applications like Execution failures and HTML failures with better result analysis

VI. CONCLUSION

A technique for finding faults in PHP Web applications that is based on combined concrete and symbolic execution. The technique not only detects runtime errors but also uses an HTML validator as an oracle to determine situations where malformed HTML is created. We address the web application issues, such benefits and views, next, bugs and software testing tools, than the analysis of static and dynamic tools and comparison DART, Apollo. Both are dynamic test generation tools but DART shows the good results. In future we can extend this analysis with comparing CUTE, EXE and implementation, results of DART, Apollo web tools.

REFERNCES

[1] C. Liu, D. Kung, P. Hsia, and C. Hsu, "Structural Testing of Web Applications," Proc. 11th IEEE Int'l Symp. Software Reliability Eng., pp. 84-96, Oct. 2000.

[2] Business Internet Group of San Francisco, "The BIG-SF Report on Government Web Application Integrity,"http://www.tealeaf.com/downloads/news/analyst_report/BIG-SF_Report_Gov2003-05

[3] Business Internet Group of San Francisco, "The Black Friday Report on Web Application Integrity," http://www.tealeaf.com/downloads/news/analyst_report/BIG-SF_BlackFridayReport.pdf, Year?

[4] S. Manley and M. Seltzer, "Web Facts and Fantasy," Proc. 1997 Usenix Symp. Internet Technologies and Systems, 1997.

[5] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications," Proc. Int'l Conf. Software Eng., pp. 25-34, May 2001.

[6] G. DiLucca, A. Fasolino, F. Faralli, and U. Carlini, "Testing Web Applications," Proc. Int'l Conf. Software Maintenance, pp. 310-319, Nov. 2002.

[7] E. Kirda, M. Jazayeri, C. Kerer, and M. Schranz, Experiences in Engineering Flexible Web Services," IEEE MultiMedia, vol. 8, no. 1, pp. 58-65, Jan. 2001.

[8] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *ICSE 2000 Limerick Ireland*, 2000.

[9] Guntram Graef and martin Gaedke. An evolution-oriented architecture for web applications. In *Second Nordic Workshop on Software Architectureb(NOSA 1999)*,1999.

[10] Ahmed E. Hassan. Architecture recovery of web applications. Master's thesis, University of Waterloo, 2001.

[11] Grady Booch Ivar Jacobsen and James Rumbaugh. *The Unified Software Development Process*. AddisonWesley, 1999.

[12] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software 12(6)*, 12(6), 1995.

[13] Paul Clements Len Bass and Rick Kazman. *Software Architecture in Practice*.AddisonWesley, 1998.

[14] S. Hansen S. Murugesan, Y. Deshpande and A. Ginige. Web engineering: A new discipline for development of web-based systems. In *Proceedings of the First ICSE Workshop en Web Engineering*, 1999.

[15] Software testing glossary available at http://www.aptest.com/glossary.html#performance testing

[16] Software testing by Jiantao Pan availableat http://www.ece.cmu.edu/~roopman/des-899/sw_testing

[17] White box testing from wikipedia, the free encyclopedia.

[18] Software testing for wikipedia available http://en.wikipedia.org/wiki/grey_box_testing#grey_box_tetsing

[19] R. Alur, P. Cerny, G. Gupta, P. Madhusudan, W. Nam, and A. Srivastava, "Synthesis of Interface Specifications for Java Classes", In *Proceedings of POPL'05 (32nd ACM Symposium on Principles of Programming Languages)*, Long Beach, January 2005.

[20] T. Ball and S. Rajamani. The SLAM Toolkit. In *Proceedings of CAV'2001 (13th Conference on Computer Aided Verification)*,volume 2102 of *Lecture Notes in Computer Science*, pages 260–264, Paris, July 2001, Springer-Verlag.

Professor Dr. Afroz S.M. received Ph.D degree from Jawaharlal Nehru Technological University Hyderabad, India in the faculty of Spatial Information Technology in the year 2009. Completed his M.Tech in Information Technology in the year 2003 and M.Sc in Applied Physics with specialization in quantum opto electronics in the year 1992 from Shri G S Institute of Technology and Science, Indore. Teaching experience spanning over 19 years, presently his job carrier is Dean, Professor and Head, Dept. of CSE at Nizam Institute of Engg. and Technology, Deshmukhi, Near Ramoji film city, Hyderabad, India. Currently his field and research interest in the Digital image processing, Data Warehousing & Data Mining(especially in spatial mining),Neural networks and OOAD. He has 8 research papers to his credit, (smafroz@yahoo.com).

N. Elezabeth Rani pursuing M.Tech (SE) at Nizam Institute of Engg & Tech, areas of interest are Neural Networks, Compiler Design and Web Applications.

N. Indira Priyadarshini pursuing M.Tech (IT) at GuruNank Engg College, areas of interest are Neural Networks, Mobile Computing and Web Applications.