www.ijmer.com Vol.1, Issue.2, pp-707-714 ISSN: 2249-6645

Dynamic Web Page Bug Detection

K. Narahari[1], Dr. M. V. Vijaya Saradhi[2]
[1]PG Scholar, Department of IT, Aurora Engineering College, Andhra Pradesh, India
[2]Professor and Head Of the Department, Department of IT, Engineering College, Andhra Pradesh, India

Abstract:
Now-a-days, most of the web pages are generated by using dynamic web applications which generate pages during their execution. There are no approaches which can detect the failures in dynamically generated web applications without executing them for several times. The Developer of the dynamic web applications has to check for failures only after executing the application in web server. If there are many failures occur, he has to correct them and again executes the application several times. The present tools are capable to test either the java code or the validating HTML content of the web page after the web application executed. There are no tools which can detect failures, execution and HTML failures, at one time. We are proposing such approach which can detect the execution and HTML failures in dynamic web applications before they are going to be executed.

Keywords: Dynamic web applications, HTML Errors, Exceptions, Validation

1. INTRODUCTION

With the wide acceptance of Internet, Web applications have become popular vehicles for conducting transactions, delivering services, and acquiring information. As a result, there is a great demand on sophisticated and high-quality Web applications. However, Web applications are often developed under a short time-to-market pressure without following traditional software engineering discipline [7]. The lacks of rigor, systematic approach, and quality assurance control have raised a concern about the quality and reliability of Web applications in spite of the testing challenges introduced by Web technologies [6].

Recently, several approaches have been proposed to address Web application testing [1, 2, 4, 5, 6]. Most of the approaches focus on testing the architectures of Web Applications and they are not applicable for testing malformed dynamic web pages and web script crashes. These seriously impact the usability of web applications. There are no approaches which can detect the failures in dynamically generated web applications without executing them for several times. If there are many failures occur, he has to correct them and again executes the application several times. The present tools are capable to test either the java code or the validating HTML content of the web page after the web application executed. Thus, we present an approach which can detect the execution and HTML failures in dynamic web applications that are developed using JSP pages, a very popular server-side script language for developing Web applications with Java technology.

JSP pages have been widely used in Java-based Web applications to handle HTTP requests, to interact with Java components like Java beans, and to generate dynamic pages. It is important to ensure that the JSP pages are written correctly and their interactions with other components are handled properly. However, JSP pages usually mix up scripts (i.e., JSP scriptlets) with HTML statements in order to generate dynamic pages. This makes JSP pages difficult to understand and test.

Our goal is to find two kinds of failures in web applications: *execution failures* that are manifested as crashes or warnings during program execution, and *HTML failures* that occur when the application generates malformed HTML. As an example, execution failures may occur 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). Although web browsers are designed to tolerate some degree of malformedness in HTML, several kinds of problems may occur. First and most serious is that browsers' attempts to compensate for malformed web pages may lead to crashes and security vulnerabilities2. Second, standard HTML renders

faster3. Third, malformed HTML is less portable across browsers and is vulnerable to breaking or looking strange when displayed by browser versions on which it is not tested. Fourth, a browser might succeed in displaying only part of a malformed webpage, while silently discarding important information. Fifth, search engines may have trouble indexing malformed pages [5].

www.ijmer.com Vol.1, Issue.2, pp-707-714 ISSN: 2249-6645

Web developers widely recognize the importance of creating legal HTML. Many websites are checked using HTML validators. However, HTML validators can only point out problems in HTML pages, and are by themselves incapable of finding faults in applications that *generate* HTML pages. Checking *dynamic* web applications (i.e., applications that generate pages during execution) requires checking that the application creates a valid HTML page on *every* possible execution path. In practice, even professionally developed and thoroughly tested applications often contain multiple faults.

This paper presents a technique for finding failures in HTML-generating web applications. In our approach, we obtain the JSP pages from the path and separate the HTML Content and Java Code from the web application. Then check for the possible HTML failures by validating HTML content and also check for the possible execution failures, such as, FileNotFound Exception, ClassNotFound Exception and MethodNotFound Exceptions etc in java code. Finally we prepare the failure report based upon the generated failures.

Thus, our approach helps the web application developer to identify the errors easily and reduces his testing time and it also reduces the burden of the web server.

The paper is organized as follows. Section 2 presents an Overview of JSP, introduces our running example, and discusses a class of failures in JSP based web applications. Section 3 presents the approach and also discusses the system implementation. Section 4 gives an overview of related work and Section 5 presents the conclusions.

2. OVERVIEW OF JAVA SERVER PAGES

Java Server Pages:

This section briefly reviews the JSP web applications. JSP stands for Java Server Pages. JSP is one of the most powerful, easy-to-use and fundamental technology for Java web developers. JSP combines HTML, XML, Java Servlets and JavaBeans technologies into one highly productive technology to allow web developers to develop reliable, high performance and platform independent web applications and dynamic websites.

Advantages of JSP:

- Separate the business logic and presentation: The logic to generate dynamic elements or content is implemented and encapsulated by using JavaBeans components. The user interface (UI) is created by using special JSP tags. This allows developers and web designers to maintain the JSP pages easily.
- Write Once, Run Anywhere: as a part of Java technology, JPS allows developers to developer JSP pages and deploy them in a variety of platforms, across the web servers without rewriting or changes.
- Dynamic elements or content produced in JSP can be served in different formats: With JSP we can write web application for web browser serving HTML format. We can even produce WML format to serve hand-held device browsers. There is no limitation of content format which JSP provides.
- Take advantages of Servlet API: JSP technically is a high-level abstraction of Java Servlets. It is now easier to get anything we have done with Servlet by using JSP. Beside that we can also reuse all of our Servlets we have developed so far in the new JSP.

JSP Example:

The code in Figure 1 illustrates the flavour of JSP. The **contentType** attribute of the page directive in JSP specifies the type and the character encoding i.e. used for the JSP response. The default MIME type is "text/html".

The lines 16 to 23 are treated as a scriptlet, where the java code can be written along with HTML content. The lines 17 and 18 get the values of username and password and the line 19 prints the user name. Line 21 sends the control to HomePage.jsp page if the values provided are matched with the given values. This example illustrates how the JSP page can be written.

Failures in JSP:

Our technique targets two types of failures that can be identified from JSP web applications. First, *execution failures* may be caused by a missing the included file, an incorrect database connection, or by an uncaught exception. Such failures are easily identified as the compiler generates an error message and halts execution. Less serious execution failures, such as those caused by the use of deprecated language constructs, produce obtrusive error messages but do not halt execution. Second, *HTML failures* involve situations in which the generated HTML page is not syntactically correct according to an HTML validator. Section 1 discussed several negative consequences of malformed HTML.

www.ijmer.com Vol.1, Issue.2, pp-707-714 ISSN: 2249-6645

```
1  <%@ page language ="java" contentType="text/html"%>
2  <html>
3  <body bgcolor="pink">
4  <form name="f1" method="post">
5      <table>
6          <tr>
7              <td>User Name</td>  <td><input type="text" name="t1" ></td>
8          </tr>
9          <tr>
10             <td>Password</td>   <td><input type="password" name="t2"></td>
11         </tr>
12         <tr>
13             <td></td>  <td><input type="submit" name="b1" value="LogIn"></td>
14         </tr>
15     </table>
16 <%
17 String user=request.getParameter("t1");
18 String pass=request.getParameter("t2");
19     out.println("<h2> Welcome "+user+" </j2>");
20     if(user.equals("abc") && pass.equals("123"))
21     {   response.sendRedirect("HomePage.jsp");
22     }
23 %>
24
25 </form>
26 </body>
```

Figure 1: A Simple JSP Program. This excerpt contains three faults (2 real, 1 seeded)

As an example, the program of Figure 1 contains three faults, which cause the following failures when the program is executed:

1. Executing the program results in an *execution failure*: the file HomePage.jsp referenced on line 21 is missing.
2. The program produces *malformed HTML* because of an unclosed HTML tag in the output.
3. The program produces *malformed HTML* when line 19 generates an illegal HTML tag j2.

The first failure is similar to a failure that our tool found in one of the PHP applications we studied. The second failure is caused by a fault that exists in the original code of the program. The third failure is the result of a fault that was artificially inserted into the example for illustration.

3. PROPOSED SYSTEM AND ITS IMPLEMENTATION

Our approach is to find the failures of HTML and execution failures at a time before the web application is going to be executed. The basic idea behind the technique is to separate the HTML content and java content from JSP page and validate both the contents separately and produce the report.
In our approach, we obtain the JSP pages from the path and separate the HTML Content and Java Code from the web application. Then check for the possible HTML failures by validating HTML content and also check for the possible execution failures, such as, FileNotFound Exception, ClassNotFound Exception and MethodNotFound Exceptions etc in java code. Finally we prepare the failure report based upon the generated failures.

Implementation:

Our technique consists of 2 major components: **Failure Detector** and **Bug Reporter** illustrated in figure 2. This section first provides a high-level overview of the components and then discusses the pragmatics of the implementation.

The inputs to our product are the program under test and an initial value for the environment. The environment of a JSP program consists of the database, cookies, and stored session information.

www.ijmer.com Vol.1, Issue.2, pp-707-714 ISSN: 2249-6645

The **Failure Detector** is responsible for detecting the all JSP pages from the given input and separating the HTML content and java content from JSP pages and sending them to the respective testing components. It has two sub components: Execution Failure Detector and HTML Validator.

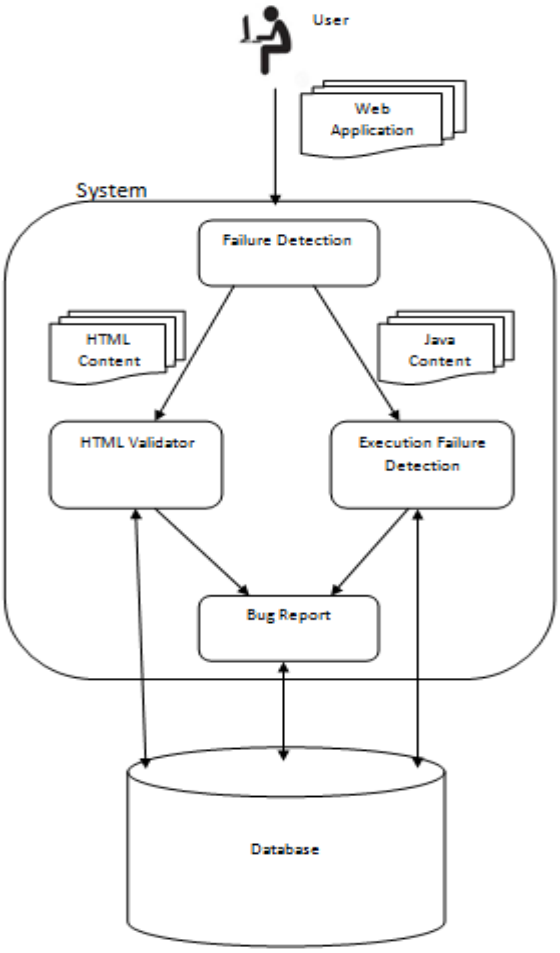

Figure 2: System Architecture

The **Execution Failure Detector** is responsible for checking the possible exceptions like FileNotFoundException, ClassNotFoundException and MethodNotFoundExceptions. These exceptions are the most common exceptions that raised when we are calling a non existing file or class file or calling non existing method.

The **HTML Validator** is responsible for validating the HTML content according to W3C Markup Validation Service [17]. Though we are using offline W3C validation service, the general validation can be done by importing and using the HTML packages.

The **Bug Reporter** is responsible for gathering all the failures that are occurred during testing the JSP pages and managing the report of failures.

a) Failure Detector:

We developed this module that can read the input directory and checks for the JSP pages. After retrieving all the pages, the module analyzes the configuration file and understands the project flow. Then it starts from the starting page and analyzes the code and splits into html and java contents and sends these to two different failure detectors. The HTML Validator is implemented by using Offline W3C Markup Validation Service[17]. The HTML content can also be validated by importing HTML validation packages that are provided by the Java Enterprise Edition API.

The **Execution Failure Detector** is implemented in order to check for the most common exceptions. FileNotFound Exception can be traced by finding out the file name that the jsp page is flowing into and compare with the list of jsp files. If the name is not found the exception is raised. Similarly, after analyzing the *web.xml*,

checking for the class names and comparison of these class names with the list of class files that are placed in /WEB-INF/classes folder can be done. By tracing the class objects, we have analyzed all the possible methods that are present in source files and taken these names into repository and by observing the flow, we have traced the MethodNotFound Exception

b) Bug Reporter:

The bug Reporter is in charge of transforming the results of the executed files into bug reports. Below is a detailed description of the components of the bug reporter.

Bug Report Repository This repository stores the bug reports found in all executions. Each time a failure is detected, the corresponding bug report (for all failures with the same characteristics) is updated with the path constraint and the input inducing the failure. A failure is defined by its characteristics, which include: the type of the failure (execution failure or HTML failure), the corresponding message (JSP error /warning message for execution failures, and validator message for HTML failures), and the statement generating the problematic HTML fragments identified by the validator (for HTML failures), or the JSP statement involved in the JSP Compiler error report (for execution failures). When the exploration is complete, each bug report contains one failure characteristics, (error message and statement involved in the failure) and the sets of path constraints and inputs exposing failures with the same characteristics.

4. EVALUATION AND RESULTS

We experimentally measured the effectiveness of our technique by using it to find faults in JSP web applications. We designed experiments to answer the following research questions:

Q1. How many faults can our technique find, and of what varieties?
Q2. How effective is our technique in terms of the number and severity of discovered faults and the line coverage achieved?
Q3. What are all the types of Failures that our technique can detect?

SCREENSHOTS:

Figure 3 represents a screen shot of our approach while detecting the failures of a given project. The approach not only detects the execution failures but also HTML failures in the project. These HTML failures can be compared by checking the same HTML content with W3C Mark up Validation Service [17].

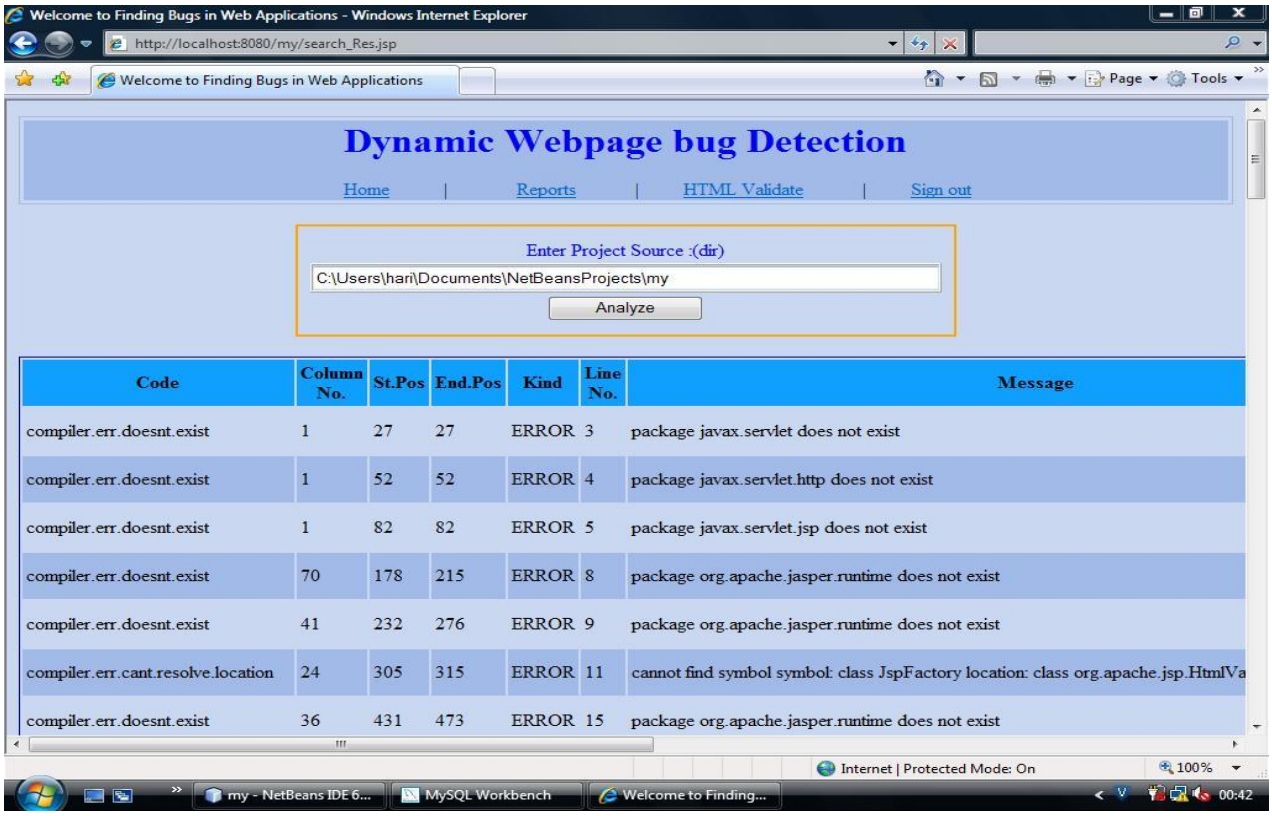

Figure 3: Screen shot of our approach when detecting failures of a project

www.ijmer.com Vol.1, Issue.2, pp-707-714 ISSN: 2249-6645

For the evaluation, we selected 3 open-source JSP web applications:

- **E-Learn**: online training for technical courses.
- **Search optimizer**: a mash-up for comparing Google and Yahoo search engines.
- **Employee management**: managing and maintaining the employee details of an organization through online.

The project details such as files and the JSP lines of code (LOC) is listed in the table 1.

Project	No. Of Files	JSP Lines Of Code
Employee Management	13	538
E-Learn	15	672
Search Optimizer	24	1170

Table 1: Subject Projects

The Table 2 presents results for each project separately. The **Execution failures** and **HTML failures** columns list the number of faults in the respective categories. The **Total failures** columns sums up the number of discovered faults.

Project	**Execution Failures**	**HTML Failures**	**Total Failures**
Employee Management	17	46	63
E-Learn	22	138	160
Search Optimizer	53	303	356

Table 2: Experimental Results

Table 3 tabulates the categories of HTML faults that are detected by our technique. The most commonly detected errors are: Missing end tag, Unopened close tag, Element not allowed, Incorrect attribute, Undefined element etc.

Fault Category	**Faults**	**Percentage**
Element Not Allowed	56	11.5
Missing end tag	157	32.2
No attribute	38	7.8
Unopened close tag	84	17.2
Incorrect attribute	56	11.5
Undefined element	46	9.5
character not allowed	22	4.5
Element declaration finished prematurely	17	3.5
Incorrect attribute value	11	2.3

Table 3: HTML failures found by our technique

Table 4 describes about the execution failures that were detected by our approach for the subjected projects.

Fault Category	Faults	Percentage
File Not Found	29	31.5
Class Not Found	27	29.3
Undefined method	36	39.1

Table 4: Execution failures detected by our approach

LIMITATIONS

We limit our scope of the system as the testing and validating JSP based dynamic web applications is a huge complex task.

JSP scriptlets: We have tested the scriptlets which contain the simple java statements.

JSP tags and Directives: We exempted the jsp directives such as <jsp:bean>, <jsp:forward> etc. As these tags are of HTML type but the uses java features.

Limited sources of input parameters: Apollo currently considers as parameters only inputs coming from the global arrays POST, GET and REQUEST. Supporting other global parameters such as ENV and COOKIE is straightforward.

5. CONCLUSION AND FUTURE WORK

We have presented a technique for finding failures in JSP based dynamic web applications that is based on analyzing the content. The work is novel in several respects. The technique not only detects run-time errors but also uses an HTML validator as an oracle to determine situations where malformed HTML is created. Our approach is the first one to detect failures of a dynamic web application before execution and also without deploying it for several times in order to test the web applications. Though we limited our approach due to the complex nature of Java based dynamic web applications, our technique works fine to find the failures of both execution and HTML.

In future, we try to work on the limitations and to detect more number of failures in both cases and also improve the usability of our technique.

REFERENCES

[1] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst., "Finding Bugs in Web applications using Dynamic Test generation and Explicit State Model Checking". *IEEE Transactions on Software Engineering, July/Aug 2010.*

[2] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. Finding bugs in dynamic web applications. *In ISSTA, pages 261–272, 2008.*

[3] F. Zoufaly. *Web standards and search engine optimization (seo) – does google care about the quality of your markup?, 2008.*

[4] A. Andrews, J. Offutt, R. Alexander, *Testing web applications by modeling with FSMs, Software Systems and Modeling. 4 (3) (2005).*

[5] M.Benedict, J. Freire, P. Godefroid, VeriWeb: Automatically Testing Dynamic Web Sites, *in: Proceedings of the 11th International World Wide Web Conference, May 2002, Hawaii, USA.*

[6] A. Ginige, S. Murugesan, Web engineering: an introduction, *IEEE Multimedia 8 (1) (2001) 14–18.*

[7] R.S. Pressman, Can internet-based applications be engineered? *IEEE Software (1998) 104–110.*

[8] Y. Wu, J. Offutt, Modeling and testing web-based applications, *GMU ISE Technical Report, ISE-TR-02-08, 2002.*

[9] S. Pynchon and K. Garber, "Sarasota's Vanished Votes: An Investigation into the Cause of Uncounted Votes in the 2006 Congressional District 13 Race in Sarasota County, Florida," *Florida Fair Elections Center Report, Jan. 2008.*

[10] T. Kohno, A. Stubblefield, A. Rubin, and D. Wallach, "Analysis of an Electronic Voting System," *Proc. IEEE Symp. Security and Privacy, pp. 27-40, 2004.*

[11] E. Proebstel, S. Riddle, F. Hsu, J. Cummins, F. Oakley, T. Stanionis, and M. Bishop, "An Analysis of the Hart Intercivic DAU eSlate," *Proc. USENIX/ACCURATE Electronic Voting Technology Workshop, 2007.*

[12] A. Yasinsac, D. Wagner, M. Bishop, T. Baker, B. de Medeiros, G. Tyson, M. Shamos, and M. Burmester, "Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware," *technical report, Security and Assurance in Information Technology Laboratory, Florida State Univ., 2007.*
[13] C. Demartini, R. Iosif, and R. Sisto. A deadlock detection tool for concurrent Java programs. *Software — Practice and Experience, 29(7):577–603, June 1999.*
[14] P. Godefroid, A. Kie˙zun, and M. Y. Levin. Grammar-based whitebox fuzzing. *In PLDI, 2008.*
[15] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. *In PLDI, 2005.*
[16] P. Godefroid, M. Y. Levin, and D. Molnar. Automated whitebox fuzz testing. *In NDSS, 2008*.
[17] http://validator.w3.org
[18] H. Cleve and A. Zeller, "*Locating Causes of Program Failures*," Proc. Int'l Conf. Software Eng., pp. 342-351, 2005
[19] Y.Minanide, "Static Approximation of Dynamically Generated Web Pages", Proc. Int'l Conf. World Wide Web 2005.
[20] http://htmlhelp.com/tools/validator/offline.
[21] http://www.htmlkit.com.
[22] G.Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, and Z.Su, " Dynamic Test Input Generation for Web Applications," Proc. ACM/SIGSO
[23] K. Sen, D. Marinov, and G. Agha, *"CUTE: A Concolic Unit Testing Engine for C,"* Proc. ACM SIGSOFT Int'l Symp. Foundations of Software Eng., pp. 263-272, 2005.
[24] F. Zoufaly. Web standards and search engine optimization (seo) – does google care about the quality of your markup?, 2008.
[25] A. Andrews, J. Offutt, R. Alexander, Testing web applications by modeling with FSMs, Software Systems and Modeling. 4 (3) (2005).
[26] M.Benedict, J. Freire, P. Godefroid, VeriWeb: Automatically Testing Dynamic Web Sites, in: Proceedings of the 11th International World Wide Web Conference, May 2002, Hawaii, USA.
[27] A. Ginige, S. Murugesan, Web engineering: an introduction, IEEE Multimedia 8 (1) (2001) 14–18.
[28] R.S. Pressman, Can internet-based applications be engineered? IEEE Software (1998) 104–110.
[29] Y. Wu, J. Offutt, Modeling and testing web-based applications, GMU ISE Technical Report, ISE-TR-02-08, 2002.

About Authors

Mr. K. Narahari received his B-Tech degree from Aurora's Technological Research Institute, Jawaharlal Nehru Technological University (JNTU), Hyderabad, Andhra Pradesh, India. He is currently working as a Assistant Professor in Ramappa Engineering College, Hanamkonda, Andhra Pradesh, India. He also worked as a Junior Research Fellow (JRF) in Indian Institute of Technology (IIT), Kanpur., India His main research interests are Web Engineering, Software Engineering and Object Oriented Modules.

Dr. M.V. Vijaya Saradhi received his Ph.D degree from Faculty of Engineering, Osmania University (OU), Hyderabad, Andhra Pradesh, India. He is Currently Working as Professor in the Department of Information Technology (IT) at Aurora's Engineering College, Bhongir, Andhra Pradesh, India. His main research interests are Software Metrics, Distributed Systems, Object-Oriented Modelling, Mobile Environment, Data Mining, Design Patterns, Object- Oriented Design Measurements and Empirical Software Engineering. He has published more than 20 Research Papers in the International Reputed Journals. He is a life member of various Professional bodies like MIETE, MCSI, MIE, and MISTE.