An Automated Web Application Testing System

Moheb R. Girgis
Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt

Tarek M. Mahmoud
Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt

Bahgat A. Abdullatif
Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt

Alaa M. Zaki
Department of Computer Science, Faculty of Science, Minia University, El-Minia, Egypt

ABSTRACT
Web applications are dynamic and interactive, as compared to traditional applications. Therefore, traditional testing techniques and tools are not sufficient for web applications testing. This paper presents a proposed Web testing approach, in which hyperlinks of the website to be tested are automatically followed one by one to retrieve all HTML texts of its pages starting from the home page. The HTML text of each encountered page is analyzed to extract the needed information about it. Then, the collected information is used in the error checking process. The proposed approach guaranties the satisfaction of two web application testing criteria, namely ***page coverage criterion*** and ***hyperlink coverage criterion***. The paper also describes an automated Web application testing system that has been developed to implement the proposed approach. The effectiveness of the proposed approach and the developed system in discovering several possible Web applications errors is demonstrated through a case study.

Keywords
Web applications testing, Web applications testing approach, Web application testing criteria, Automated web application testing system.

1. INTRODUCTION
A Web application is a system which typically is composed of a database (or the back-end) and Web pages (the front-end), with which users interact over a network using a browser [1]. A web page can be either static, in which case the content is fixed, or dynamic, such that its contents may depend on user input. User input to a web application consists of both navigational requests and data provided often through forms, which eventually affect the state of the underlying code on the server. [2]

Web applications are complex, ever evolving, and rapidly updated software systems. Their testing is both challenging and critical. It is challenging because traditional testing methods and tools are not sufficient for web-based applications, since they do not address their distinctive features. Examples of the new features of web applications are: extensive use of events, rich graphical user interface, and incorporation of server side scripting. Testing web based applications is critical because failure may be very costly. [3]

The main aim of the testing of a Web application is to discover failures in the required services/functionality, in order to verify the conformance of the application behavior with the specified requirements. Web application components are usually accessed by navigation mechanisms implemented by hyperlinks, so link checking must be considered to ensure that neither unreachable component, nor pending/broken links are included in the application. [4]

In this paper, a Web testing approach is proposed and an automated Web application testing system, which implements this approach, is described. In the proposed approach, hyperlinks of the website to be tested are automatically followed one by one to retrieve all HTML texts of its pages starting from the home page. The HTML text of each encountered page is analyzed to extract the needed information about it. Then, the collected information is used in the error checking process.

The paper is organized as follows: Section 2 gives a review of several of web application testing techniques that have been proposed in the literature. Section 3 describes the proposed Web testing approach. Section 4 describes the automated Web application testing system that has been developed to implement the proposed approach. Section 5 presents a case study to illustrate the error exposing ability of the proposed Web testing approach and the developed supporting system. Section 6 presents the conclusion of the work presented in this paper.

2. RELATED WORK
Web applications are dynamic and interactive, as compared to traditional applications. Therefore, traditional testing techniques and tools are not sufficient for web applications testing. A variety of web application testing techniques has been proposed. This section gives a review of several of these techniques.

Kung et al [5] developed a test generation method for web testing to capture structural and behavioral test artifacts of web applications. The entities are represented as objects, and their structures, relationships and dynamic behaviors are described. An object oriented Web Test Model (WTM) has been proposed which represents artifacts from object, behavior and structure features.

Song et al. [6] have proposed a model based on web frameset and browser interactions. Web framesets are employed in applications where layouts of some pages are identical. The process begins with modeling of application with framesets, followed by considering browser interactions along with framesets, modeling the web navigation, formalizing the navigation model, and finally generating and executing test cases.

Lee and Offutt [7] proposed a system that generates test cases using a form of mutation analysis. It focuses on validating the reliability of data interactions among Web-based software components. Specifically, it considers XML based component interactions.

Ricca and Tonella [8] proposed a UML model of Web application for high level abstraction. The model is based entirely on static HTML links and does not incorporate any dynamic aspects of the software. Any Web application can be

seen as an instance of the UML model. The model is supported by a tool that creates a static graph based on HTML links and another that creates tests comprised of sequences of URLs.

Yang et al. [9, 10] presented an architecture for test tools that is directed towards testing Web applications. The architecture consists of five subsystems including test development, test measurement, test execution, test failure analysis and test management.

Benedikt et al. [11] presented VeriWeb, a dynamic navigation testing tool for Web applications. VeriWeb explores sequences of links in Web applications by nondeterministically exploring action sequences, starting from a given URL. Excessively long sequences of links are limited by pruning paths in a form of path coverage. VeriWeb creates data for form fields by choosing from a set of name-value pairs that are initialized by the tester.

Di Lucca et al. [12] suggested a two-stage black box testing approach. The first stage addresses unit testing of a Web application, while the second stage considers integration testing. The scope of a unit test is a single application page, either a client or server page, while the scope of an integration test is a set of Web pages that collaborate to implement an application's use case.

Lei Xu and Baowen Xu [13] introduced a framework for web testing beginning with requirement analysis based on object model, interactive relation model and architectural model, to test case generation via combinatorial method, followed by execution of test cases.

Andrews et al. [14] proposed state machines to model state-dependent behavior of Web applications and to design test cases. In their approach, the process for test generation comprises two phases: in the first phase, the Web application is modeled by a hierarchical collection of FSMs, where the bottom-level FSMs are formed by Web pages and parts of Web pages, while a top-level FSM represents the whole application. In the second phase, test cases are generated from this representation.

Qian et al. [15] have proposed a web testing model that constructs a Page Flow Diagram (PFD) of web pages and transitions followed by conversion into a Page Test Tree (PTT), a spanning tree, from which a test translator is employed to generate test cases, and finally execute them to generate test report.

Turner et al. [16] have proposed an activity oriented technique for automated test code generation. The developer begins by identifying activities in a web application, followed by developing a test activity graph depicting dependent and independent activities. Finally, an activity test algorithm is employed to generate test cases.

Guangzhou Jiang and Shujuan Jiang [17] have presented a quick test model for performance testing, based on testing flow of web applications. They introduced a new performance index called successful request rate in order to test the performance of web application. They also contributed a testing method, combined with LoadRunner tool to provide effective solution to quick performance testing. The quick testing process begins with planning the test, followed by LoadRunner script creation, scenario definition, execution and result analysis.

Li et al. [18] have proposed UML Based approach that models a large web application as hierarchical profile use case diagrams called Use Case Transition Model (UCTM). Traversing the UCTM from top to down, each use case is described as a sequence diagram, which automatically converts it into a Restricted Message on Vertex Graph (RMOVG). A vertex in RMOVG represents one message in sequence diagram. According to Constraint Message Coverage (CMC) criteria, each message must be traversed at least once. Test cases generated from RMOVG satisfy CMC and result in reduced number of test cases.

3. THE PROPOSED WEB TESTING APPROACH

In the proposed Web testing approach, the hyperlinks of the website to be tested are automatically followed one by one to retrieve all HTML texts of its pages starting from the home page. The HTML text of each encountered page is analyzed to extract the needed information about it. Then, the collected information is used in the error checking process. The algorithm, shown in Figure 1, depicts the steps of the proposed approach.

Website Analysis Algorithm
Begin
Input the URL of the website to be tested.
Add this URL to URL_list and mark it as unvisited.
While URL_list contains unvisited links Do
Get first unvisited URL in the URL_list and mark it as visited.
Send an http request to the web server to request the web page that is associated with the given URL.
Download the HTML text for this web page
Analyze HTML Tags in the HTML text as follows:
(i) If the HTML Tag is a href tag, do one of the following actions according to the associated link:
- If link to mail, go to mail links analysis.
- If link to place within the same page, go to internal link analysis
- If link to offsite page, discard it.
- If link to documents, go to files link analysis.
- If link to web page within the host site, add the URL of this page to URL_list and mark it as unvisited, if it is not in the list.

(ii) If the HTML Tag is a form tag go to form analysis.
If there is any cookie go to cookie analysis.
End While
Error Checking
Report Generation.
End

Figure 1: The steps of the proposed web application testing approach

By following the steps of the proposed approach, each page and each hyperlink is visited once. This guaranties the satisfaction of two web application white-box testing criteria, namely ***page coverage criterion*** and ***hyperlink coverage criterion***. These criteria are defined as follows [8]:

Page coverage criterion: every page in the site is visited at least once in some test case.

Hyperlink coverage criterion: every hyperlink from every page in the site is traversed at least once.

4. SYSTEM DESCRIPTION

This section describes the structure of an automated web application testing system that implements the proposed Web testing approach, described in Section 3. The system consists of the following modules, as shown in Figure 2:

- Web navigation module
- HTML analysis module
- Link execution module.
- Error checking module

These modules are described in the following subsections through an example website called addressbook (http://sourceforge.net/projects/php-addressbook/). The features of this website include: Manage contacts & groups, Export Excel, vCard - Import vCard, LDIF (ThunderBird), Show geographical maps of groups, Detects and displays over 12 languages, Displays the next birthdays, Support ActiveSync for iPhone and Android, … etc. The system tests this web site in offline mode.

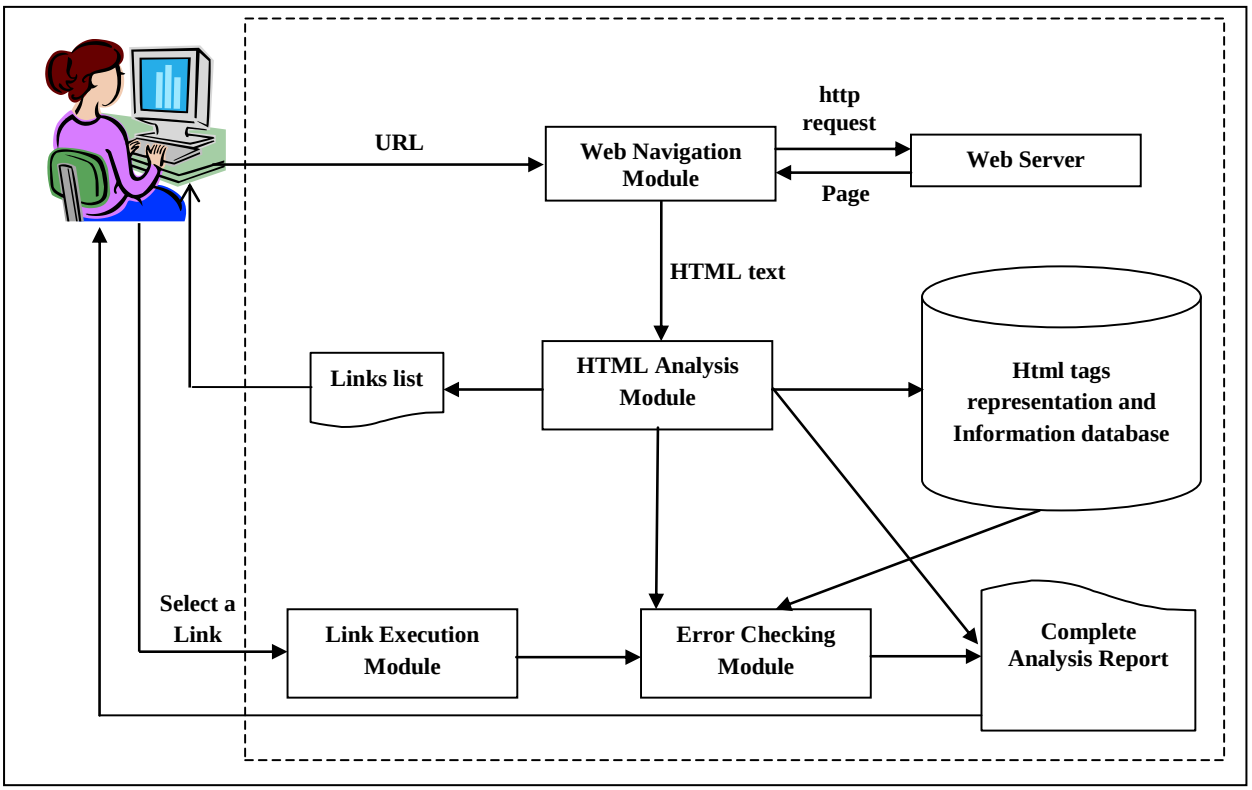
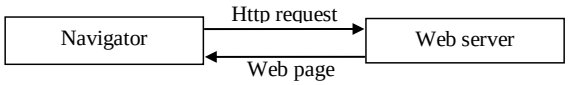


Figure 2: The structure of the proposed automated web application testing system

4.1 Web Navigation Module

This module accepts the address of the website to be tested and crawls through the website to retrieve all HTML texts of its pages.

In the offline mode, to get the HTML text of any page from the web site to be tested, local server (wamp server) is used, and this web site is navigated using localhost with http request as shown in the following figure:



But in the online mode the HTML text of any page is obtained by using its domain name directly. For example, to test the addressbook website example in online mode, the system gets the HTML text directly using the URL http://sourceforge.net/projects/ php-addressbook/, which is given as input to the system to navigate the website and retrieve all HTML texts of its pages.

4.2 HTML Analysis Module

This module takes the HTML text of a web page retrieved by the navigation module as input and analyzes all HTML tags on this page to extract the needed information about them. HTML tags, which can be link tags or form tags, and their related information, are described below.

Link tags: Without links, the World Wide Web wouldn't be a web at all! Link tags are divided into four types

1- Link to a page within the domain: A tag of this type is analyzed to extract the page name and the text of the link.
2- Link to a website outside the domain: A tag of this type is analyzed to extract the URL and the text of the link.
3- Link to send email to a specific address: A tags of this type is analyzed to extract the user and host of the link.
4- Link to download files: A tag of this type is analyzed to extract the file name and the text of the link.

The HTML analysis module produces a list of all links to pages within the domain of the website being tested in both URL and text forms, as shown in Figure 3.

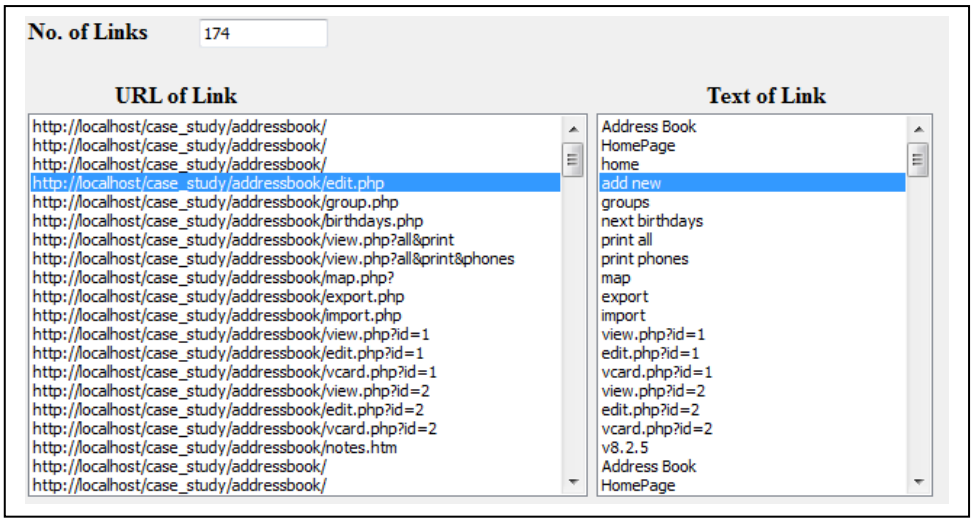


Figure 3: List of all Links of the example website.

Form tags: Forms are used to take input from a user in a web page. The HTML analysis module analyzes each form tag to extract the elements of this form which are described below:

1- Method (GET or POST): Data from a form with method="GET" is posted by appending the data to the end of the script URL, while data from a form with method="POST" is sent as a separate packet to the HTTP server.
2- Action: the page or software that processes the data that entered in the form fields.
3- Input fields: the fields that accept the data from the user. We extract the name, data type, and the value of each field.
4- Submit button: the button that starts processing the form data.

The HTML analysis module produces a list of the forms contained in the website being tested, as shown in Figure 4.

Also, the HTML analysis module analyzes cookies, if any, to extract the needed information about them to test whether:

1. The information in the cookie is encrypted or not.
2. The cookie will expire after session ends or not.

Finally, the HTML analysis module produces a final report containing all extracted information about the website, such as: number of pages, name of these pages, number of links to pages and the text of these links, … etc., as shown in Figure 5.

From this report the user can see the pages or links or document or video names by clicking on the corresponding dropdown list, as shown in Figure 6.

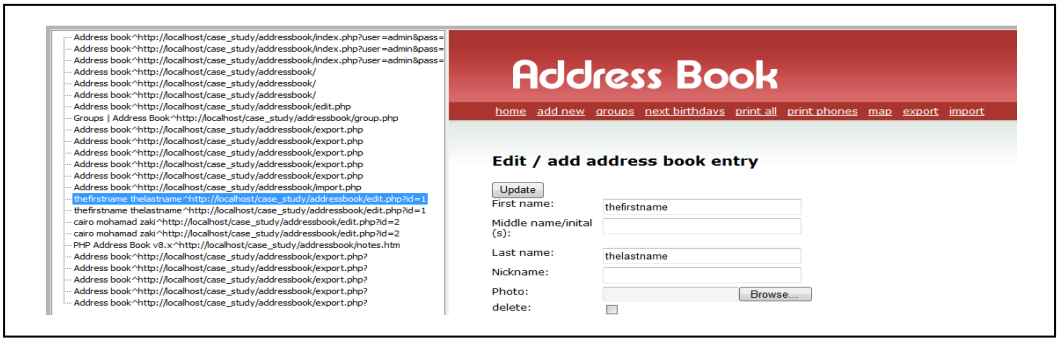


Figure 4: List of form pages within the example web site.

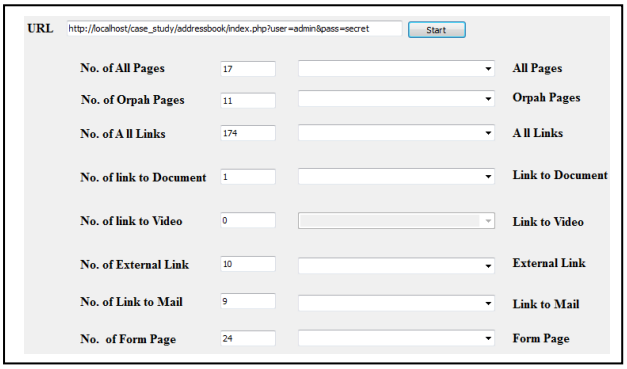


Figure 5: Analysis report of the example web site

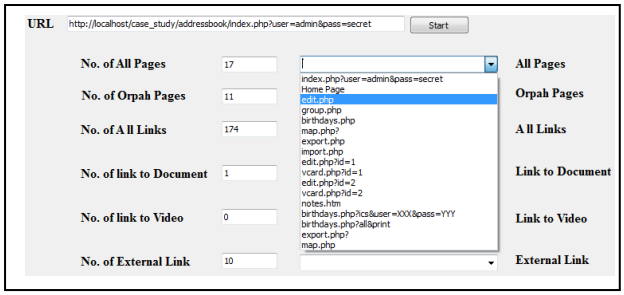


Figure 6: List of pages within the example web site

4.3 Error Checking Module

After analyzing the HTML texts of all web pages and collecting the needed information about every page, the error checking module checks for the following errors:

1- Errors in outgoing links from the current page to any page within the domain.
2- Errors in links from the current page to web pages in an external domain (off-site web pages).
3- Errors in links from the current page to download files.
4- Errors in links from the current page to send e-mails.
5- Errors in links going to other position in the same page.
6- Pages that cannot be accessed (orphan pages) (in offline mode only).
7- Links that cause transition to wrong pages (wrong transition).
8- Errors in cookies (unencrypted or unexpired saved cookies).

This module can detect all the types of errors mentioned in the above list in both offline and online mode except the orphan page error, which can be detected only in offline mode. This error can't be detected in online mode, because in this mode the module can see only all the website pages that are available (i.e., accessible).

In the case of wrong transition error, this module takes the information about all links in the home page (text and URL) as reference, and during the analysis of other pages, if it finds one of these links with same text but different URL, it flags it as a wrong transition error.

At the end, this module produces an error report containing all the detected errors of each error type shown in the above list.

4.4 Link Execution Module

The link execution module allows the user to execute any link individually. From the list of all links in the website, produced by the analysis module, the user can select any link to execute to check whether it is broken or not. This module is useful in regression testing when new pages are added to the website. In this case we can test only the links related to the new pages rather than testing all links. Also, in the error report produced by the error checking module, the link execution module allows the user to select any link to execute and see the result.

5. CASE STUDY

In this section, the error exposing ability of the proposed approach and the developed system will be demonstrated through a case study. In this case study we used the sample website, addressbook.

Several errors were seeded in the web pages of the website, then the erroneous website was presented to the developed system and the generated reports were studied to see whether the system detected the seeded errors or not.

The seeded errors were as follows:

1. Orphan pages error: New web pages test1, test2, test3 and test4 were added to the website.
2. Link to file error: A file name was change from import_sample.csv to import.csv.
3. Broken links error: The paths of two web pages, view.php and csv.php were changed.
4. Internal links error: The Licence and Disclaimer sections in the web page notes.htm were commented.
5. Wrong transitions error: The links in the web page import.php were changed as shown in Table 1.

Table 1: Original links and corresponding wrong links

Original Links	Wrong Links
add new	add new
groups	groups
next birthdays	next birthdays
map	map
export	export
import	import

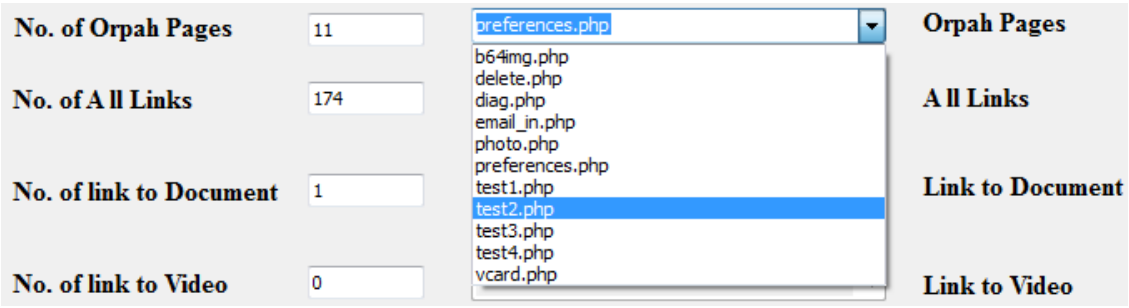


Figure 7: List of the orphan pages within the example web site.

Some of the reports produced by the system for the example Web site are described below:

1) *The analysis report produced by the HTML Analysis Module*, which shows number of pages, name of these pages, number of links to pages and the text of these links, … etc., as shown in Figure 5.. This figure shows, for example, that the number of orphan pages is 11. Clicking the drop down list of orphan pages displays these pages, as shown in Figure 7. As can be seen in this figure, the list includes the newly added pages.
2) *The test reports produced by the error checking module*:
   (i) Links to pages test report, which shows all links to pages in the web site and for each link it shows whether it is broken or not, as shown in Figure 8. From this figure we see that 32 links are broken and all of them refer to the web pages which their paths were changed (Seeded error type No. 3). When any

broken link is selected and executed (by clicking the button test), the "webpage cannot be found" error page is displayed, as shown in Figure 9.

(ii) Links to files test report, which lists the files that cannot be found, as shown in Figure 10. This figure shows the name of the file, import_sample.csv that was changed (Seeded error type No. 2). The word "false" next to it means that the file is not found.

(iii) Internal links test report, which lists all internal links in the web site and true or false next to each link to indicate whether the link is broken or not, respectively, as shown in Figure 11. It shows that the links to Licence and Disclaimer sections are broken, as their sections were commented (Seeded error type No. 4).

(iv) Wrong transition test report, which lists the pages that include links that cause wrong transitions, as shown in Figure 12. In this figure, the pages list contains only one page, import.php, where some links in it were changed (Seeded error type No. 5). When this page is selected, the URL and text of the links that cause wrong transitions in it will be displayed in the linkURL and LinkText lists, respectively, as shown in the figure.

(v) The final error report, which summarizes the errors that are discovered by the system, as shown in Figure 13.

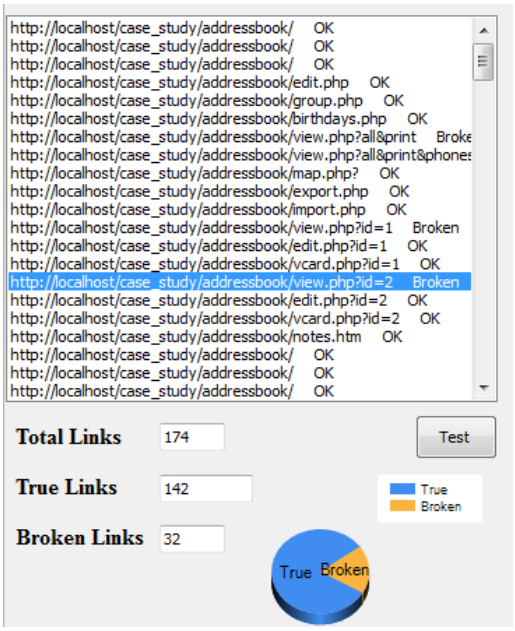


Figure 8: Links to pages test report

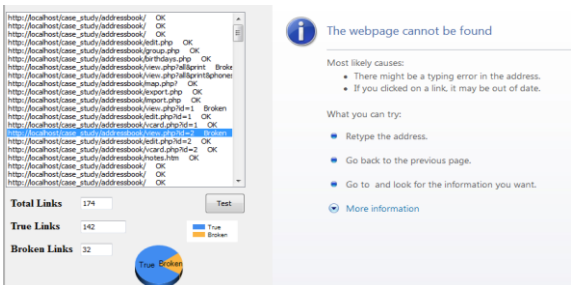


Figure 9: Result of execution of a broken link

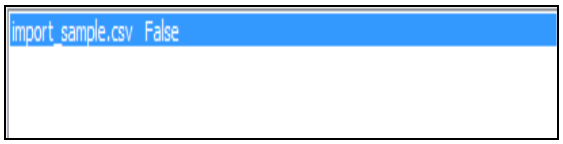


Figure 10: Links to files test report

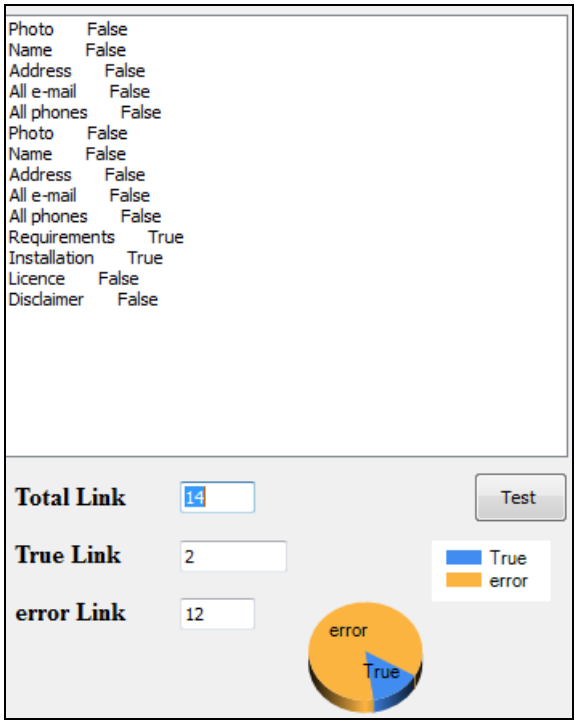


Figure 11: Internal links test report

Figure 12: Wrong transition test report

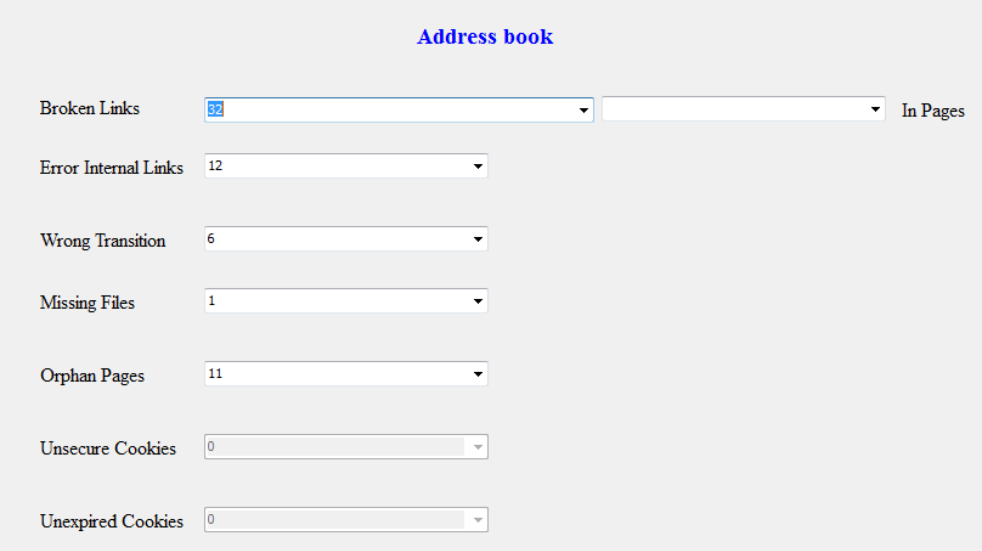


Figure 13: Final error report

Figure 14 shows the seeded errors' types and frequency. All the reports, produced by the system, showed that it has detected all these seeded errors, which demonstrates its effectiveness in Web applications testing.

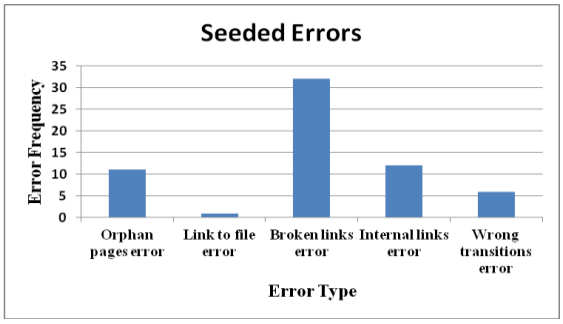


Figure 14: Seeded errors types and frequency

6. CONCLUSION

This paper presented a proposed Web testing approach, in which hyperlinks of the website to be tested are automatically followed to retrieve all HTML texts of its pages starting from the home page. The HTML text of each encountered page is analyzed to extract the needed related information. Then, the collected information is used in the error checking process. The proposed approach guaranties the satisfaction of two web application testing criteria, namely ***page coverage criterion*** and ***hyperlink coverage criterion***.

The paper also described an automated Web application testing system that has been developed to implement the proposed approach. This system consists of 4 modules: *Web navigation*, *HTML analysis*, *Link execution*, and *Error checking*. The Web navigation module retrieves the HTML texts of the pages of the website to be tested. Then, the HTML analysis module analyzes the HTML tags in these texts to get needed information about hyperlinks and forms. Finally, the error checking module uses this information to uncover possible errors in the web site being tested. The system is able to detect several types of errors that may occur in Web applications, such as hyperlinks errors, unreachable (orphan) pages, wrong transition, and cookies errors. The link execution module allows the user to select a particular link from the list of all Links, produced by the HTML analysis module, and executes it to check whether it is broken or not.

The effectiveness of the proposed approach and the developed system in discovering several possible Web applications errors has been demonstrated through a case study. In this case study, several errors were seeded in the web pages of a sample website, then the website was presented to the system. The reports produced by the system showed that it has detected all the seeded errors, which indicates that it is very effective in Web applications testing.

It should be noted that in the presented approach, the execution of Form pages is not considered. The proposed approach is currently being enhanced to test form pages. This requires providing input data for the variables collected through the form, and tracing the processing of these data and the sequence of links to be followed accordingly.

7. REFERENCES

[1] Li, Yuan-Fang, Das, Paramjit K. and Dowe, David L. 2014. Two decades of Web application testing—A survey of recent advances. Information Systems, Vol. 43, pp. 20–54.

[2] Sampath, E., Gibson, S., Sprenkle, S. and Pollock, L. 2005. Coverage Criteria for Testing Web Applications. Technical Report 2005-17, Computer and Information Sciences, University of Delaware.

[3] Mansour, N. and Houri, M. 2006. Testing web applications. Information and Software Technology, Vol. 48, pp. 31–42.

[4] Di Lucca, G. A. and Fasolino, A. R. 2006. Testing Web-based applications: The state of the art and future trends. Information and Software Technology, Vol. 48, pp. 1172–1186.

[5] Kung, David C., Liu, Chien-Hung and Hsia, Pei 2000. An Object Oriented Web Test Model for Testing Web Applications. In: Proceedings of IEEE 24th Annual

International Computer Software and Applications Conference, (COMPSAC2000), Taipei, Taiwan, pp 537–542, October 2000.

[6] Song, Bo, Miao, Huaikou, Chen, Shengbo 2009. Considering Web Frameset and Browser Interactions in Modelling and Testing of Web Applications. International Conference on Computational Intelligence and Software Engineering (CiSE 2009), 11-13 Dec. 2009, pp. 1 – 4, Wuhan.

[7] Lee, Suet Chun and Offutt, Jeff 2001. Generating test cases for XML-based Web component interactions using mutation analysis. In Proceedings of the 12th International Symposium on Software Reliability Engineering, pp. 200-209, Hong Kong China, November 2001, IEEE Computer Society Press.

[8] Ricca, F. and Tonella, P. 2001. Analysis and testing of Web applications. In 23rd International Conference on Software Engineering (ICSE `01), pp. 25-34, Toronto, CA, May 2001.

[9] Yang, J., Huang, J., Wang, F. and Chu, W. 1999. An object-oriented architecture supporting Web application testing. In First Asian-Pacific Conference on Quality Software (APAQS '99), pp. 122-129, Japan, December 1999.

[10] Yang, Ji-Tzay, Huang, Jiun-Long, Wang, Fen-Jian and Chu, William C. 2002. Constructing an object-oriented architecture for Web application testing. Journal of Information Science and Engineering, Vol. 18, No. 1, pp. 59-84, January 2002.

[11] Benedikt, Michael, Freire, Juliana and Godefroid, Patrice 2002. VeriWeb: Automatically testing dynamic Web sites. In Proceedings of 11th International World Wide Web Conference (WWW'2002), Honolulu, HI, May 2002.

[12] Di Lucca, G. A., Fasolino, A. R., Faralli, F. and De Carlini, U. 2002. Testing Web Applications. In: Proceedings of International Conference on Software Maintenance, IEEE Computer Society Press: Los Alamitos, CA, pp 310–319.

[13] Xu, Lei and Xu, Baowen 2004. A Framework for Web Application Testing. International Conference on Cyberworlds, 18-20 Nov. 2004, Tokyo, Japan, pp. 300-305.

[14] Andrews, A. A., Offutt, J. and Alexander, R. T. 2005. Testing Web Applications by Modeling with FSMs. Software Systems and Modeling, Vol. 4, No. 2.

[15] Qian, Zhongsheng, Miao, Huaikou and Zeng, Hongwei 2007. A Practical Web Testing Model for Web Application Testing. Third International IEEE Conference On Signal-Image Technologies And Internet Based System, 16-19 December 2007, Jiangong Jinjiang, Shanghai, China

[16] Turner, David A., Park, Moonju, Kim, Jae hwan and Chae, Jin seok 2008. An Automated Test Code generation Method for Web Applications using Activity Oriented Approach. International Conference on Automated Software Engineering, IEEE Computer Society, Los Alamitos, pp. 411-414.

[17] Jiang, Guangzhu and Jiang, Shujaun 2009. A Quick Test Model of Web Performance Based on Testing Flow and its Application. Sixth Web Information Systems and Applications Conference, 18-20 Sept. 2009, pp. 57–61, Xuzhou, Jiangsu.

[18] Li, Liping, Miao, Huaikou and Qian, Zhongsheng 2008. A UML-Based Approach to Testing Web Applications. International Symposium on Computer Science and Computational Technology, Shanghai, China, pp. 397-40