[1.]S.K. SHINDE, [2.]R.M. MORE

FAULTS FINDING ANALYZER OF WEB APPLICATIONS

[1.]Vidya Pratishthan's Kamalnayan Bajaj Institute of Engg. & Technology, Baramati, Dist. Pune, INDIA
[2.]Shree Datta Polytechnic, Shirol, INDIA

Abstract: Web text and dynamically generated web pages exist common errors, and they seriously affect the usability of Web applications. Current tools to webpage validation cannot manage the dynamically created pages that are ever-present in today's Internet. The proposed tool takes source code files as information sources. It removes HTML, ASP tags and logic codes from extracted files. Then the proposed tool discovers HTML faults and execution faults from these extracted files. The method used to make test cases automatically runs the tests taking constraints from the resource database on inputs and reduces the requirements on inputs to failing tests so that the resulting defect reports are small and important in finding the faults. Our proposed tool implements the technique for Web technologies such as PHP, C#, ASP.net. The tool makes test inputs for a Web application, controls the application for crashes, and confirms that the output conforms to the HTML specification. The proposed tool is to act as a testing tool to find faults from all LOC (line-of-code) in web applications. Comparatively more advanced tools find faults during the rendering process, so it uses line coverage so less as compared to the proposed tool.
Keywords: Faults, Bug Finder, basic validator, Web Applications, failures

INTRODUCTION

Powerful test creation tools, such as DART [1], Cute [2], and EXE [3], make tests by executing an application on input values and then building additional input values by solving symbolic conditions obtained from exercised control paths. Such approaches have not been studied in the domain of web applications, which pose special challenges due to the dynamism of the coding languages, the advantage of implicit parameters, their use of persistent state, and their designs of user interaction. The offered work continues dynamic test generation to the area of web applications that dynamically create web pages during execution, which are typically given to the user in a browser. Web development languages including server-side web programming languages such as PHP, ASP.net, and C# are used as input in the proposed work [4].

Our goal is to find two classes of failures in web applications: execution crashes that show as errors or warnings during the process execution, and HTML failures that happen when the application allows twisted HTML. Execution flops may occur, for example, when a web form calls an irregular function or reads a missing file. Some HTML code includes an error warning, and execution of the application may be suspended, depending on the hardness of the fault. If the input contains not syntactically well-formed HTML, then the proposed tool generates output containing HTML faults. HTML faults are generally not as notable as execution faults because Web browsers are intended to provide some degree of malformedness in HTML, but they are unacceptable for several reasons. The first and most critical is that browsers' attempts to compensate for malformed web pages may lead to crashes and security vulnerabilities. Second, standard HTML performs faster. Third, deformed HTML is less portable across browsers and is vulnerable to breaking or watching apart when displayed by untested browser versions. Fourth, a browser force succeeds in presenting only part of a malformed webpage, while silently discarding important information. Fifth, search engines may have trouble indexing deformed pages [13].

Web developers widely know the importance of creating error-free HTML. HTML validators are used to control many websites. However, HTML validator can only point out problems in HTML pages and are by them incompetent of finding faults in applications that generate HTML pages. Checking dynamic Web applications 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 include multiple faults. So, this tool has implemented using symbolic execution to find faults in web applications. This scheme continues dynamic test creation to the area of web applications that dynamically create the HTML pages throughout execution, which offer to the user in a browser.

LITERATURE REVIEW

The main strength of DART is that testing is often performed completely automatically on any program that compiles – there's no got to write any driver or harness code. During testing, DART detects standard errors like program crashes, assertion violations, and non-termination. Preliminary experiments to unit test several samples of C programs are very encouraging. [1]

In unit testing, a program was divided into units of collection of methods. A piece of the unit can try by creating contributions for solitary section work. The entry method may contain pointer contentions, in which case the contributions to the unit are memory

graphs. The paper talks about the issue of mechanizing unit testing with memory charts as sources of info. The methodology utilized expands on before work consolidating representative and reliable execution, and all the more explicitly, utilizing such a request to produce test contributions to investigate all plausible execution ways. CUTE [2] is a variation on the DART approach.[2]

EXE is a valuable bug-discovering system that automatically creates inputs that crash genuine code. EXE utilizes secure, piece level precise representative execution to discover significant mistakes in code and naturally create inputs that can hit these faults [3]

Fuzz testing is a valuable method for discovering security vulnerabilities in programming. Customarily, fluff testing devices apply irregular transformations to very much framed contributions of a program and test the different qualities. Paper introduces an option white box fuzz testing approach motivated by ongoing advances in symbolic execution and dynamic test generation. Their methodology records a genuine keep running of the program under test on very much framed information, emblematically assesses the recorded follow, and assembles limitations on sources of info catching how the program utilizes these. The gathered requirements are then invalidated one by one and unraveled with a limitation solver, creating new information sources that activity diverse control ways in the program [9].

PROPOSED WORK

The system architecture is the conceptual model that defines the structure and behavior of the system. An architecture description is a precise representation of a system. System architecture represents the structure of system components, external visibility of those components, and their relationships between them. The language used for architecture description is called the architectural description language.

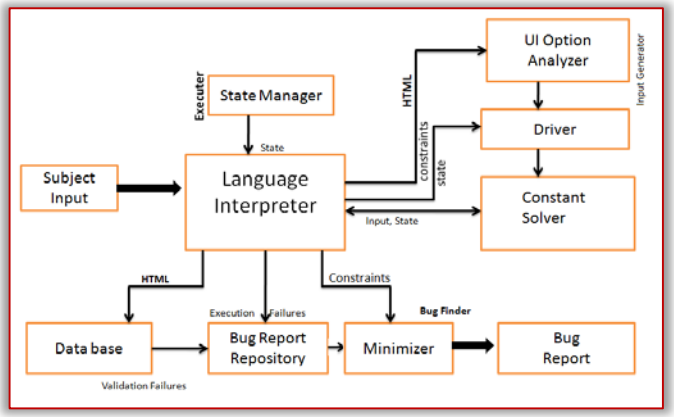

Figure 1: The Proposed system for finding faults in websites

The proposed tool is to implement our technique for Web Technology. This module consists of the following major components, input generator, executer, and bug finder illustrated in Figure 1.

— The executor is responsible for executing a web programming script with a given input in a given state. The executor contains two subcomponents:

(a) The Language Interpreter is a language interpreter that we modify to propagate and record path constraints and positional information associated with the output. This positional knowledge use to determine which failures are likely to be symptoms of the same fault.

(b) The State Manager restores the given case of the environment (database, session, and cookies) before the execution and stores the new environment next to the execution.

— The Input Generator implements the algorithm described below. The Input Generator contains the following subcomponents:

(a) The User Interface Analyzer examines the HTML code output of each execution to change the interactive user choices into new inputs to execute.

(b) The Driver generates new path checks from the checks found during the execution.

(c) The Constraint Solver computes an task of values to input parameters that satisfy a given path checks.

—The Bug Finder practices an oracle to find HTML failures, stores all bug reports, and finds the minimal requirements on the input parameters for each bug report. The Bug Finder has the following subcomponents:

(a) The database stores the HTML faults in the output of the program.

(b) The Bug Report database stores HTML and execution faults report found during all executions.

(c) The Input Minimizer determines, for an assigned bug report, the shortest path constraint on the input parameters that results in inputs producing the same failure as in the report.

EXPERIMENTAL SETUP

This module program has implemented using C#.net. The experiment is carried out by different input datasets and analyzing output with the dataset in the Windows 7 operating system. We have taken four datasets implemented using PHP server-side scripting language.

Table 1 shows a time practiced for the detection of faults by the existing tools and time needed for the detection of faults for the proposed tool. The proposed tool takes less time for the detection of execution and HTML faults present in the web applications as compared to the existing tools.

Table 1: Time required to process input

Input	Time (in seconds) required to process Proposed System	Existing System
timeclock	210	425
Phpbb2	195	420
Phpsysinfo[17]	300	720
Synthetic Website Dataset 1	256	505*
Synthetic Website Dataset 2	350	680*

"*" Indicates expected values for time requirement to find faults generated by existing systems.

As a shown in table 1 above, execution time reduced to 50% averagely due to system executes line of code sequentially. In case of existing systems, generate input dynamically hence time required to execute LOC is high. Also LOC for existing system is below the 50% while proposed system used more than 90% except scripting (Java Script, VB Script)

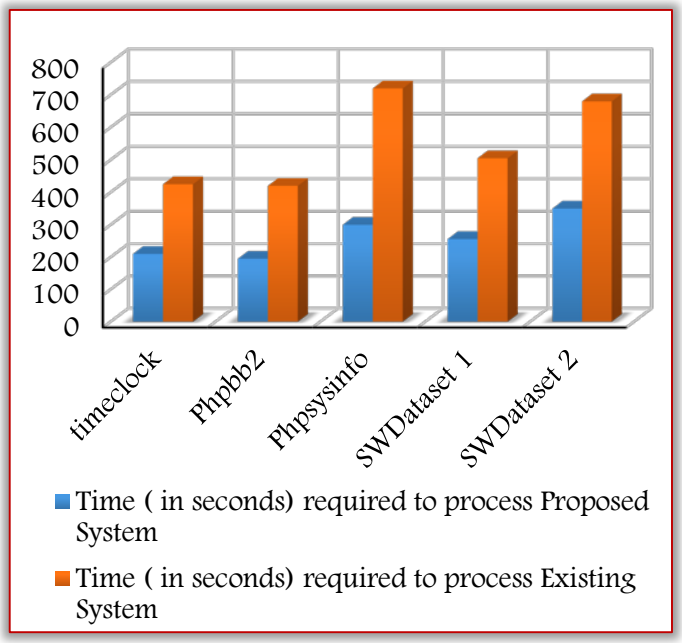

Figure 2: Time Required Comparison

Figure 2 shows some faults detected by the proposed tool and the existing tool.

Table 2: Error detection

Input	No of errors detected Proposed System	Existing System
timeclock	395	387
Phpbb2	35	30
Phpsysinfo[17]	17	9
Synthetic Website Dataset 1	158	151*
Synthetic Website Dataset 2	60	54*

"*" Indicates expected values for errors generated by existing systems.

Table 2 shows better performance by the proposed system to find faults in respective datasets.

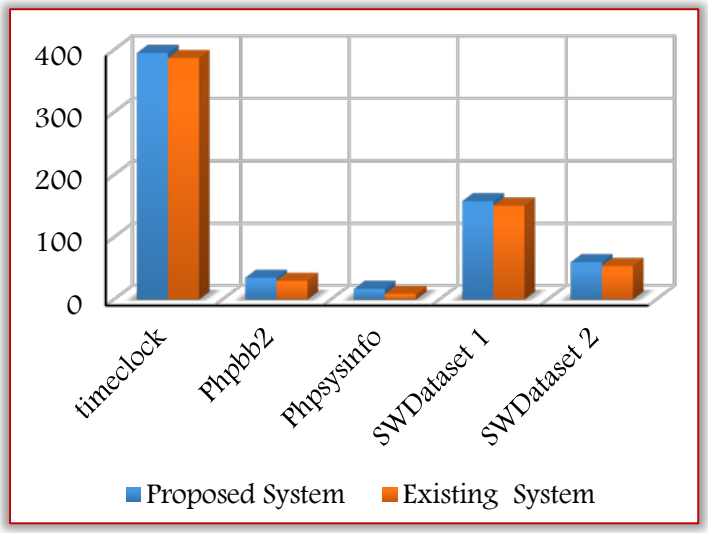

Figure 3: Error Detection Comparison

Table 3: LOC coverage

Input	LOC coverage (%) Proposed System	Existing System
timeclock	94	26.8
Phpbb2	92	31.7
Phpsysinfo[17]	91	56.2
Synthetic Website Dataset 1	96	52*
Synthetic Website Dataset 2	90	54*

"*" Indicates expected values for LOC (line of code) coverage generated by existing systems.

Table 3 shows excellent LOC coverage by the proposed system with compare to existing system.

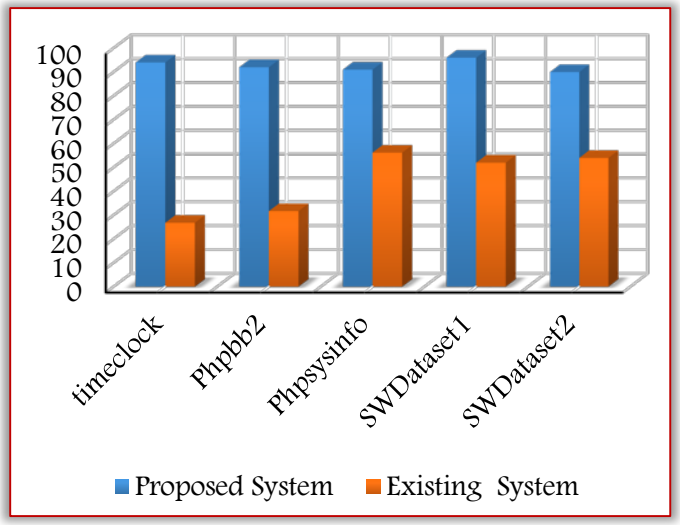

Figure 4: LOC coverage

CONCLUSION

We have introduced a technique for finding faults in web applications implemented in PHP and C#.net technology. This tool is to give a new approach to detecting HTML and execution faults. This tool has utilized the dynamic inputs to check the fault events and perform an automated examination to decrease

the extent of the fault prompting inputs. The previous tools, for example, Apollo, randomized systems, identify the faults from static applications of PHP. However, our tool detects static, dynamic bugs as well as HTML errors present in web applications implemented in PHP as well as C#.net.

The proposed tool detects HTML errors and execution errors in less time as compared to the existing tools. The accuracy of error detection of this tool is better than that of the existing tool. Hence, the precision of the web applications has improved by a proposed system.

The proposed tool is to act as a testing tool to find faults from all LOC (line-of-code) in web applications. More advanced tools find faults during rendering process, so line coverage is less but this tool is act as testing tool, so line coverage is more compare to other tools.

References

[1] Godefroid, Klarlund N, and Sen K, "DART: Directed Automated Random Testing," Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, pp. 213-223, 2005.

[2] Sen K, Marinov D, and Agha G, "CUTE: A Concolic Unit Testing Engine for C," Proc. ACM SIGSOFT Int'l Symp. Foundations of Software Eng., pp. 263-272, 2005.

[3] Cadar C, Ganesh V, Pawlowski P M, Dill D L, and Engler D.R., "EXE: Automatically Generating Input of Death," Proc. Conf. Computer and Comm. Security, pp. 322-335, 2006.

[4] Artzi S, Kie zun A, Dolby J, Tip F, Dig D, Paradkar, "Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking" IEEE Trans. on Software Engg. Vol. 36, NO. 4, July/Aug 2010

[5] Shinde S K, Joshi S D.,Kaushik D K., "Fault-Tolerant System for Dynamic Web Applications", IJEDR, volume 3 issue 1, (2015), pp. 489-494.

[6] Holzmann G J, "The Model Checker SPIN," Software Eng., vol. 23, no. 5, pp. 279-295, 1997.

[7] Benedikt M, Freire J, and Godefroid P, "VeriWeb: Automatically Testing Dynamic Web Sites," Proc. Intel Conf. World Wide Web, 2002.

[8] Clause J and Orso A, "Penumbra: Automatically Identifying Failure-Relevant Inputs Using Dynamic Tainting," Proc. Intel Symp. Software Testing and Analysis, 2009.

[9] Godefroid P, Levin M Y, and Molnar D, "Automated Whitebox Fuzz Testing," Proc. Network Distributed Security Symp., pp. 151-166, 2008

[10] Cleve H and Zeller A, "Locating Causes of Program Failures," Proc. Intel Conf. Software Eng., pp. 342-351, 2005.

[11] Misherghi G and Su Z, "HDD: Hierarchical Delta Debugging," Proc. Intel Conf. Software Eng., pp. 142-151, 2006.

[12] Csallner C, Tillmann N, and Smaragdakis Y, "DySy: Dynamic Symbolic Execution for Invariant Inference," Proc. Intel Conf. Software Eng., pp. 281-290, 2008.

[13] Zoufaly F, "Web Standards and Search Engine Optimization (SEO)—Does Google Care About the Quality of Your Markup?" 2008.

[14] Shinde S K, Joshi S D, "Iterative Code Reviews System For Detecting And Correcting Faults From Software Code Documents" IJARET Volume 5, Issue 11, November (2014), pp. 61-67

[15] Minamide Y, "Static Approximation of Dynamically Generated Web Pages," Proc. Intel Conf. World Wide Web, 2005.

[16] Shinde S K, Joshi S D, "Schema Inference & Data Extraction from Templatized Web Pages", IEEE, International conference on Pervasive Computing (ICPC) 2015, Sinhgad College of Engineering, Pune, 10.1109/PERVASIVE.2015.7087084, Jan.2015 pp. 1-6

[17] Shinde S K, Joshi S D, "Web Based Requirement Elicitation Tool" (IJARCET) Volume 4 Issue 9, September 2015 pp. 3719-3722

ACTA TECHNICA CORVINIENSIS – Bulletin of Engineering
ISSN: 2067-3809