A Browser Compatibility Testing Method Based on Combinatorial Testing*

Lei Xu[1,3], Baowen Xu[1,2,3], Changhai Nie[1,3], Huowang Chen[2,3], and Hongji Yang[4]

[1] Department of Computer Sci. & Eng., Southeast University, Nanjing 210096, China
[2] Computer School, National University of Defense Technology, Changsha 410073, China
[3] Jiangsu Institute of Software Quality, Nanjing 210096, China
[4] School of Computing, De Montfort University, England
bwxu@seu.edu.cn

Abstract. For ensuring the displaying effects of Web applications, it is important to perform compatibility testing for browsers under the different configurations and it is a hard task to test all. So this paper focused on the improvements for browser compatibility testing. Firstly, we introduced the related work. Then we analysed the functional speciality of the current popular browsers, and provided the preconditions to simplify problems. Next we gave an instance and brought forward the single factor covering method and the pair-wise covering design to gain testing suits. Finally, we introduced the assistant tools we have developed and the future work.

1 Introduction

In recently years, Web applications are attracting more and more users with their important characters of universality, interchangeability and usability [11]. Browser compatibility testing is needed to guarantee the displaying functions of Applets, ActiveX Controls, JavaScript, CSS, HTML, etc in all kinds of system configurations. This job is quite fussy, so we demand some test cases to cover the huge combinations.

At present, people start researching web testing and propose some elementary methods [3,4,6,7,8]. This paper focuses on how to improve browser compatibility testing. Section 2 presents the related work. Section 3 provides two methods to obtain the suits of test cases. Section 4 is concerned with the tools and the conclusions.

2 Related Work

At present, the browser is constitutive of HTML Render, Java VM, JavaScript, Plug-in Handler, etc. Browsers take charge of parsing and showing the Web page elements. By now, there are many types of browsers, such as NN, IE, AOL, etc. Each type of browser

* This work was supported in part by the National Natural Science Foundation of China (NSFC) (60073012), National Grand Fundamental Research 973 Program of China (2002CB312000), National Research Foundation for the Doctoral Program of Higher Education of China, Opening Foundation of Jiangsu Key Laboratory of Computer Information Processing Technology in Soochow University, and SEU–NARI Foundation.

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 310–313, 2003.

has different versions, and they can be applied on the platforms of Windows, Macintosh, UNIX, Linux, etc. Different browsers may have different functions, parsing results and fault-tolerances; different type of computers may have different font types and sizes. So it is urgent to test the displaying of applications in different browser environments.

Presently, the combinatorial testing is used in many software-testing fields, and the research focuses on the generation of test cases. David M. Cohen put forward a heuristic generating method based on pair-wise coverage [1,5]. Y. Lei proposed a gradually expanding generation method for pair-wise covering test cases based on parameter sequence [10]. Noritaka Kobayashi brought forward a kind of algebra method to generate cases, which was better than heuristic method [9]. Our study emphasizes on how to use these methods to the Web compatibility testing.

3 The Methods for Browser Compatibility Testing

3.1 Preliminary Methods

Firstly, it is needed to make clear of the related configuration requirements. Next, we should confirm that all the relative equipments are useable. Further more, we should find out the popular equipments and compress the kinds into a controlled range. Then, we can generate test cases based on different algorithms and execute the compatibility testing. Suppose we need to test *n* kinds of equipments after the pre-process, named $d_1, d_2, ..., d_n$, and d_1 has a_1 values, …, d_n has a_n values. It needs $m = \max_{1 \le i \le n} a_i$ test cases to cover all the values of every parameter at least, $m = \max_{1 \le i \ne j \le n} a_i * a_j$ test cases to cover all the pair-wise combinations of every parameter at least… and so on. With the increase of parameters, the number of test cases increases quickly.

The relationships between the Web page elements and the relative parsing equipments are direct, thus they have little interacting influences among themselves. So pair-wise coverage is enough, and sometimes the single factor coverage is suitable.

3.2 Browser Compatibility Testing Based on Single Factor Coverage

Now we give an example of compatibility testing under the popular configurations. Supposing the software under testing (SUT) is a network game, we only consider five equipments for simplify, and d_1=video, d_2=audio, d_3=printer, d_4=browser, and d_5=operating system, $a_1 = a_2 = a_3 = a_4 = a_5 = 2$, shown in Table 1.

Table 1. The Equipments Types for Web Compatibility Testing

Video	Audio	Printer	Browser	Operating System
A1	B1	C1	D1	E1
A2	B2	C2	D2	E2

If we consider all the possible combinations, we need 32 test cases (2^5= 32). The single factor covering method changes the value of one factor once a time and keeps the typical values of other factors. In our example, we designate the typical values of all the equipments, and when one changes its value, others keep their typical values. Through this method, we obtain the testing suits as follows:

(1) A1, B1, C1, D1, E1
(2) A2, B1, C1, D1, E1
(3) A1, B2, C1, D1, E1
(4) A1, B1, C2, D1, E1
(5) A1, B1, C1, D2, E1
(6) A1, B1, C1, D1, E2

As a result, we gain the six test cases to replace the thirty-two ones. In the view of combinatorics, there are five parameters in the SUT and each parameter has two values, then the number of arbitrary two parameters is 40 ($C_5^2 \times 2^2 = 40$), while this method only cover 10 ($C_5^1 \times 2^1 = 10$). This method is easy to accomplish and suits for the situation that every factor does not interfere with others, but when the SUT has many factors and each factor has many values, this method is not practical.

3.3 Browser Compatibility Testing Based on Pair-Wise Coverage

In order to make sure the possible influences between the arbitrary two factors, we should have the testing based on pair-wise coverage.

Definition 1 Suppose $A=(a_{i,j})_{n\times m}$. The j-th column represents the j-th parameter, and the elements of the j-th column are from the finite symbol set T_j (j=1, 2,…, m), i.e. $a_{ij} \in T_j, i = 1,2,...,n$. If the arbitrary two columns of A satisfy these conditions: the total symbol combination of T_i and T_j are occurred in the pair-wise which consists of the i-th and j-th columns, we call A the *pair-wise covering table*.

The row number, n, of matrix A is the number of test cases. If the number is the least positive integer to ensure the conditions, we call A the least pair-wise covering table, and each row of A is a test case [2,12]. Now we give the testing suits based on *pair-wise covering table*:

(1) A1, B1, C1, D2, E1
(2) A1, B1, C2, D1, E2
(3) A1, B2, C1, D1, E1
(4) A2, B1, C2, D1, E2
(5) A2, B2, C1, D2, E2
(6) A2, B2, C2, D2, E1

The above six test cases cover all the pair-wise covering situations, so it has better quality than the former six cases which are based on single factor coverage.

3.4 Testing Steps

After generating the test cases, we can perform the browser compatibility testing. Firstly, select the related equipments and prepare for the testing. Next, build a matrix for testing information, in which, each row represents one kind of configuration, each column for one displaying page element. Then work in the respective configure environment. We should check the showing functions and impressions of every Web page element in detail, compared with the specifications. Finally, after obtaining the full matrix, we can analyse the testing status and draw some conclusions such as wanting parsers, hardware conflicts, element's functional errors, etc.

4 Conclusion Remarks

Presently, there are multiple kinds of assistant tools for Web browser compatibility testing in practice. They take effective actions in reducing the testing tasks and improving efficiency. But due to the limitation of the algorithms, these tools only fit for some special cases and do not have general properties.

We have done much research in the test cases generation and selection. Our algorithms concluded the idea of single factor coverage, pair-wise coverage, orthogonal array experimental design, uniformity design, etc, and we made progress in the construction of coverage tables. At present, we have developed the related testing assistant tools based on the above algorithms. As a consequence of these research, we can obtain the most appropriate algorithms and test cases in different situations when we testing a system with multi-factor and multi-level.

The work of browser compatibility testing is very burdensome. In most cases, it is impossible to cover all the possible combinations during software testing. In this study, we analysed the characters of Web applications and the function speciality of current popular browsers, and simplified the problems greatly. Then we provided two methods, i.e. single factor coverage and pair-wise coverage to obtain the suits of compatibility test cases. At last, we presented the assistant tools we developed for the test cases generation. Future work includes algorithm optimisation and generalisation.

References

1 Williams, A.W. and Probert, R. L.: A Practical Strategy for Testing Pair-wise Coverage of Network Interfaces. Proc. of 7th Int. Symp. Software Reliability Engineering (1997): 246–254.
2 Nie, C. and Xu., B.: An Algorithm for Automatically Generating Black-box Test Cases Based Interface Parameters. Chinese Journal of Computer (in Chinese), to appear in 2003.
3 Kallepalli, C. and Tian., J.: Measuring and Modeling Usage and Reliability for Statistical Web Testing. IEEE Trans. Software Engineering, 2001, 27(11): 1023–1036.
4 Kung, D.C., Liu, C.H. and Hsia, P.: An Object-Oriented Web Test Model for Testing Web Applications. Proc. of 1st Asia-Pacific Conf. on Quality Software (APAQS) (2000): 111–121.
5 Cohen, D. M., Fredman, M. L.: New Techniques for Designing Qualitatively Independent Systems. Journal of Combinational Designs, 1998, 6(6): 411–416.
6 Ricca, F. and Tonella, P.: Web Site Analysis: Structure and Evolution. Proc. of Int. Conf. on Software Maintenance (ICSM) (2000): 76–86.
7 Xu, L., Xu, B. and Chen, Z.: Survey of Web Testing. Computer Science (in Chinese), 2003, 30(3): 100–104.
8 Xu, L., Xu, B. and Chen, Z.: A Scheme of Web Testing Approach. Journal of Nanjing University (in Chinese), 2002, 38(11): 182–186.
9 Kobayashi, N., Tsuchiya, T. and Kikuno, T.: A New Method for Constructing Pair-wise Covering Designs for Software Testing. Information Processing Letters, 2002, 81(2): 85–91.
10 Mandl, R.: Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing. Communications of the ACM, 1985, 28(10): 1054–1058.
11 Zhang, W., Xu, B., Zhou, X., Li, D. and Xu, L.: Meta Search Engine Survey. Computer Science (in Chinese), 2001, 28(8): 36–41.
12 Xu, B., Nie, C., Shi, Q. and Lu, H.: An Algorithm for Automatically Generating Black-Box Test Cases, Journal of Electronics, 2003, 20(1): 74–77.