Orthogonal Array and Virtualization as a Method for End-to-End Testing of IT Architecture and Applications

Ljubomir Lazić, *Member, WSEAS*

Abstract — **We propose in this paper a general framework for an integrated End-to-End Testing of IT Architecture and Applications using the simultaneous application of combinatorial testing and virtualization. Combinatorial testing methods are often applied in cases of the configuration testing. Virtualization, in the process of testing, is based on setting the necessary environment to multiple virtual machines, which run on one or in smaller groups of physical computers, which are: reduce the cost of equipment and related resources, reduce the time required to manage the testing process, and favours raising removal of test infrastructure. Together, combinatorial testing and virtualization presents practical approach to improving process of testing, through the balancing quality, cost and time.**

Key-Words — **Combinatorial testing, Environment Virtualization, Software Testing, Virtual Mashines.**

I. Introduction

TESTING is a crucial step in the development of a software-intensive system, as it checks the compliance of a system to the end user requirements [1]. The development of software testing systems must be performed in effective and efficient manner. It is easy to see that an effective testing is a very good indicator of the quality product and efficient testing procedure to ensure the faster development cycle that is an imperative requirement for large organization. The prime objective of the System Testing is to cover all forms of the testing techniques related to systems to ensure the successful development and application of software and technology. Software testing involves the process of detecting software discrepancies so that they can be corrected before they are installed into a live environment supporting operational business units. To better support this complex task of software-testing, this study proposes identifying and applying a general framework for an integrated End-to-End Testing of IT Architecture and Applications using the simultaneous application of combinatorial testing and virtualization.approach to software testing. System testing is an integral, costly, and timeconsuming activity in the software development life cycle. In addition, because testing involves running the system being tested under a variety of configurations and circumstances, automation of execution-related activities offers another potential source of savings in the testing process.

In this paper, we consider a problem that arises in black box testing: generating small test suites (i.e., sets of test cases) where the combinations that have to be covered are specified by input-output parameter relationships of a software system. That is, we only consider combinations of input parameters that affect an output parameter, and we do not assume that the input parameters have the same number of values. To solve this problem, we propose interaction testing, particularly an Orthogonal Array Testing Strategy (OATS) as a systematic, statistical way of testing pair-wise interactions [1]. In software testing process (STP), it provides a natural mechanism for testing systems to be deployed on a variety of hardware and software configurations. The combinatorial approach to software testing uses models to generate a minimal number of test inputs so that selected combinations of input values are covered.

The paper shows that the combinatorial testing and virtualization together can dramatically improve the process of testing. The Web application example points the way how to use virtualization to cover a wide range of test environments and how to obtain the configuration testing to be more effective.

II. Framework for an Integrated End-to-End Testing of IT Architecture and Applications

Today’s companies and organizations are increasingly dependent on the success of the distributed online applications that they deploy. These applications provide a multitude of functionality, ranging from delivering products and services directly to customers to facilitating internal communication. Given the importance of these applications, they usually undergo rigorous testing before their deployment.

However, they are only one component of the big picture. If the underlying infrastructure (e.g. the application server) is unavailable, users will not be able to access the desired services provided by these applications no matter how robust the applications are. What infrastructure support do enterprise applications need? In our view, they need support from at least four categories of

This work was supported in part by the Ministry of Science and Technological Development of the Republic of Serbia under Project No. TR-13018..

Lj. R. Lazić, Državni Univerzitet, Novi Pazar, (telefon: 381-64-6666706; e-mail: llazic@np.ac.yu).

infrastructure components: hardware equipment, operating systems, middleware, and network connectivity.

Typical hardware equipment on enterprise networks includes servers, workstations, load balancers, switches, routers, and firewalls. Operating systems run on some of the hardware equipment, e.g. servers and routers. Middleware includes the non-OS software between the applications and hardware, such as application containers and messaging service.

Network connectivity among the hardware equipment supports the communication between end users and applications. Figure 1 shows a typical enterprise infrastructure.

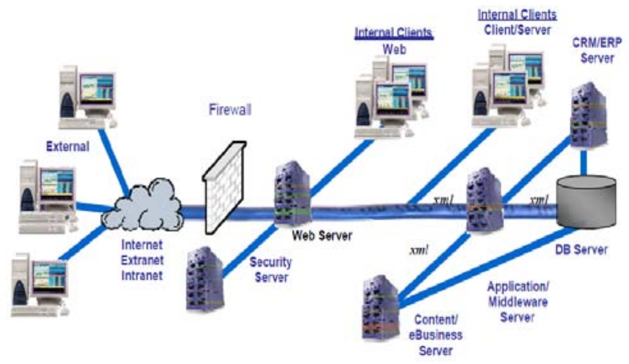

Fig. 1. A Typical Multi-Tier IT Architecture Today

Given the complexity of large-scale enterprise infrastructures and the interdependencies between components, infrastructure failures could occur during the deployment and operation of an application. In fact, many failures in the delivery of online services stem from the issues of the underlying infrastructure such as server failures and configuration errors. In practice, test beds usually have a much smaller scale and complexity than the deployed infrastructure due to the cost of setting up and managing the tested. To address the deficiencies of existing infrastructure testing tools, our project aims to construct a prototype testing environment and develop the associated tools to evaluate the reliability and performance of large-scale enterprise infrastructures [2-4].

Our framework, Integrated and Optimized Software Testing Process - IOSTP [1], has two main components: (1) a methodology to build a virtual test bed that can accurately emulate any infrastructure topology and simulate failures, attacks and other types of stresses on the infrastructure to identify defects and bottlenecks; and (2) a optimization model and a tool to automatically generate different test scenarios on the model. We are taking the following steps to build IOSTP.

System testing is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

System testing is actually done to the entire system against the Functional Requirement Specification(s) (FRS) and/or the System Requirement Specification (SRS). Moreover, the system testing is an investigatory testing phase, where the focus is to have almost a destructive attitude and test not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

The following examples are different types of testing that should be considered during System Testing:

- Functional testing
- Usability testing
- Performance testing (Load, Volume, Stress)
- Compatibility testing
- Security testing
- Smoke testing
- Exploratory/Adhoc testing
- Regression testing
- Reliability testing
- Recovery testing
- Installation testing
- Accessibility

The challenge for developers, QA teams, and management alike is how to speed up their testing processes and increase accuracy and completeness - without breaking their already tight budgets. In the age of accelerating product lifecycles and pressures on reducing the cost, the impact of traditional approach to testing has had a serious impact on IT organizations. The business climate of today is such that IT organizations are asked to do more with fewer resources and without any significant reduction in the quality of the product that is being delivered. When IT organizations make attempts to cut on the cost, Software Testing is often the first item that would be cut.

By implementing automated testing, companies can dramatically increase both the speed and accuracy of their testing processes, providing a higher return on investment (ROI) from software projects while dramatically cutting risk.

IOSTP follows FURPSSI (Functionality, Usability, Reliability, Performance, Security, Scalability and Installation & Compatibility) model for System Testing. There is no question that rigorous functional testing is critical to successful application development. By automating key elements of functional testing, companies can meet aggressive release schedules, test more thoroughly and reliably, verify that business processes function correctly, and ultimately generate higher revenue and customer satisfaction from their online operations. A strategic approach in developing a test automation framework using tools and methodologies will improve the test coverage in regression cycles and reduce the test effort in subsequent release cycles as depicted in Fig. 2.

III. Combinatorial testing - Orthogonal Array Testing Strategy (OATS) and Techniques

Testing a software system requires the creation of test cases, which contain values for input parameters and the expected results. Exhaustive testing for all of the possible combinations of parameters, in most cases it is not possible, it is not feasible, or the cost is out of the available budget. The main goal of using different methods and techniques of testing is to create a smaller number of combinations of parameters and their values, which will be tested.

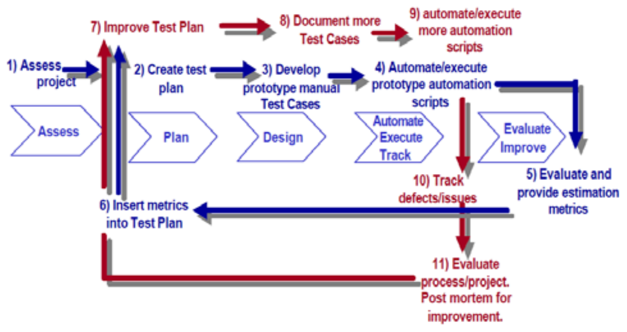
7) Improve Test Plan
8) Document more Test Cases
9) automate/execute more automation scripts
1) Assess project
2) Create test plan
3) Develop prototype manual Test Cases
4) Automate/execute prototype automation scripts
Assess
Plan
Design
Automate Execute Track
Evaluate Improve
5) Evaluate and provide estimation metrics
10) Track defects/issues
6) Insert metrics into Test Plan
11) Evaluate process/project. Post mortem for improvement.

Fig. 2. The IOSTP's Test Automation Process (TAP) for End-To-End Architecture Testing

The OATS provides representative (uniformly distributed) coverage of all variable pair combinations. This makes the technique particularly useful for integration testing of software components. It is also quite useful for testing combinations of configurable options (such as a web page that lets the user choose the font style, background colour, and page layout). As an example of the benefit of using the OATS technique over a test set that exhaustively tests every combination of all variables, consider a system that has four options, each of which can have three values. The exhaustive test set would require 81 test cases (3 x 3 x 3 x 3 or the Cartesian product of the options). The test set created by OATS has only nine test cases, yet tests all of the pair-wise combinations. The OATS test set is only 11% as large at the exhaustive set and will uncover most of the interaction bugs. It covers 100% (9 of 9) of the pair-wise combinations, 33% (9 of 27) of the three-way combinations, and 11% (9 of 81) of the four-way combinations. What degree of interaction occurs in real system failures? Within the NASA database application, for example, 67 percent of the failures were triggered by only a single parameter value, 93 percent by two-way combinations, and 98 percent by three-way combinations. The detection-rate curves for the other applications studied are similar, reaching 100 percent detection with four- to six-way interactions. An orthogonal array is a balanced two-way classification scheme used to construct balanced experiments when it is not practical to test all possible combinations.

Definition: Orthogonal array $O(\rho, k, n, d)$

- An orthogonal array is denoted by $O(\rho, k, n, d)$, where:
- ρ is the number of rows in the array. The k-tuple forming each row represents a single test configuration, and thus ρ represents the number of test configurations.
- k is the number of columns, representing the number of parameters.
- The entries in the array are the values $0, \dots, n-1$, where $n = f(n_0, \dots, n_{k-1})$. Typically, this means that each parameter would have (up to) n values.
- d is the strength of the array.

The OATS provides representative (uniformly distributed) coverage of all variable pair combinations. This makes the technique particularly useful for:

- integration testing of software components,
- testing combinations of configurable options (such as a web page that lets the user choose the font style, background colour, and page layout).

Example: For n variables with v values, k-way combinations, Number of combinations for all combibnations is:

$$\rho_{Comb} = \binom{n}{k} \cdot v^k \qquad (1)$$

The OATS method provides much lower number of combinations for $k=2$ way interaction, ie. pair-wise interaction of maximum No. of tests as:

$$\rho_{OATS} = n^2 + v_{\max} \log^2 v_{\max} \qquad (2)$$

In a specific example of a 12 variables: 7 Boolean, two 3-value, one 4-value, two 10-value in a typical test configuration for k-way interaction requires corresponding number of test combinations as shown in next Table:

k	**# test cases**
2-way	100
3-way	405
4-way	1,375
5-way	4,220
6-way	10,902

IV. Virtualization Properties and Advantages

Virtualization allows that more of the software environments, which in this case are called virtual machines (VMs), could be physically executed at the same time, at only one physical computer (host), sharing the same hardware resources among them. Communication between host and virtual machine is provided by the software, generally called: the monitor, or hypervisor, which can be run directly on the physical computer, or may be a layer between the host operating system and virtual machines. There are several virtualization approaches. It is considered that the native virtualization and the paravirtualization are the best for software testing [2] [3]. The most important advantages of using virtual machines are: reduction of costs; isolation of applications, easier testing, standardization of testing and portability. In addition, tested software accepts them as separate machines. Also, in the case of the crash of some of the virtual machine, due to applicable error or OS error, other virtual machines will continue to run, keeping the functionality of other parts of the system, as shown in Fig. 3. While VMs benefits all sound ideal, virtual machines do have two main drawbacks: they share physical resources with the host and any other running virtual machines, and they carry some processing overhead. So it could not be expected the same performance from a virtual machine as do from a physical one.

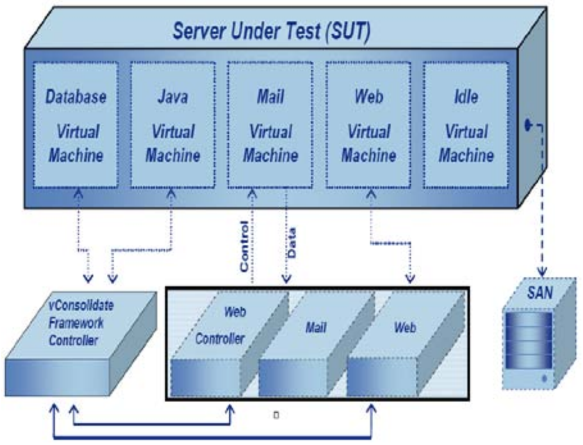

Fig. 3. Typical VMs test configuration

V. APPLICATION - CASE STUDY

An example was the Web applications testing on different platforms, in order to check whether the request is fulfilled: "Web application can be used by different clients (Web browser), different customized and ran on different operating systems" [4].

In this case, only the client side and a combination of OS, web browser, web browser settings and defined localization in the role of the platform, was considered. Execute applications on the Web server is done in controlled conditions, which can be precisely defined in advance.

In the testing process of the Web application configuration, parameters for the test cases creation were: OS, selected localization, Web browser, support for JavaScript, ActiveX controls, cookies. But, at glance, it is clear that there are a number of test configurations on the client side, which are necessary to provide to carry out exhaustive testing.

Using combinatorial testing, we have defined all pairs of combinations, and later, the reduced set of test cases for testing the configuration. Testing was realised by means of using virtualization.

The process included the following steps:

1. Parameters that will be tested are identified:

a. *OS, localization, web browser, support for JavaScript, cookies, ActiveX controls,*

2. Certain values are possible - selected for each parameter:

a. *Client OS*: **Windows XP, Windows Vista, Mac OS X 10, Windows 2000, Linux**

b. *Browser:* **Internet Explorer 6, Internet Explorer 7, Mozilla Firefox 2, Mozilla Firefox 3, Apple Safari 3, Opera 9**

c. *Localizations:* **Albanian, Croatian, Hungarian, Serbian, Slovenian, another localization**

d. *JavaScript:* **allowed, not allowed**

e. *Cookies:* **allowed, not allowed**

f. *ActiveX control:* **allowed, not allowed**

3. Limits are defined:

a. In order to avoid the risk of loss of valid pairs, there are not allowed to create certain combinations (Browser Apple Safari 3 can be tested only on Mac OS X 10. At the same time, this OS will not be tested in combination with other browsers).

b. There are defined values (seeds), which must appear as a test case, within the generated set of test cases, because they are expected as most likely combination of values of parameters.

c. Weight factors, i.e. specific values for parameters are estimated (more emphasis is given to the following parameters values: client OS - Windows XP, browser - Internet Explorer 7, JavaScript - allowed, Cookies - allowed, the ActiveX control - allowed).

4. The total number of test cases, regarding the defined variables and their values for all combinations, were 1440.

5. Virtual machines are created under defined configurations for testing.

Thus, in case study, the number of initial configuration, we had to test, was 1440. Applying the all-pairs algorithm, to extract the unique combination of pairs, the initial number of the necessary configurations, starting from 1440, was reduced to 38. Thus, the 2.6% of the total number of the theoretically possible configurations covered all the pairs of variables. Testing the final set of selected test configurations was done by applying virtual machine.

VI. CONCLUSION

The aim of this paper is to point out the possibility of improving the process of testing software systems. Initial idea was that the use of the software virtualization in the process of testing will reduce the requirements for the necessary hardware and software resources. At the same time, virtualization is combined with the combinatorial testing, in order to reduce the number of test cases that need to test, while this does not impair the accuracy and reliability testing software. On the basis of acquired experience and obtained test results, it can be noted that the virtualization of the application and the combinatorial testing were good decision. This is especially true in the case of configuration testing, where was necessary to contribute to the reduction of the test resources, such as were: time, required hardware and software configurations.

LITERATURA

[1] Lj. Lazic, N. Mastorakis, "Orthogonal Array application for optimal combination of software defect detection techniques choices", WSEAS TRANSACTIONS on COMPUTERS, August 2008, pp. 1319-1336.

[2] S. Seetharaman, K. Murthy, "Test Optimization Using Software Virtualization", IEEE Software, IEEE Computer Society, September/October 2006, pp. 66 - 69.

[3] G. Goth, "Virtualization: Old Technology Offers Huge New Potential", IEEE Distributed Systems Online, vol. 8, no. 2, 2007.

[4] S. Popovic and Lj. Lazic."Orthogonal Array And Virtualization As A Method For Improvement Configuration Testing", Proceedings of 1st IEEE Eastern European Regional Conference on the Engineering of Computer Based Systems - ECBS-EERC 2009, Novi Sad, September 7th-8th 2009, pp.148-149.