A Simultaneous Application of Combinatorial Testing and Virtualization as a Method for Software Testing

LJUBOMIR LAZIĆ
Department for Mathematics and Informatics
State University of Novi Pazar
SERBIA
llazic@np.ac.yu, http://www.np.ac.yu

SNEŽANA POPOVIĆ
School of Computing
Union University of Beograd
SERBIA
spopovic@raf.edu.rs, http://www.raf.edu.yu

NIKOS E MASTORAKIS
Technical University of Sofia,
English Language Faculty of Engineering
Industrial Engineering, Sofia 1000, Sofia
BULGARIA
http://www.wseas.org/mastorakis

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. The combinatorial approach to software testing uses models, particularly an Orthogonal Array Testing Strategy (OATS) is proposed as a systematic, statistical way of testing pair-wise interactions to generate a minimal number of test inputs so that selected combinations of input values are covered. 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 favors 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: - Software Testing, Virtual Machines, Environment Virtualization, Combinatorial testing

1 Introduction

Applications are usually executed in very complex environments, which consist of: multi client and server machines, different operating systems, a large number of applications written in different programming languages, different database, as well as networks that connect all these components. 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 [1-8]. Our research [2][1] concluded that test application, which will work in such environments, requires significant test resources, such as: costs for the purchase of hardware and software required, the cost of raising the hardware configuration, the cost of creating the appropriate software configuration, operation and the time required for the tests. 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 simultaneously applying combinatorial testing and virtualization technique to

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

software testing. System testing is an integral, costly, and time consuming 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. One approach for creating a test environment is the use of virtual machines (VMs), which allow better use of hardware while, at the same time, simply and quickly set the required software configuration. Software development typically involves developing and testing for different target environments, but dedicating a physical computer to each environment can be expensive. Besides the initial purchase cost, physical computers take up space, use power, and require maintenance. Virtual machines can reduce this cost by providing a way to run multiple development and test environments on one physical computer. Another problem with dedicating a physical computer to each environment is that setting up target environments can be quite time consuming. In this situation, virtual machines can save the time. In order to duplicate a particular environment, it is possible to create a library of virtual hard disks, that are pre-loaded with specific sets of software. Test team can clone the disks that they need and quickly replicate a particular environment in a virtual machine.

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, 3, 11]. 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 presents that the combinatorial testing and virtualization together can dramatically improve the process of testing. The 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.

2 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 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.

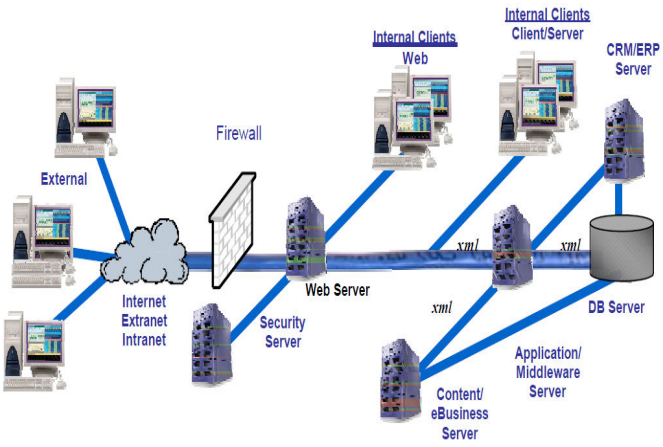

Figure 1. A Typical Multi-Tier IT Architecture Today

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 [6, 12, 13-15].

Our framework, Integrated and Optimized Software Testing Process - IOSTP [2], 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.

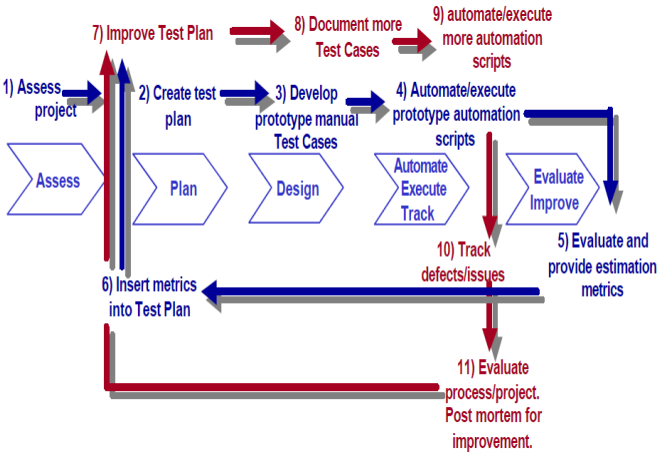

Figure 2. The IOSTP's Test Automation Process (TAP) for End-To-End Architecture Testing

3 Combinatorial testing

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.

3.1 Orthogonal Array Testing Strategy (OATS) and Techniques

The Orthogonal Array Testing Strategy (OATS) provides representative (uniformly distributed) coverage of all variable pair combinations. This makes the technique particularly useful for integration testing of software components (especially in OO systems where multiple subclasses can be substituted as the server for a client). 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 color, and page layout). What degree of interaction occurs in real system failures? Within the NASA database application [11], 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. Dr. Genichi Taguchi was one of the first proponents of orthogonal arrays in test design. His techniques, known as Taguchi Methods, have been a mainstay in experimental design in manufacturing fields for decades. Orthogonal arrays are two dimensional arrays of numbers which possess the interesting quality that by choosing any two columns in the array you receive an even distribution of all the pair-wise combinations of values in the array. The method of orthogonal arrays is an experimental design construction technique from the literature of statistics. In turn, construction of such arrays depends on the theory of combinatorics. 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. The size and shape of the array depend on the number of parameters and values in the experiment. Orthogonal arrays are related to combinatorial designs. 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. The size and shape of the array depend on the number of parameters and values in the experiment.

Definition 1: 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, …, n – 1, where $n = f(n_0, ..., n_{k-1})$. Typically, this means that each parameter would have (up to) n values.
- d is the strength of the array (see below).

An orthogonal array has strength d if in any $\rho \times d$ sub-matrix (that is, select any d columns), each of the $n*d$ possible d-tuples (rows) appears the same number of times (>0). In other words, all d-interaction elements occur the same number of times.

Here is some terminology for working with orthogonal arrays followed by an example array in Table 1 [2,10]:

- **Runs** - ρ: the number of rows in the array. This directly translates to the number of test cases that will be generated by the OATS technique.
- **Factors** - k: the number of columns in an array. This directly translates to the maximum number of variables that can be handled by this array.
- **Levels** - n: the maximum number of values that can be taken on by any single factor. An orthogonal array will contain values from 0 to *Levels-1*.
- **Strength** - d: the number of columns it takes to see each of the $Levels^{Stength}$ possibilities equally often.
- Orthogonal arrays are most often named following the pattern $\mathbf{L_{Runs}(Levels^{Factors})}$.

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 (using the orthogonal array given in Table 1) 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.

Table 1. An $\mathbf{L_9(3^4)}$ orthogonal array with 9 runs, 4 factors, 3 levels, and strength of 2

	Factors			
Runs	0	0	0	0
	0	1	1	2
	0	2	2	1
	1	0	1	1
	1	1	2	0
	1	2	0	2
	2	0	2	2
	2	1	0	1
	2	2	1	0

The test set could easily be augmented if there were particularly suspicious three- and four-way combinations that should be tested. Interaction testing can offer significant savings. Indeed a system with 20 factors and 5 levels each would require $\mathbf{5^{20}}$ = **95 367 431 640 625** i.e. almost $\mathbf{10^{14}}$ exhaustive test configurations. Pair-wise interaction testing for $\mathbf{5^{20}}$ can be achieved in **45** tests. But what if some failure is triggered only by a very unusual combination of three, four, or more values? It's unlikely that our 45 tests would detect this unusual case. We would need to test at least three- and four-way value combinations. Combinatorial testing beyond pair wise is rare, however, because good algorithms for higher strength combinations haven't been available or were too slow for practical use. In the past few years, advances in covering-array algorithms, integrated with model checking or other testing approaches, have made it practical to extend combinatorial testing beyond pair wise tests [11]. If some failure is triggered only by an unusual combination of more than two factor interactions, how many testing combinations are enough to detect all errors? What degree of interaction occurs in real system failures? Surprisingly, researchers hadn't studied these questions when the US National Institute of Standards and Technology (NIST) began investigating causes of software failures in 1996 [11], as we already mentioned above. Study results showed that, across various domains, all failures could be triggered by a maximum of four- to six-way interactions. As Fig. 3 shows, the detection rate increased rapidly with interaction strength. 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.

These results are not conclusive, but they suggest that the degree of interaction involved in faults is relatively low, even though pair wise testing is insufficient. Testing all four- to six way combinations might therefore provide reasonably high assurance.

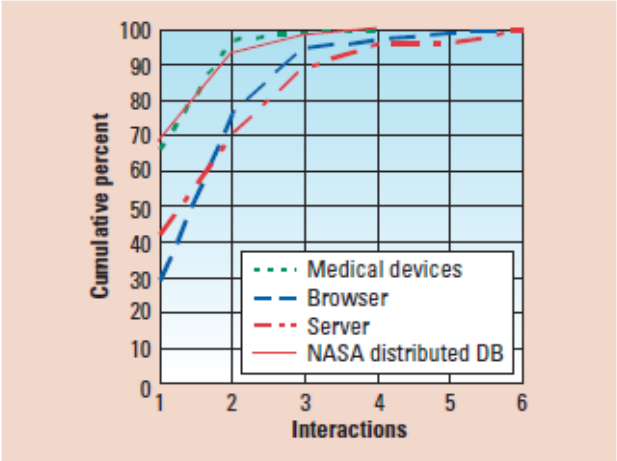

Figure 3. Error-detection rates for four- to six-way interactions in four application domains: medical devices, a Web browser, an HTTP server, and a NASA distributed database [11].

3.2 How to use this technique

The OATS technique is simple and straightforward. The steps are outlined below. The OATS technique is simple and straightforward. The steps are outlined below.

1. Decide how many independent variables will be tested for interaction. This will map to the **Factors** of the array.
2. Decide the maximum number of values that each independent variable will take on. This will map to the **Levels** of the array.
3. Find a suitable orthogonal array with the smallest number of **Runs**. A suitable array is one that has at least as many **Factors** as needed from Step 1 and has at least as many levels for each of those factors as decided in Step 2.
4. Map the **Factors** and values onto the array.

5. Choose values for any "left over" **Levels**.
6. Transcribe the **Runs** into test cases, adding any particularly suspicious combinations that aren't generated.

In a process of the combinatorial testing, tester generates tests that cover all double, triple or n-pairs combination of test parameters defined in the formal requirements for testing. Coverage of the pairs combination means that for any two parameters P_1 and P_2, and any valid values for the v_1 for parameter P_1 and v_2 for parameter P_2, there is a test in which the P_1 has the value v_1 and P_2 has the value v_2 [1-3].

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:

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

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

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

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	1,275
5-way	4,220
6-way	10,902

Case studies [2-4, 10] give evidence that the approach compared to conventional approaches is:

- more than twice as efficient (measured in terms of detected faults per testing effort) as traditional testing,
- about 20% more effective (measured in terms of detected faults per number of test cases) as traditional testing.

It is appropriate that the combinatorial testing uses orthogonal arrays and all-pairs algorithm for providing the following advantages [2]:

- Significantly reducing the cost and raises the quality of testing is achieved by intelligent generating test cases,
- Dramatically reduced overall number of test cases compared to exhaustive testing,
- Detects all faults due to a *single parameter* input domain,
- Detects all faults due to *interaction of two parameter* input domains,
- Detects many faults due to *interaction of multiple parameter* input domains

At this time, combinatorial testing is a very mature technique of testing, supported by a large number of tools to generate test cases [4].

4 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 para-virtualization are the best for software testing [5-6].

VM can simulate very realistic software configurations. Applications, which are tested on the VM can be assigned to different hardware and software resources, where the test can be done at the same time. During the test it is possible to use VMs library, which includes a set of previously created virtual machines, ready to be used. The most important advantages of using virtual machines are: reduction of costs; isolation of applications, easier testing, standardization of testing and portability.

Optimizing virtual environments can offer significant benefits to your virtual infrastructure. Some of the platform's benefits experienced in native environments apply to virtualized systems. But, making adjustments in processor affinity, memory, and how you deploy VM disk storage can help improve performance further. It depends on many factors, both specific to the virtual machine manager (VMM) being deployed, types of users, and the particular workloads being run, and general optimization parameters available on the servers. You should also consider recommendations by your VMM developer, and possibly experiment on non-

production environments to achieve added performance benefits for your particular workloads, users, and system hardware. We apply the techniques outlined in this paper for many benchmarks.

4.1 Reduction of costs

Total Cost of Ownership (TCO) in the organization is defined as a cost of possession, exploitation and maintenance of computer systems. TCO also includes costs for hardware and software, as well as the cost of installation, training, support, upgrade and repair. The best practice is to always test upgrades, patches and new applications in a non-production environment that emulates your production network as closely as possible. However, purchasing the hardware to create a parallel network can get expensive. It's much more cost effective to create your test network in VMs on one or a few physical computers. Each VM operates as a separate member of the network, with its own IP address.

Virtualization reduces TCO, in the following ways [7]:

- Increase system utilization (existing servers are used less than 10% of the capacity)
- Reduces the necessary hardware (about 25%)
- Contributes in preserving the environment, reduces energy consumption and reduces the required level of air conditioning (operating costs are reduced by about 50%).

Taking into account the money needed to obtain software licenses for the host machine, compared by costs for virtual machines. The cost can be further reduced, using at the same time integrated environment for development, as well as for the purpose of the testing. Currently, licenses costs are the same for traditional installation, as well as for an installation of the virtual environment. In the near future is expected decreasing the cost of licensing software for virtual environments.

4.2 Application Isolation

Virtual machines allow isolation of individual, or application groups, in their own environment, which can be run on the same physical machine. Besides, except for reducing the required hardware resources, simplifies and hardware management.

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. 4.

4.3 Easy test plans execution

VM make test plans easy to be executed. The most of the VM provide features of state recording (snapshot) and the return to the previous state (rollback). This means that it is possible to stop the VM, record the current state, and return back, as often as necessary. Also, it is possible to run a new test in a "clean" machine, without affecting the previous installation of the already tested software.

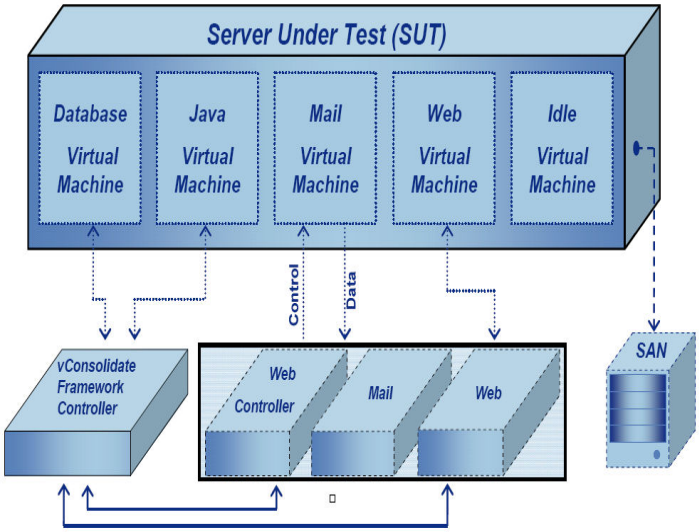

Figure 4. Typical VMs test configuration

4.4 Standardization

Applying the virtual machine, it is possible to ensure standardization of systems. Different virtual machines, as an OS guest, run on the same, standardized hardware. Standardized hardware platform reduces the cost of testing, and sharing the same hardware environment increases the efficiency of IT resources. In practice, different host machines often belong to different generations of hardware. Furthermore, in these cases, behaviour of the VM still is the same. Also, raising more VM on a single hardware platform, which is checked and reliable, it is possible quickly to locate causes of errors, made in testing the application, and to reduce maintenance costs of hardware for the purpose of testing.

4.5 Portability

Individual VM can be easily moved from one physical machine to another. The most VM software puts your drive within the host environment in the form of only one file. At the same time, the state

snapshots are recorded in a separate file on the host system. Virtual machines allow for binary compatibility between platforms. Therefore, in the case of the VM relocation from one to another host machine, it is enough, instead of VM install, just copy the virtual disk file and snapshot to the new host machine's drive.

For example, an application workload may grow over time due to changing business requirements. As a result, the memory or compute resources of the physical server hosting the workload may become constrained. When this happens, additional capacity can made available by migrating the virtual machine (VM) containing the application to a less-utilized server.

Live migration enables us to perform such migrations within a server cluster without interrupting the services the VMs are providing. It reduces the extra work traditionally involved with moving VMs, including notifying users, shutting down the applications, moving the VMs to new servers, and then restarting the VMs and each of the applications. Eliminating these steps improves flexibility and efficiency in managing data centre resources.

Live migration provides the foundation for advanced data centre capabilities such as:

Dynamic load balancing

When resources such as processor or memory are constrained on one physical server, we can utilize additional capacity available in the cluster by live migrating VMs to a less heavily loaded server in the same cluster.

Maintenance without VM downtime

We can perform server maintenance, upgrades, and refreshes without incurring VM downtime by using live migration to move VMs off the host server before we shut it down.

Advanced load-balancing scenarios

Scenarios such as power-aware load balancing and support for affinity and anti-affinity rules determine the allocation of virtualized workloads to physical servers (for example, certain VMs can be hosted on the same server for performance reasons).

4.6 Disadvantages

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. Because they contend for resources in this way, the following are not good uses of virtual machines:

- Performance and stress testing. Results may not be accurate because the amount of resources available for a given operation can fluctuate.
- Running multiple resource-intensive virtual environments on the same physical computer. Performance will be sub-optimal unless Tester's computer is sized adequately. Tester's host computer must have the sum of all of the physical resources required by the running virtual machines, plus what the host system needs, plus about another 10 percent for overhead.

5 Configuration testing in virtual environment

Configuration testing is the process of testing the system on a machine with different combinations of software and hardware. The number of possible combinations, which are supplied as a Cartesian product, often is too big to be tested out for every single combination. For example, Web applications testing could be covered by a huge number of possible combinations of: OS version, browser, Web server, etc. In the case of 8 different versions of OS, 7 different versions of browsers, 6 different web servers, and only 10 localizations, the number of different configuration is: 8x7x6x10 = 3360. Adding different software components, as well as Plug-in's and ActiveX controls for the different versions of Web browser, as well as setting the browser's options, the number of possible combinations exceeds all objective testing that can be done. However, if orthogonal arrays are applied, in this case (4 factors, i.e. with parameter 8, 7, 6, and 10-level variations, respectively) there are only 100 combinations to test on $O(2^4,10^4)$. Having in mind that, in the case when the number of combinations is very large, it is practically impossible to carry out testing within the available time, budget and within other resources. One of the possible approaches to software configuration testing is the use of VM.

5.1 Test configuration

The main characteristics of test configurations depend on the software applications and hardware and software environments:

- Software applications are intended for use in different environments. Characteristics of the environment, or environment factors, which should be considered, are: system software, network connections and hardware platform.
- Particular environment is defined by a combination of hardware and software.
- Each individual surrounding matches a set of values for each of these factors. Test configuration is a single combination of environmental factors.
- Example of a typical test configuration, in the present applications, is shown in Fig. 5:

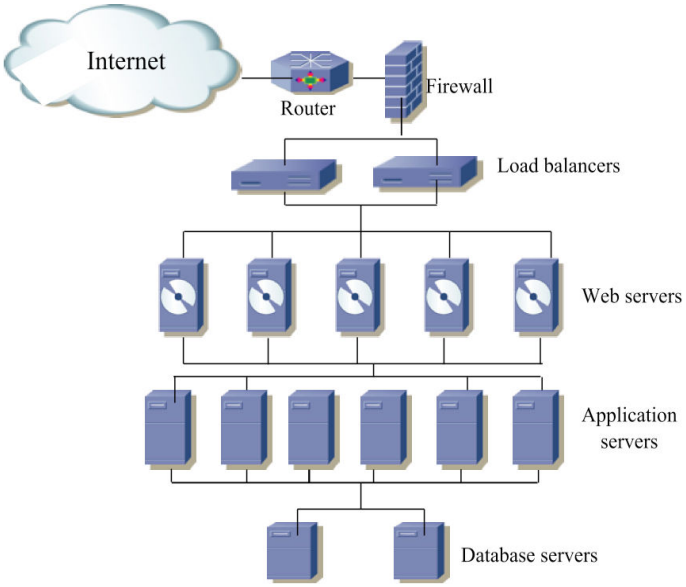

Figure 5. Typical test infrastructure

- One of the possible configurations is: Windows XP, ADSL connection and a PC with 2GB RAM.
- Combining of certain versions of an operating system and drivers for the printer, it is possible to create several test configurations for the printer.
- To reach the high reliability of the planned environment, the application must be tested in a large number of test configurations, or environments.

Virtual machines could be attached to physical networks just as if they were physical, or they could structure a virtual network for testing different scenarios, while isolating virtual machine network traffic to the host computer. This is useful for patching virtual machines, providing general network access to them, and validating different network scenarios that might be relevant in software testing. As previously mentioned, library of virtual hard disk files could be created and used to recreate a particular environment.

5.2 Combinatorial test design process

1st Step

Modelling the input space and / or configuration space. The result of modelling is expressed by factors and corresponding levels of those factors.

2nd Step

An obtained model represents the entrance to the procedure of the combinatorial design, where the combinatorial object is to be generated. The combinatorial object often is called "the design factor" and it is represented by a set of factors and levels.

3rd Step

Generated combinatorial object can be used to design a test set or a test configuration. What will be designed depends on the demands.

Within the combinatorial test design, it is possible to automate the 2nd and the 3rd step. A combinatorial test process designing is shown on the figure 6.

6 Application - Case Study

Combinatorial approach can be applied to configuration testing as outlined below [2]:

- Testing of complex systems with multiple configurations,
- Interoperability testing,
- Web testing,
- Known that faulty interaction between system components is a common source of system failures,
- Re-use existing suite of (system) test cases,
- Test at least for all two-way interactions among various system components because exhaustive testing (i.e. executing a suite of test cases for *all* possible configurations) cannot be afforded,
- Assume that the risk of an interaction failure among three or more components is balanced against the ability to complete testing within a reasonable budget,
- Calculate the minimal set of test configurations that test each pair-wise combination of components.

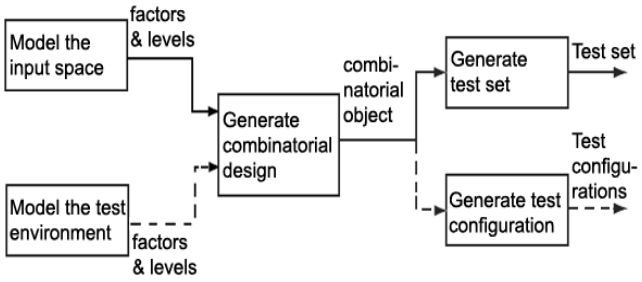

Figure 6. Combinatorial test design process [8]

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

Insurance company plans to offer, to local customers, tools for calculating the price of the premium for a property and a life insurance. It is important that the value of premiums and insurance will be displayed in the local currency. Furthermore, dates of an insurance period would be presented by localized format. As verification and calculation is to be on the client side, it is necessary to allow cookies receive, and it is necessary to provide JavaScript and ActiveX controls support. Also, it is necessary to show the text in the appropriate language and script, while the look and functionality of pages should be the same, regardless of the browser to be used.

Application software was created by a small IT agency (with four employees), using very limited hardware resources.

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 (parameter values for the selected browser on the basis of reports NetApplications Corporation in January 2009. The [9]):
   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 (Windows XP, Internet Explorer 7, English, JavaScript allowed, allowed cookies, ActiveX controls, allowed).
   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. Using PICT tool [4], there were created only 38 needed test cases, which cover all pairs of parameters. It should be emphasized that the tool does not use weight factors in such cases when there exist two, potentially contradictory requirements: the coverage of the combination with the lowest number of test cases and the selection of values in accordance to their defined weight.
5. Virtual machines are created under defined configurations for testing.

Generated test cases are shown in Table 2.

6.1 Advantages and uses

Advantages and benefits of the combinatorial test techniques, aimed for application testing, it is possible to realize even in the case of the small dimension systems. Thus, in previous example, 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.

Due to the fast settings and quick access to different test configurations, at only one physical host machine, testing was done much faster, sparing costs for the provision additional hardware, required for the physical settings, test configuration generation.

Table 2. Generated test cases

Generated test cases for case study

Browser	OS	Local	JS	Cookies	ActiveX
FF 2	Linux	Serb	N	N	Y
FF 2	Vista	Alb	Y	Y	N
FF 2	Vista	Hung	Y	Y	Y
FF 2	Win2k	other	Y	N	Y
FF 2	WinXP	Croat	Y	Y	Y
FF 2	WinXP	Slov	Y	Y	Y
FF 3	Linux	Alb	N	N	Y
FF 3	Vista	Croat	N	N	Y
FF 3	Vista	Serb	Y	Y	Y
FF 3	Win2k	other	Y	Y	N
FF 3	WinXP	Hung	Y	Y	Y
FF 3	WinXP	Slov	Y	Y	Y
IE6	Linux	Hung	Y	Y	Y
IE6	Linux	Slov	N	Y	Y
IE6	Vista	other	Y	Y	Y
IE6	Win2k	Alb	Y	N	Y
IE6	WinXP	Croat	Y	N	N
IE6	WinXP	Serb	Y	Y	Y
IE7	Linux	Alb	Y	Y	Y
IE7	Vista	Slov	N	N	N
IE7	Win2k	Croat	N	Y	Y
IE7	Win2k	Hung	N	Y	Y
IE7	Win2k	Serb	Y	Y	Y
IE7	WinXP	other	N	N	Y
IE7	WinXP	Serb	Y	Y	Y
Opera	Linux	Croat	Y	Y	N
Opera	Linux	other	Y	Y	Y
Opera	Linux	Serb	N	Y	N
Opera	Vista	other	Y	N	N
Opera	Win2k	Slov	Y	N	Y
Opera	WinXP	Alb	Y	Y	Y
Opera	WinXP	Hung	Y	N	N
Safari	MacOS	Alb	Y	N	Y
Safari	MacOS	Croat	Y	Y	Y
Safari	MacOS	Hung	Y	Y	Y
Safari	MacOS	other	N	Y	Y
Safari	MacOS	Serb	N	Y	N
Safari	MacOS	Slov	Y	Y	Y

7 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.

Now more than ever, it's clear that virtualization makes good business sense. For many companies, the question isn't, "Should we virtualize?" but rather, "How can we transition to a virtualized environment to increase business benefits and cost effectiveness?" Companies with more current technology can also benefit. Today's companies are harnessing the power of virtualization to consolidate resources, enhance energy efficiency, increase compute capabilities and reduce total cost of ownership (TCO), all while achieving greater business flexibility. Today's solutions can accommodate a variety of hardware architectures in the same resource pool. By investing in an end-to-end virtualized environment now, you can add new hardware to existing virtualization pools whenever your organization needs additional compute power.

We demonstrated in this paper, the example of testing the compatibility of an Web applications testing on different platforms, in order to check whether the request is fulfilled: "Web application X can be used by different clients (Web browser), different customized and ran on different operating systems". 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 total number of test cases, regarding the defined variables and their values for all combinations, were 1440. Using PICT tool [4], there were created **only 38 needed test cases**, which cover all pairs of parameters.

The execution of the planned tests experienced that the used VM software solutions ran stable. Test process was relatively easy to manage, and time required for test configuration, execution and repetition of test cases and reset the state of the system was acceptable. 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.

References

[1] D. M. Cohen, S. Dalal, J. Parelius, G. Patton, "The Combinatorial Design Approach to Automatic Test Generation", IEEE Software, pp. 83-87, September 1996.

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

[3] J. Czerwonka, "Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios", Microsoft Corporation, Software Testing Technical Articles, February 2008.

[4] http://www.pairwise.org/tools.asp.

[5] S. Seetharaman and K. Murthy B.V.S., "Test Optimization Using Software Virtualization", *IEEE Software*, vol. 23, no. 5, pp. 66-69, Sep./Oct. 2006.

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

[7] Van Doorn, L., "Hardware Virtualization Trends", Keynote presentation at the Second International Conference on Virtual Execution Environment, 2006, Ottawa, Canada

[8] A. Mathur, "Foundations of Software Testing", Addison-Wesley Professional, ISBN-10: 8131716600, 2008.

[9] Market Share by Net Applications http://marketshare.hitslink.com/os-market-share.aspx

[10] A. W. Williams, "Software components interactions testing: coverage measurement and generation of configurations", PhD thesis, Computer Science, Ottawa-Carleton Institute for Computer Science, School of Information Technology and Engineering, University of Ottawa, 2002.

[11] D.R. Kuhn, Y.Lei, R. Kacker, "Practical Combinatorial Testing - Beyond Pairwise", IEEE IT Professional, June 2008.

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

[13] 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.

[14] Lj. Lazić, N. Mastorakis, Cost Effective Software Test Metrics, WSEAS TRANSACTIONS on COMPUTERS , Issue 6, Volume 7, June 2008.

[15] Lj. Lazić, N. Mastorakis, "The COTECOMO: COnstractive Test Effort COst Model. WSEAS 10th EUROPEAN COMPUTING CONFERENCE in Vouliagmeni Beach, Athens, Greece, September 25-27, 2007 (abstract), full paper is published in SPRINGER VERLAG, Proceedings of the European Computing Conference, Volume 2, Series: Lecture Notes in Electrical Engineering, Vol. 27 , Mastorakis, Nikos; Mladenov, Valeri (Eds.), ISBN: 978-0-387-84813-6, 89-110. (June 2009)