Assessing the Applicability of a Combinatorial Testing tool within an Industrial Environment

Peter M. Kruse¹, Onn Shehory², Nelly Condori-Fernandez³, Tanja E. J. Vos³, Daniel Citron², and Bilha Mendelson²

¹ Berner & Mattner Systemtechnik GmbH, Berlin, Germany
  peter.kruse@berner-mattner.com
  http://www.berner-mattner.com/

² IBM Research Lab, Haifa, Israel
  {onn,citron,bilha}@il.ibm.com
  http://www.haifa.il.ibm.com/

³ Universidad Politécnica de Valencia, Valencia, Spain
  {nelly,tvos}@pros.upv.es
  http://www.pros.upv.es/

Abstract. This paper describes a case study executed to evaluate a combinatorial test design approach within the industrial setting of IBM Research. An existing combinatorial test suite was compared against a prioritized combinatorial test suite that was generated with the Combinatorial Tree Editor XL Professional (CTE). The prioritization technique was recently developed and added to the CTE in the context of the FITTEST project. Test design for the new test suite was carried out by the developers of the prioritization technique. Test implementation and execution was done by the industrial partner of the System Under Test. This case study has investigated whether the prioritized combinatorial technique is useful to complement current testing practices at IBM Research. The focus of this study is on fault finding capability of artificially injected faults that have been selected and prioritized using domain knowledge and expertise, and efficiency of test case execution. Conclusions of this study are that for the testing of the target product in a simulated environment, the improved combinatorial testing tools do qualify as useful and this type of testing will be included in current practices.

Keywords: Combinatorial testing, CTE XL tool, case study, applicability evaluation

1 Introduction

In software testing, testers are often faced with too many test cases to be performed for real large-scale systems and consequently testers have to solve the problem of selecting reasonable subsets.
Combinatorial test design techniques can help testers with this problem by systematically sampling subsets [10]. In contrast to other (e.g. random) techniques, combinatorial testing comes with reliable coverage levels, i.e. test suites conforming to certain coverage levels therefore have a guaranteed quality in terms of covered parameter interactions.

A coverage level $t$ determines the degree of parameter interaction. So for $t = 1$, each parameter value is used at least once in the resulting test suite, it is combined only with itself. For $t = 2$, each parameter value is combined with each single other parameter value at least once (that is why it is also called pair-wise). The highest possible $t$ is determined by the number of parameters for the system under test. With $t=n$, all possible parameter interactions are contained in the resulting test suite. Coverage levels with $t < 1$ are undefined.

For real world systems, parameters are typically not freely combinable, as there are interdependencies between the values of different parameters.

Several combinatorial testing techniques have introduced different ways of specifying these constraints [5]. This significantly reduces the test suite size while delivering a guaranteed level of parameter coverage.

In this paper we present a case study for evaluating the CTE XL Professional tool [11] (the Classification Tree Editor, or CTE for short, developed and commercialized by Berner&Mattner) within a realistic environment of the IBM research lab in Haifa. An industrial system, called IT Management Product (IMP) was used as system under testing (SUT). It is a large-scaled networked system that controls and optimizes resource management, such as creating and reconfiguring virtual machines on demand.

Several studies can be found that evaluate combinatorial testing techniques in a controlled experimental environment (e.g. [7], [2], [3]). However, more feedback from practical applications of combinatorial testing is still needed [4]. Consequently, in the context of the FITTEST project [21] we have initiated work on creating more evidence about the application of combinatorial testing in real settings. In [15] a study can be found that presents a case done at a company called Sulake to evaluate the effectiveness and efficiency of using model-based combinatorial testing, supported by the above mentioned CTE, within their industrial setting of testing Habbo hotel. Another study can be found in [8], which reports on a case conducted for evaluating the learnability of the CTE within a company called SOFTEAM.

This study is the third in this series and aims to get even more empirical evidence of the applicability of this testing technique within the context of a real testers team and an industrial system. Specifically, we focus on analysing two different test generation strategies: i) the first one supported by the CTE, and ii) the second one supported by the current IBM Research testing practices.

The paper is organized as follows: Section 2 presents the case study design. Section 3 discusses the main threats that were identified and mitigated in this
case study. In section 4, we report the obtained results. Finally, section 5 concludes the paper.

2 Design of the case study

In this section, we present the planning and design of the case study, which was carried out according to our framework proposed for empirically evaluating testing tools [20] and guidelines proposed by [16].

2.1 Objective - What to achieve?

As indicated above, the main motivation of IBM Research was to find out whether the combinatorial testing tool CTE XL Professional [11] can help improve the current combinatorial testing practices of the IBM research lab. Consequently, in this case study, we have decided upon the following research questions to investigate the usefulness of the CTE XL Professional within the IBM environment:

RQ1 [Effectiveness] Can the test cases generated by the CTE finding faults that are marked as important by testers of the selected SUT at IBM Research?

RQ2 [Efficiency] How long does execution of the test cases generated by the CTE take when it is used in the testing environments at IBM Research?

2.2 Context- where was the case study performed

The study was executed at IBM Research Lab in Haifa, where the testing itself is conducted by a designated research team. This research team is responsible for building the testing environment for future developments of an IT Management Product (IMP) (similar to [1]), a resource management system in a networked environment (more details in Section 2.3). This research team tests the IMP system in a simulated testing environment, this way enabling the testing of new versions of the IMP. The subjects with whom this study was conducted consisted of three senior testers from IBM research team, who have a high knowledge level about the IT Management Product(SUT).

Moreover, given that one of the objectives of this case study is to evaluate the fault detection capability of the CTE tool, in order to avoid and reduce some threats, the testing process is also supported by one senior tester from the B&M company (owner of the CTE XL professional tool). This way, the test models and abstract test cases will be generated in better conditions. However, the concretization of these abstract tests will be carried out by the IBM testers, by using the simulated testing environment.

Finally, since the IMP is a mature system, and in order to be able to measure fault-finding capability, several faults were injected into it within the simulated environment to mimic potential problems that had can be surfaced in such a system.
2.3 Objects of the study: the SUT and the injected faults

The System under Test (SUT) The IBM IT Management Product (similar to [1]) is a distributed application for managing system resources in a networked environment. It consists of a Management Server (MS) that is an application that communicates with multiple managed clients and with users of the management system (typically system administrators). Managed clients are physical or virtual resources distributed over a network.

The IT Management Product system is used by IBM customers for managing IBM hardware and virtual devices, such as servers, Virtual Machines (VMs), switches and storage devices. The application has been developed for several years by IBM. IBM considers this application to be an important product, and hence is keen on assuring its quality through testing. IBM Research is developing the simulated environment to allow better testing of the system. Consequently, this simulated environment seems a good object for study.

The case study is to be performed on some new versions of this simulated system which are still under development and have not yet been released for customer use. Consequently, the case study system shares structure and protocols with the versions available to customers. As a result, public resources available for production version are relevant to the case study as well.

The term end-point is used in reference to the managed resources on client nodes, from the point-of-view of the management server. The MS executes operations on the end-points either per requests from the users, or autonomously based on defined policy (e.g. a recovery attempt after an end-point failure). The end-points have two-way communication with the MS. The communication operations supported are listed below (not all available for testing in this case study though): 1. Discovery: MS finds what end points are reachable; 2. Control: MS sends commands (configuration, Operative System restart) to end-points; 3. Query: MS queries status and configuration of end-points; 4. Report: End-points report problems to MS, and recovery from problems; and 5. Deploy: Deploy and obtain disposed of end-points.

The MS keeps an inventory of the current managed resources and their states in a standard database. The case system supports several protocols and message structures. Yet, in the case study, we are focusing only on HTTP messages. In the case system, these are the easiest to capture, interpret, and modify for case study purposes. Below we present an example of two messages, a request GET and a response in JSON format.

```
2013:02:28:17:20:08;GET
https://.../ibm.com:8422/ibm/.../rest/IMP/workloads/500165838;;200;{
    "workload" : {
        "hosts" : {
            "uri" : "https://.../workloads/500165838/hosts"
        },
        "createBy" : "root",
        "resilient" : "false",
        "approvalRequired" : "false",
```
In this message, at time 2013:02:28:17:20:08 there was a request to get the status of the workload with id 500165838. On successful execution this service returns an HTTP status code of 200 (OK) and the response in JSON format containing relevant information about the workload. A full list of the possible HTTP requests in the API of IBM IT Management Product can be found at [1].

The faults injection In order to evaluate the fault-finding effectiveness of the CTE tool, IBM Research injected manually 10 representative faults into the SUT. Those 10 injected faults were based on real faults that were identified in earlier versions of the product. Table 1 lists these faults together with their respective severity level.

<table>
<thead>
<tr>
<th>Num</th>
<th>Injected Fault type</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF1</td>
<td>Fails to create unique new IDs</td>
<td>LOW</td>
</tr>
<tr>
<td>IF2</td>
<td>Fails to open a JSON file</td>
<td>LOW</td>
</tr>
<tr>
<td>IF3</td>
<td>Fails to validate workload size when resizing</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>IF4</td>
<td>Fails to delete workload fails</td>
<td>HIGH</td>
</tr>
<tr>
<td>IF5</td>
<td>Fails to find a specific object ID</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>IF6</td>
<td>Fails to create workload when resources are not enough for a deployment</td>
<td>HIGH</td>
</tr>
<tr>
<td>IF7</td>
<td>Fails to create a virtual server</td>
<td>HIGH</td>
</tr>
<tr>
<td>IF8</td>
<td>Fails to create a virtual disk</td>
<td>HIGH</td>
</tr>
<tr>
<td>IF9</td>
<td>Fails to get compatibilities</td>
<td>LOW</td>
</tr>
<tr>
<td>IF10</td>
<td>Fails to write to a JSON file</td>
<td>MEDIUM</td>
</tr>
</tbody>
</table>

For example, if the faults IF10, IF5, IF6, and IF1 are injected, the following errors (failures) are reported by the system:

– Cannot create Storage Volume ID
– Cannot link storage volume HASH
– Cannot link virtual server
– No such file or directory
– Cannot write customization file
– Cannot update assigned disk information
– Cannot create Operating System
– Cannot update Server

In order to distinguish these faults from others, a comment in the log files generated by IBM Research’s simulation environment was added which started with “Injected bug”.

2.4 Subjects - Who apply the techniques?

The subjects who applied the technique are employees of Berner&Mattner (B&M) and IBM Research.

The B&M subject was a 32-year old tester from B&M with 7 years of software development experience, 5 years of experience with testing. The test designer has previous knowledge on the CTE XL Professional tool and holds a Computer Science degree, as well as a Certified Tester Foundation Level certificate.

The IBM Research subject was a 30-year old senior tester from IBM Research with 10 years of software development experience, 5 years of experience with testing of which 4 years with a system similar to the IMP and the approach described in sub-section 2.3.

Moreover, two additional IBM senior tester participated in the study for validating the combinatorial model.

2.5 Treatments - What are being studied?

In this case study we identified the test cases generation strategy as main independent variable, which includes two treatments: The generation strategy supported by 1) the combinatorial testing tool CTE XL Professional [9] and 2) the current testing practices implanted at IBM research lab.

The Classification Tree Editor CTE XL Professional The classification tree method developed by Grochtmann and Grimm offers a graphical notation for specifying test parameters [6]. For each influence factor (e.g. parameter) of the system under test, a classification is added to a classification tree. For each classification, representative classes are then added following the principles of boundary value analysis and equivalence class partitioning [13]. This results in a tree consisting of classifications and classes. For semantic purposes, classifications can be grouped using compositions. Test cases are then defined by selecting exactly one class for each classification.

An overview of the combinatorial testing process supported by the CTE XL professional tool is depicted in Figure 1
This pure test modeling approach was later extended with aspects of combinatorial test design by introducing Boolean dependency rules to describe constraints and a test generation language to formulate combinatorial coverage rules [11]. The Classification Tree Editor CTE XL allows the automatic generation of combinatorial test suites such as pairwise.

One result of the FITTEST project was the test case prioritization with the classification tree method [9]. After assigning weights to elements of the classification tree, the automated test case generation produces a test suite containing test cases in descending order of importance. The test suite can be further optimized (that is, reduced) by applying coverage criteria, i.e. only covering 90% of a usage profile.

The IBM testing process

The test scenarios designed for the IT Management Product system use high level descriptions of complex customer use-cases to support exploratory test case design. The objective is to maximize the coverage of system use-cases. In principle these test scenarios are automated by a long script. This script takes as an argument a configuration file that is used to define the different values of parameters to set the simulated environment in which the tests are run.

Deciding which combination of parameters to chose is a challenging task since there exists a wide variety of factors that influence the behaviour of the system.

Since IBM Research policies prevent us from comparing with the current way that IBM Research does combinatorial testing this study only studies the CTE XL Professional combinatorial testing tool as a case for generating a combinatorial test suite for the selected SUT.
2.6 Variables - What are being measured?

The dependent variables are those for measuring the applicability of the FITTEST tools in terms of effectiveness, and efficiency. Next we present their respective defined metrics:

1. Measuring effectiveness:
   (a) amount of injected faults detected by both TS\textsubscript{ibm} and TS\textsubscript{cte}
   (b) type of faults detected by both TS\textsubscript{ibm} and TS\textsubscript{cte}
2. Measuring efficiency. For both TS\textsubscript{ibm} and TS\textsubscript{cte}:
   (a) size of the test suites: number of test cases
   (b) time needed to execute both the test suites
3. Moreover for TS\textsubscript{cte} we will measure size metrics about the created CTE classification trees: number of classes, number of compositions

The main independent variable of this case study is the test-cases generation strategy, whose treatments are: the CTE XL professional tool, and the IBM current testing process (For more detail, see treatments section). Moreover, the type of injected faults was identified as possible factor that could affect to our dependent variables.

2.7 Protocol

The steps adopted in this case study in order for the subjects to collaborate and measure the data identified in the previous section is depicted in Figure 2. This protocol respects the business policies implanted at IBM Research that allow little or no access to external parties.

1. Describe the System Under Test (SUT), the weights of the classifications and the requirements for testing [IBM Research subjects]
2. Study this description and make a first version of the Classification Tree modelling the combinatorial aspects and the weighted priorities. [B&M subject]
3. Go through various iterations to validate the model to make sure that SUT description was interpreted the right way [skype meeting between IBM Research and B&M subject]
4. Once model has been validated, automatically generate the abstract test cases using the CTE XL Profesional and create a test suite which will be denominated TSS\textsubscript{abstract}\textsubscript{cte}.
5. Select top four abstract test cases from the TSS\textsubscript{abstract}\textsubscript{cte}.
6. Derive concrete test cases from the abstract once, designing TSS\textsubscript{concrete}\textsubscript{cte}. [IBM Research subjects]
7. Select a test suite TS\textsubscript{ibm} which will be used to compare [IBM Research subject]
8. Execute TS\textsubscript{ibm} [IBM Research subject]
9. Execute TS\textsubscript{cte} [IBM Research subject]
10. Select and inject the faults [IBM Research subject]
11. Execute TS\textsubscript{ibm} with fault injection [IBM Research subject]
12. Execute TS\textsubscript{cte} with fault injection [IBM Research subject]
3 Threats to validity

Internal validity. It is of concern when causal relations are examined. In our case study, an internal validity threat is related to the test models to be used for automatically constructing the concrete test cases. Because of the quality of test cases can be affected by the quality of test models. In order to reduce this threat, we asked B&M to validate the CTE model in collaboration with IBM (who specified the testing requirements as well as the SUT description). This was done in three iterations.

On the other hand, the derivation of concrete test cases from abstract test cases could be also affected, more even if this would have been was carried out manually by IBM team (e.g. effort in understanding abstract test cases). This threat can be mitigated, by automatizing this test cases concretization process.

Regarding to the involved subjects from IBM, although they had a high level of expertise and experience working in the industry as testers, they had no previous knowledge of the CTE XL professional tool. This threat was reduced by means of a closer collaboration between B&M and IBM, by complementing their competences in order to avoid possible mistakes in the generation of abstract test cases. However, this collaboration could have caused another type of threat to internal validity such as research bias. Being aware of this threat, we consider that it was reduced with the participation of UPV researchers, who managed the case study.

External validity.

External validity. It is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case. Our results rely on one industrial case study using a given set of faults. Moreover, given the different size of both test suites used in this case study (IBM and CTE), a same number of test cases were respectively selected in order to make a more fair comparison. The obtained results about
the applicability of CTE need to be evaluated with more SUTs. However, although running such studies is expensive in terms of time consuming, we plan to replicate it in order to have a more generalizable conclusions.

**Construct validity.**

Construct validity. This aspect of validity reflect to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions. This type of threat is mainly related to the use of injected faults to measure the fault finding capability of our testing strategies. This is because the types of faults seeded may not be enough representative of real faults. In order to mitigate this threat, the IBM team identified representative faults that were based on real faults, identified in earlier time of the development. This identification although was realized by a senior tester, the list was revised by all IBM team that participated in this case study.

With respect to the effort to set up and use the CTE tool, we could not fully mitigate the threat caused by some self-reported measures of working time (e.g. time for creating manually CTE trees). Accuracy of these measures could have been affected by other factors (e.g. social psychological actors).

## 4 Results and Discussion

A total of 5 classification trees were constructed over 5 iterations to generate five different test suites $T_{S_{cte}}$.

The Table 2 presents descriptive measures specific for the Classification Trees (CTs) over the iterations. Tree $CT_5$ was finally used to generate $T_{S_{cte}}$, which generated 117 abstract test cases (See Figure 2).

<table>
<thead>
<tr>
<th>Item</th>
<th>$CT_1$</th>
<th>$CT_2$</th>
<th>$CT_3$</th>
<th>$CT_4$</th>
<th>$CT_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of classifications</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Number of classes</td>
<td>70</td>
<td>70</td>
<td>73</td>
<td>70</td>
<td>74</td>
</tr>
<tr>
<td>Number of test cases</td>
<td>72</td>
<td>72</td>
<td>85</td>
<td>72</td>
<td>117</td>
</tr>
<tr>
<td>2-way coverage</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

However, given that the testers from IBM Research had some time restrictions for executing all the test cases from $T_{S_{ibm}}$ that has a total of 30 test cases, IBM team decided to select the 4 top most important test cases from $T_{S_{ibm}}$, which were finally used for execution in this case study. The same prioritization criteria, that were specified in the CTs, were used by IBM Research testers to select those 4. Similarly, we chose the first 4 test cases from $T_{S_{cte}}$ that were prioritized automatically by the CTE tool. This way, we had a fair comparative evaluation respect to the amount of test cases.
The concretization of abstract test cases is done simply by putting together JSON commands based on their appearance in the abstract model.

Next, we present the results obtained for each one of our research questions.

### 4.1 RQ1: Fault detection effectiveness

As is shown in Table 3, the coverage of injected faults with the TS\textsubscript{cte} is significantly higher than TS\textsubscript{ibm} (80% vs 50%). Moreover, using the TS\textsubscript{cte}, more failures were detected for the faults with a high (F4 and F6) and medium (F3 and F5) severity.

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS\textsubscript{ibm}</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TS\textsubscript{cte}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Therefore, the results indicate that the CTE XL Professional tool is more effective in finding the types of faults injected. IBM Research confirmed that the CTE tool, within the context of the studied environment, could contribute to improve the effectiveness of its current testing practice.

### 4.2 RQ2: Efficiency

As we can see in Table 4, the time to execute TS\textsubscript{ibm} without fault injection is smaller than executing TS\textsubscript{cte} (36.75 minutes vs 85.49 minutes). This can be due to the fact that pairwise coverage of TS\textsubscript{cte} is higher than TS\textsubscript{ibm} and hence there is more changing of configuration parameters which evidently takes time.

The TS\textsubscript{ibm} execution time with fault injection was greater than the time to execute TS\textsubscript{cte} (27.97 minutes vs 17.78 minutes) due to the fact that TS\textsubscript{cte} found more faults and hence would stop execution on finding these.
Overall, the execution times were considered quite acceptable by IBM Research, considering the fact that the effectiveness of the tests can be improved and more faults and failures can be detected in an efficient way (as was discussed in RQ1).

Table 4. Efficiency measures for execution of both test suites.

<table>
<thead>
<tr>
<th>Variable</th>
<th>TS_{ibm}</th>
<th>TS_{CTE}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time with fault injection</td>
<td>27.97</td>
<td>17.78</td>
</tr>
<tr>
<td>Execution Time without fault injection</td>
<td>36.75</td>
<td>85.49</td>
</tr>
</tbody>
</table>

According to the leader of the IBM Research testing team, both the CTE and the IBM test planning prove efficient enough to meet typical time-tables of the Quality Assurance teams. In light of these results, it makes sense to use the CTE tool to increase the number of fault types detected. However, combining CTE with IBM's test practices could still maximize this number of faults found. CTE can be also used as an initial automated planner as it works very fast at that stage (test modelling). Concretizing abstract test cases can be solved by the IBM team. In summary, CTE can provide benefits to the IBM testing team.

5 Conclusions and Future work

We have reported a case study for evaluating the combinatorial testing tool CTE with a real user and real tasks within a realistic environment of the IBM research lab, the IBM IT Management Product. Considering that IBM's objective was to see "which is better" (current practice versus CTE tool) the following were the results of the case study:

1. The CTE tool can improve the fault-finding effectiveness of current test suites designed at IBM, but only if used complementary to the current practice (CTE capability to model and generate abstract test cases demonstrated the usefulness of the approach for detecting a good proportion and variety of fault types (80% vs 50%).

2. The test suites generated by a combinatorial tool can be executed efficiently. The execution time for the tests generated by the CTE, although higher, was quite acceptable for IBM since it still allows testing the system in the testing time budget. Also, the use of this approach with the automation of the concrete test creation, makes it very appealing for integration this process in IBM Research. Another advantage is the tests that are small in size that allow easy bug finding in case of fault discovery.

Compared to a similar study we have executed and published here [15], we can confirm that the results are similar with respect to effectiveness and efficiency.
In both studies, the test suites generated with the CTE found more amounts of faults and also more severe faults. Also, in both studies executing the test suites generated with the CTE took more time. In the present study with IBM Research this was found acceptable and fit into their testing budget. For the company Sulake (that was subjected to the other study [15]) this longer execution time was not acceptable since they investigated the use of the CTE tests into a continuous integration testing cycle during the day and more testing time would mean less deployments a day which was not an option.

This is the 7th case study that has been published that uses our methodological framework described in [20]. The objective of this framework was to reduce some of the entry barriers for conducting case studies with companies. The framework can simplify the design of case studies for comparing software testing techniques while ensuring that the many guidelines and check-list for doing empirical work have been met. As was mentioned in the introduction, 3 of these studies were related to combinatorial testing (the current paper, [15] and [8]). The other 4 case studies using the framework can be found here: [19], [18], [12], [14] and evaluated different automated test case design tools that have been evaluated during the EvoTest [17] and FITTEST [21] project.

From these seven experiences, we can modestly say that our methodological framework in [20] indeed has simplified case study work with companies since it can be used as a vehicle for carefully planning a study and this way improving communication and technology transfer. Future work we consider are enriching the framework with other aspects that might be considered important when evaluating applicability like learnability, integration into the organization’s culture, etc. Moreover, we need to augment the body of knowledge about the costs and benefits of the tool with more evaluations, by applying qualitative methods to gather data from relevant stakeholders involved in the testing process (e.g., tester, analyst, developer).

Acknowledgment

This work was financed by the FITTEST project, ICT-2009.1.2 no 257574.

References


