Process for Unattended Execution of Test Components

Emma Torres Orue, Martha D. Delgado Dapena, Jorge Lodos Vigil, and Ezequiel Sevillano Fernandez

***Abstract*—We describe the process to perform software tests. In an enterprise that produces a product line, even if they all have the same goal, they may differ with regard to its development platform, programming language, layer architecture or communication strategies. The process allows standardizing, coordinating and controlling the test execution for all workgroups, no matter their individual characteristics. We present roles, phases, activities and artifacts to address the centralization, reusing and publication of the test scripts and the results of their execution. Additionally, it involves the virtualization for creating test environments, defining steps for its management and publication. Also is presented a tool that supports the process and allow the unattended execution of test components. Finally, we describe two pilot projects demonstrating the applicability of the proposed solution.**

***Index Terms*—Software test process, testing tools, unattended test execution, virtual laboratories.**

I. Introduction

TESTING is one of the key activities regarding software quality assurance and quality control. The testing phase should be properly planned and organized in order to prevent errors from manifesting in production and cause undesirable behavior, while minimizing the time and effort employed [1], [2]. Pressman argues that the strategy to test software must provide a map that describes the steps to be taken as part of the test plan, must indicate when they are planned and when these steps will be performed, as well as how much effort, time and resources will be consumed [1]. Several institutions are engaged in the definition of models for software quality [3], [4]. Besides, standards to fulfill the testing process have been designed, for example: IEEE 1008-87 Standard for Software Unit Testing, IEEE 1012-98 Standard for Software Verification and Validation and IEEE 829-98 Standard for Software test Documentation. At the same time, methodologies and processes have been proposed [1], [2], [5], describing activities, roles, and artifacts related to conduct tests within the software development phases.

Manuscript received on February 25, 2013; accepted for publication on May 3, 2013; final version received on June 19, 2014.

Emma Torres Orue, Jorge Lodos Vigil, and Ezequiel Sevillano Fernández are with Segurmatica, Centro Habana, Zanja 651, Havana, Cuba (e-mail: emma@segurmatica.com, lodos@segurmatica.com, ezequiel@segurmatica.com).

Marta D. Delgado Dapena is with the Informatics Studies and Systems Center in the Polytechnic Institute "José Antonio Echevarría," Marianao, 114 Ave. 11909, Havana, Cuba (e-mail: marta@ceis.cujae.edu.cu).

Software product line engineering has received much attention for its potential in the reuse of artifacts throughout the project life cycle [6], [7], [8]. Similarly to the artifacts designed during the implementation, testing artifacts have the same opportunity of being reusable taking into account the similarities identified in the product line [8], [9]. There are some studies related to the generation of test cases and building test scripts from the definition of similarities and variations within a production line [10], [11]. However, scarce references have been found related to the standardization of the execution of the different test components that can be generated in an organization that develops product line.

The automating of the execution of test components is an advantageous aspect in the validation of the elements in a production line [8], [9], [11]. It is suggested by [8] that automation allows artifacts to be tested immediately after being generated and integrated into the system. There are dissimilar solutions to automate generation and execution of test scripts [12], [13], [14], [15]. On the other hand, there are tools to achieve unattended execution of components, from

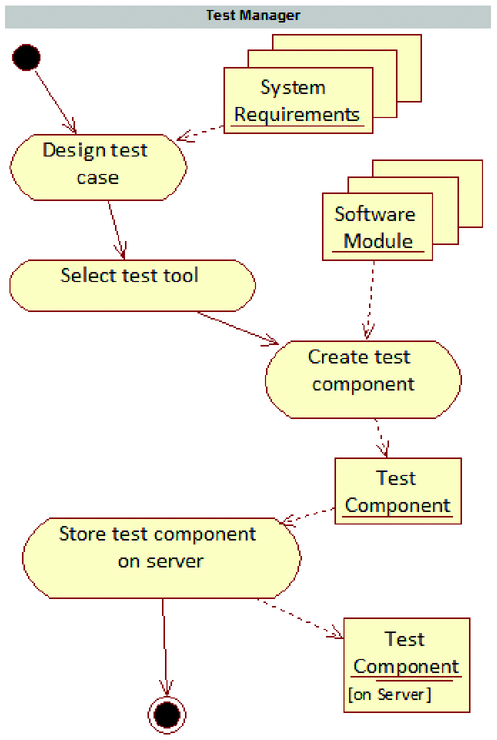


Fig. 1. UML Activity diagram of the phase Test Component Generation.

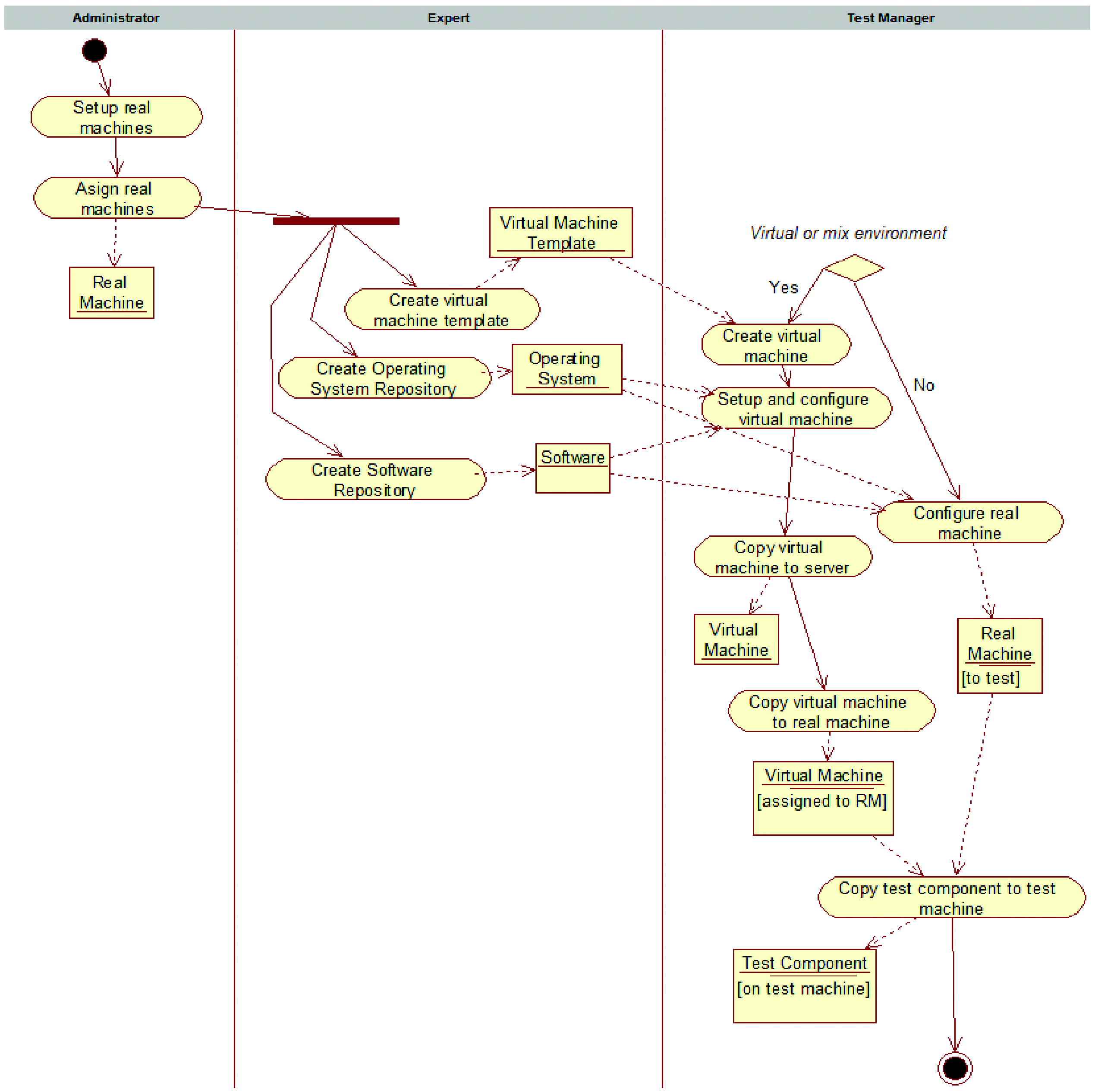

Administrator
Expert
Test Manager
Setup real machines
Asign real machines
Real Machine
Virtual Machine Template
Create virtual machine template
Create Operating System Repository
Operating System
Create Software Repository
Software
Virtual or mix environment
Yes
No
Create virtual machine
Setup and configure virtual machine
Configure real machine
Copy virtual machine to server
Virtual Machine
Real Machine [to test]
Copy virtual machine to real machine
Virtual Machine [assigned to RM]
Copy test component to test machine
Test Component [on test machine]

Fig. 2. UML Activity diagram of the phase *Test Environment Creation*.

models, coding scripts languages or test managers with friendly user interfaces [16], [17], [18], [19]. However, these solutions are designed to run scripts generated by specific testing tools. This could implicate the use of different applications for unattended test execution in order to perform all the tests in a product line.

Due to the importance of using large and diverse test environments in order to validate the artifacts in a product line, it has been decided to incorporate the use of virtual machines to facilitate the creation and maintenance of testing laboratories. The integration between virtual laboratory systems [20], [21] and testing tools [17], [18], [22] allows testing applications on virtual machines based on defined test cases. These solutions make possible to record and play back the test runs, as well as to store the results and record conditions that expose the errors for a follow-up. IBM Rational Quality Manager [2] and TestComplete [22] are proprietary solutions offered by the software companies IBM Rational Software and SmartBear respectively. The limitation of Rational’s tool is that the test scripts can only be generated by products sold by the company [2], [17]. Furthermore, TestComplete is designed only for machines with Windows operating systems.

This paper describes a process to guide the unattended test execution in real and virtual laboratories. It also shows the characteristics of a tool that enables the execution of scripts generated by any application, either directly or through a wrapper component. The article presents in the first section the phases and roles involved in the process. The second section shows the automation of the process described using the tool designed to support it. After that, two pilot projects conducted to validate the proposed solution are presented.

II. Process for the Execution of Unattended Tests in Heterogeneous Environments

The process consists of three phases: *Test Component Generation*, *Test Environment Creation* and *Test Execution and Collecting Results*. The inputs are the system modules under test, the system requirements and the associated unit tests. The outputs of the process constitute a knowledge base that stores all the information related to the test runs. Fig. 1 shows a diagram with inputs and outputs of the process. This process is divided in iterations repeated frequently and each one starts from the addition of new modules or modifications to the system in development.

The set of roles in the process includes conventional roles in the stage of software testing such as the Test Manager, the Tester and the Auditor. It also introduces the Expert and Administrator roles. The first one is responsible for making and publishing reusable artifacts for other specialists such as templates, operating systems and software. The second one manages and assigns the computers for developers and test managers, as needed.

A. *Phase I: Test Component Generation*

The test manager is the one that develops the Component Test artifact. This artifact is a test script generated by a program that automates the validation of one or more functionalities of a system module in different environments. The selected tool must ensure that the script execution results provide as much information as possible. This condition will help discover the source of error in case of failures. Finally, the created component is stored on a server that contains a repository for this artifact. The Fig. 1 displays the activity diagram of this phase.

B. *Phase II: Test Environment Creation*

The test environment creation involves the Administrator, the Expert and the Test Manager. Fig. 2 shows how the Test Manager prepares test environments based on real machines, assigned by the Administrator, and also based on previously created virtual machine templates and compiled software by the Expert. The testing machines, real or virtual, will have the basic requirements of hardware and software ensuring a successful test run. They must also guarantee the preconditions for the test execution; such as published data, configuration files, among others. Finally, the version of the system under test with its related test components is installed. Both, the templates and virtual machines created, should be stored and published on the server.

C. *Phase III: Test Execution and Collecting Results*

In the previous phase, the new versions of the software with their associated test components are incorporated. At this stage the associated test components must be run and the results stored. It is recommended to perform this task automatically and unattended, since this would allow the specialist to save time and effort. The results of execution will be stored centrally and will remain public for all specialists involved in the project. Fig. 3 illustrates the activity diagram of this phase.

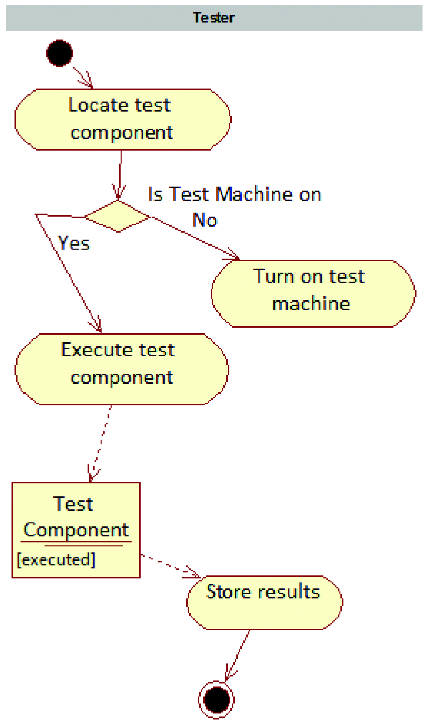


Fig. 3. UML Activity diagram of the phase Test Execution and Collecting Results.

During the implementation of the process, artifacts that record information from the test components, as well as the machine, time and physical path of where they are running, are created. Additionally, the results contain data regarding to the start date, context parameters and user who initiates the process. In Fig. 4 it is shown a class diagram, depicting as entities the repositories obtained during the process.

III. Quality Tool

Quality is a multilayer system developed by the .NET platform. Fig. 5 illustrates the relationship among the system modules. Through a web interface the test components information and its schedules can be recorded. The tool allows defining test suites by grouping test components, which may run simultaneously or sequentially. The Server Web Service provides the functionality to manipulate test labs in real and virtual environments. The Web Interface and the Server Web

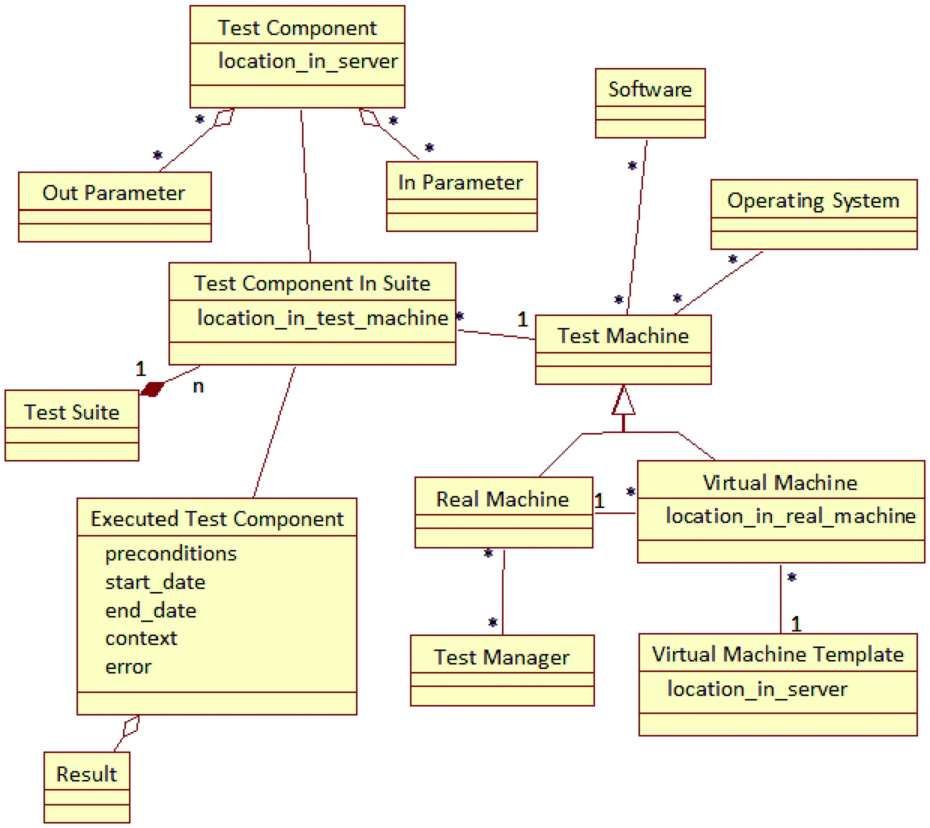


Fig. 4. UML Class Diagram with related entities in the process for running tests unattended.

Service obtain and send data to the database through the library that models the business entities.

During the stage of *Test Execution and Collecting Results*, the Server Web Service determines the real execution machine, and it communicates with the Client Web Service installed on that computer to indicate it to run the test component. According to the configuration, the Client Web Service executes the test script located in a storage device of the real machine or in a virtual machine allocated on it. The scheduled execution is managed by the SQL Agent [23]. The following describes the steps of the process automated with the Quality tool.

The phase *Test Components Generation* is performed with the help of tools available on the market to automate the execution of tests [2], [15], [16], [18], [19]. Communication libraries have been designed to create test scripts that interact with the Quality tool for storing the results and events generated from the runs. It also includes the insertion of the output parameters, which specifies whether the execution was successful or not. In case of error detection, the outputs should reflect the causes.

To handle test components generated by tools that do not allow direct use of the communication library, a wrapper executor has been created. This binary is a test component that's able to run another test script and store the outputs in the Quality System database through the communication library. The input parameters of the wrapper are the test script parameters and the path where the test component is located. Finally, the Test Manager stores the test scripts created in a server repository and inserts into the Quality system interface the scripts names, parameters and locations on the server, as can be seen in Fig. 6.

The first task to start *Test Environment Creation* is performed by the Administrator. He must introduce in the Quality tool the name, operating system and software installed on the physical computers equipped for test execution, as well as the Test Managers who can operate with them. The Expert records information about operating systems, software and virtual machine templates available for create test environments. The Test Manager stores the data of its virtual machines and distributes them to the real machines assigned to it by the Administrator. Fig. 7 captures the data from a real machine and the list of its virtual machines.

During the last stage, test components are organized in test suits to be executed in a particular order. The Test Manager indicates for each script, the test machine (virtual or real) that will execute it. Next, the system makes a copy of the component from the server to the test machine. Components within a test suite may be performed sequentially or simultaneously in one or more test machines. The suite of tests is called Quality Control Process (QCP), and its configuration can be seen in Fig. 8.

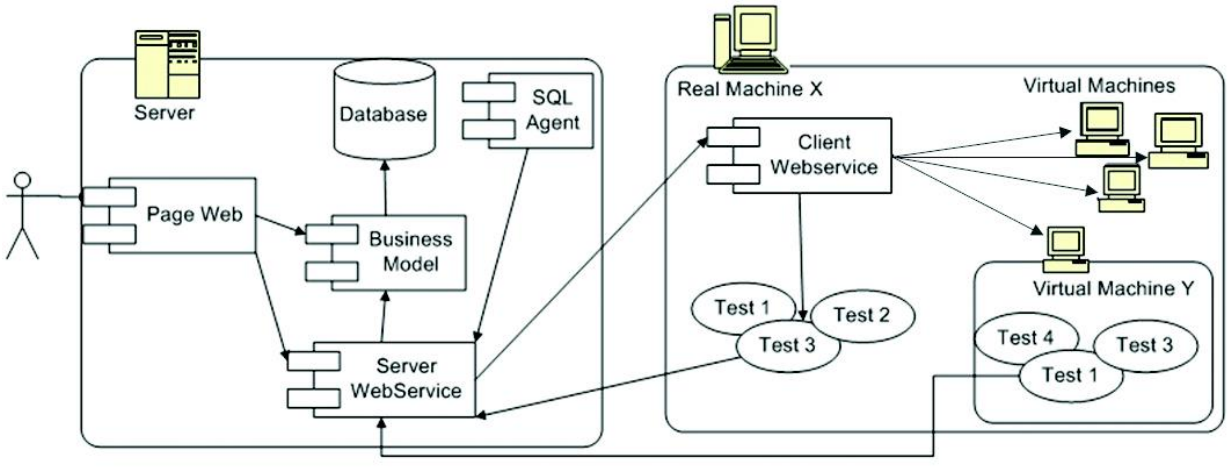


Fig. 5. Internal structure of the Quality Tool.

The input parameters value of the test components can vary for the same test suite, as well as the execution schedules. The definition of these terms is called Instance of the Quality Control Process. This concept allows that one defined test suite can be executed at different times and under different conditions determined by the values of the parameters and the configured test environment. At this point, the required data for a test suite execution is recorded. The Instance of the Quality Control Process can be executed from the web interface by user demand, or on a scheduled date and time.

The Auditor can get the results and events related to every execution of test suite from the Quality web interface. Additionally, during the configuration of the Quality Control Process, Test Managers can identify email addresses to send reports about test execution. These reports can be sent at beginning and/or ending of the execution of a test suite and/or a single test script; likewise if a system error is detected.

IV. Application in Pilot Projects

The designed process was applied to carry out the unattended execution in two pilot projects in a software development company. Those software projects present differences regarding technology, programming language, architecture and team size, demonstrating the applicability of the proposal in various development environments. Results from this experiment demonstrate the improvements that the given solution provides to the test process in the selected areas of the software company.

A. Pilot Project 1

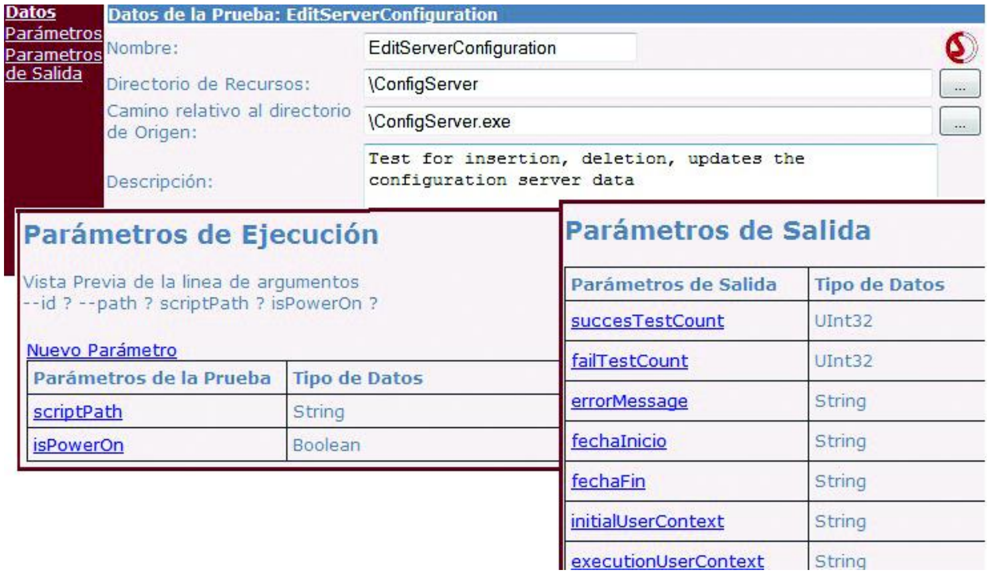
The first pilot project works on a library implemented in C++ native language by a single developer. It is a multiplatform class library designed to extract files from diverse compression formats. Associated with this, there is a test project that contains more than 50 unit tests by format, which were created using the Boost library [24]. These tests validate all the library features. This project was developed and verified prior to the conception of the proposed solution. The developer ran the test project resulting binary in his workstation whenever he wanted to validate it. The library belongs to a system in production at the stations of the company's customers. The process phases performed on the library are explained below.

During the first phase a test script is created using the wrapper provided by the Quality tool. This way, the wrapper execution makes a run on the tests contained in the test project binary and its results are stored in the Quality database. The test input parameter of this component is the relative path where the test project binary is located on the running machine. Later a directory that contains the wrapper and the binary is created on the server. Finally the test component data created is saved in the Quality web interface.

The test environment for the library consists of a real machine. In such a machine Framework .NET 2.0 and IIS 5.1 were installed. Also the tool web service for client machines was published. Subsequently the information related to the test station is stored. The development test environment took a few hours, however this is done only the first time it is introduced into the system.

Through the Quality webpage a test suite composed by the compiled test component is designed. The prepared real machine is selected, indicating the location within the machine where the component will be run. Additionally, the input parameter value representing the relative path of the test project binary is set. The directory containing the test component must be copied from the server to the client machine before execution. The test will be executed with user permission system. Fig. 9 displays the configuration of the test instance for this test component.

The test suite execution takes place every month. For this, one of the schedules configured in the system was selected. The following image shows the description of the selected schedule. After each run, the results are mailed to the library’s developer.



Datos de la Prueba: EditServerConfiguration

Nombre:	EditServerConfiguration
Directorio de Recursos:	\ConfigServer
Camino relativo al directorio de Origen:	\ConfigServer.exe
Descripción:	Test for insertion, deletion, updates the configuration server data

Parámetros de Ejecución

Vista Previa de la linea de argumentos
--id ? --path ? scriptPath ? isPowerOn ?

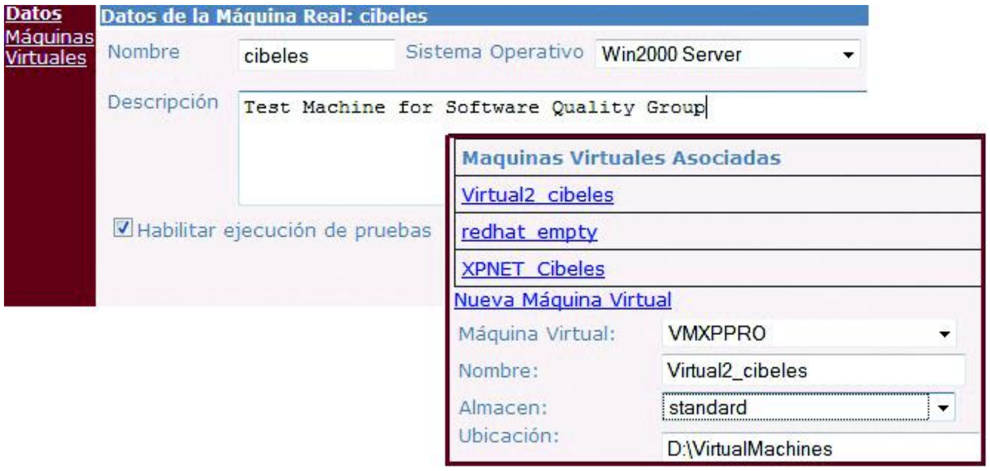
Nuevo Parámetro

Parámetros de la Prueba	Tipo de Datos
scriptPath	String
isPowerOn	Boolean

Parámetros de Salida

Parámetros de Salida	Tipo de Datos
succesTestCount	UInt32
failTestCount	UInt32
errorMessage	String
fechaInicio	String
fechaFin	String
initialUserContext	String
executionUserContext	String

Fig. 6. Test Edition Page of QUALITY tool.



Datos de la Máquina Real: cibeles

Nombre	cibeles
Sistema Operativo	Win2000 Server
Descripción	Test Machine for Software Quality Group

☑ Habilitar ejecución de pruebas

Maquinas Virtuales Asociadas
Virtual2_cibeles
redhat_empty
XPNET_Cibeles

Nueva Máquina Virtual

Máquina Virtual:	VMXPPRO
Nombre:	Virtual2_cibeles
Almacen:	standard
Ubicación:	D:\VirtualMachines

Fig.7. Real and Virtual Machine Association Page of QUALITY tool.

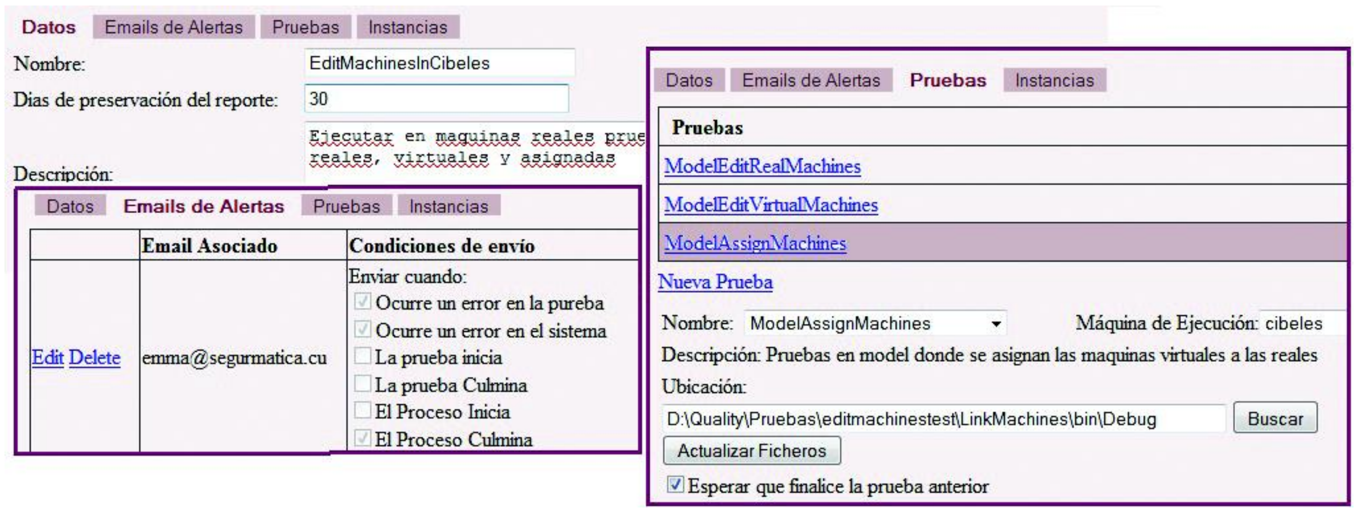
Results obtained in the experiment

The tests run to validate the extraction of the different formats of the library took 49 seconds. In the first test suite execution, 42 errors were detected, especially related to the extraction of files whose formats are less common in the client machines. As the errors detected were solved by the developer, the tool allows to record and report the system progress to other specialists monthly. The recurring execution prevents the introduction of new defects caused by the implementation of additions or modifications, and if this were to happen, the automated process provides a way of finding them in a short time.

B. *Pilot Project 2*

This pilot project concerned a multilayer system implemented on the .NET platform whose development was in progress at the solution application. It is a distributed system whose architecture consists of a web interface, two web services, and three class libraries. The development team includes internal, temporary internal, and external company developers. As features were added, the developers implemented the related unit tests, which were executed on the workstations. Next, the process applied to this project with the described conditions will be exposed.

In the course of Test Component Generation stage, the Visual Studio Team Suite development tool was used for create scripts. Those test components directly reference the libraries provided by the Quality tool to communicate with the system. Table 1 shows the test types enclosed on the test components created. One component perform one or more validation or verification actions to the system under test, e.g. the Unit Tests script, run all unit tests of a particular system module. For each test, component folders with its files were created on the server. Through the Quality web interface, the information related to these components was stored.



Datos | Emails de Alertas | Pruebas | Instancias

Nombre: EditMachinesInCibeles

Dias de preservación del reporte: 30

Descripción: Ejecutar en maquinas reales prue reales, virtuales y asignadas

Datos | Emails de Alertas | Pruebas | Instancias

	Email Asociado	Condiciones de envío
Edit Delete	emma@segurmatica.cu	Enviar cuando: ✓ Ocurre un error en la pureba ✓ Ocurre un error en el sistema ☐ La prueba inicia ☐ La prueba Culmina ☐ El Proceso Inicia ✓ El Proceso Culmina

Datos | Emails de Alertas | Pruebas | Instancias

Pruebas
ModelEditRealMachines
ModelEditVirtualMachines
ModelAssignMachines

Nueva Prueba

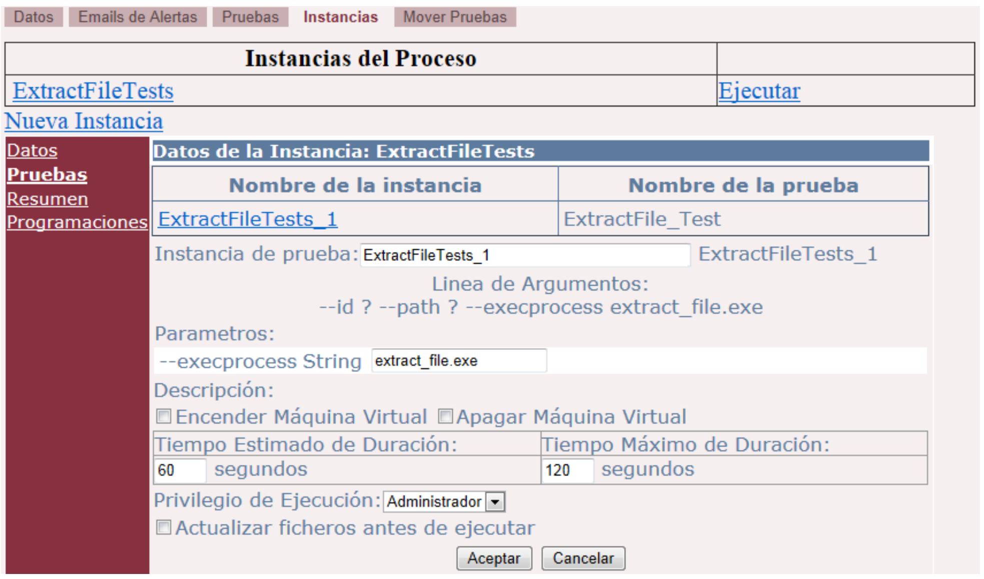
Nombre: ModelAssignMachines Máquina de Ejecución: cibeles

Descripción: Pruebas en model donde se asignan las maquinas virtuales a las reales

Ubicación: D:\Quality\Pruebas\editmachinestest\LinkMachines\bin\Debug

✓ Esperar que finalice la prueba anterior

Fig. 8. Quality Control Process Editing Page of QUALITY tool.



Datos | Emails de Alertas | Pruebas | Instancias | Mover Pruebas

Instancias del Proceso	
ExtractFileTests	Ejecutar

Nueva Instancia

Datos de la Instancia: ExtractFileTests

Nombre de la instancia	Nombre de la prueba
ExtractFileTests_1	ExtractFile_Test

Instancia de prueba: ExtractFileTests_1 ExtractFileTests_1

Linea de Argumentos:
--id ? --path ? --execprocess extract_file.exe

Parametros:
--execprocess String extract_file.exe

Descripción:

☐ Encender Máquina Virtual ☐ Apagar Máquina Virtual

Tiempo Estimado de Duración:	Tiempo Máximo de Duración:
60 segundos	120 segundos

Privilegio de Ejecución: Administrador

☐ Actualizar ficheros antes de ejecutar

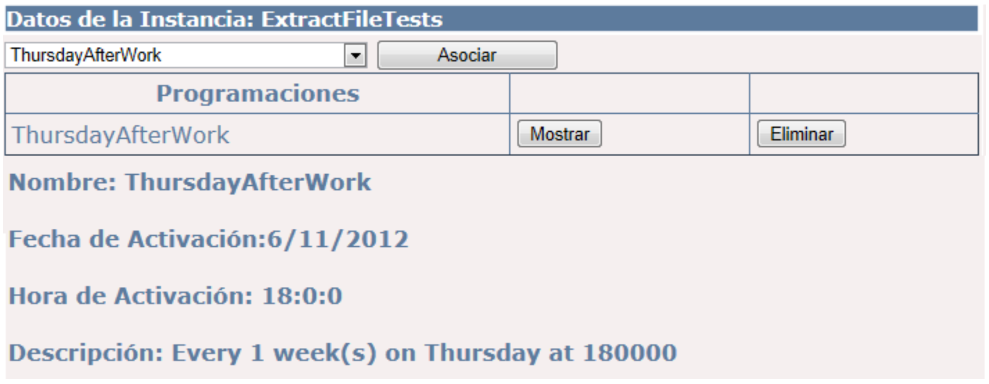
Fig. 9. Test Instance Edition Page of QUALITY tool.

The test environment was equipped for two physical machines and four virtual machines. The Fig. 11 shows the test lab established. Real Machine # 1 is intended only to hold three virtual machines, in which test components executions occur. Another virtual machine, fulfill the Server system role, so the tests run on this machine focus on server functions. On this machine the system under test web service for server was installed. Real Machine # 2 plays a dual role, as well as housing the Virtual Machine # 3, run tests on it directly. Because Virtual Machines # 1, 2, 3 and the Real #2 are client machines; the client web service of the system under test was installed on them. The Server and Client machines communicate each other across the local network. Real Machine # 2 coincides with the one built in Pilot Project 1. Therefore, it was not necessary to perform the installation process to incorporate it into the system.

TABLE I.
TEST TYPES DISTRIBUTION BY THE TEST COMPONENTS.

Test Types	Test Component Count
Unit Tests	1
Database Consistency Checking	1
Project Build	1
Functional Tests	20
Integration Tests	5
Web resources availability Checking	1

During the third phase, test scripts are grouped according system functionalities to be verified. Table 2 summarizes the configured tests suite or Quality Control Processes (QCP), the component test types involved and the executions machines. The size of a QCP is expressed as *a*/*b*, where *a* indicates the


Datos de la Instancia: ExtractFileTests

ThursdayAfterWork | Asociar

Programaciones		
ThursdayAfterWork	Mostrar	Eliminar

Nombre: ThursdayAfterWork

Fecha de Activación:6/11/2012

Hora de Activación: 18:0:0

Descripción: Every 1 week(s) on Thursday at 180000

Fig. 10. Selecting a schedule for a test suite execution by Quality tool.

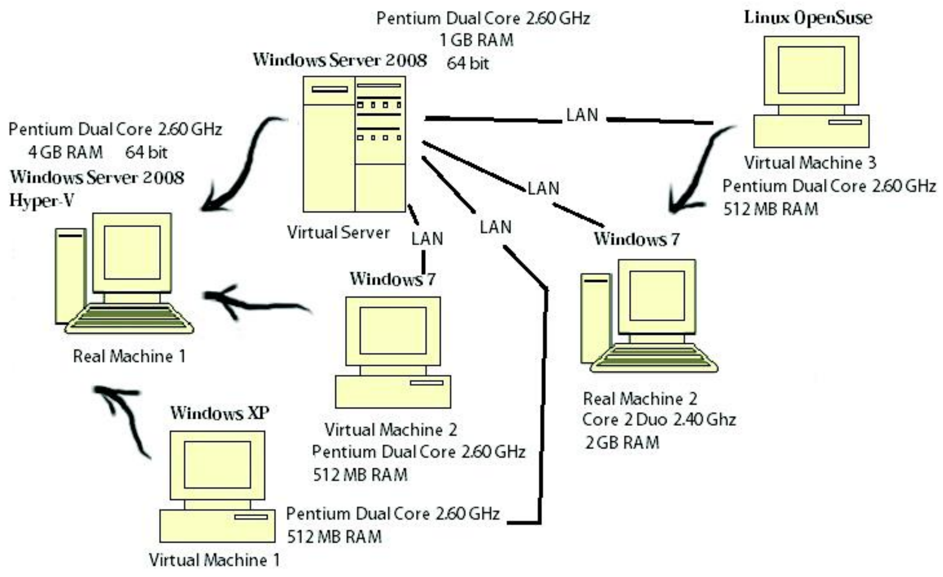


Fig. 11. Test environment for the multilayer system under test.

different test scripts implicated count and *b* represents the number of times the test components are called. For example, the QCP designed to run unit tests consists of three different components: one for Unit Tests, others for Database Consistency Checking and Project Building. Because the system under test consists of 5 core modules, the QCP calls the test scripts 15 times. Another example is the test suite to check the client web service. In this case, there is a test component and it is called four times, for each client machine in the test lab.

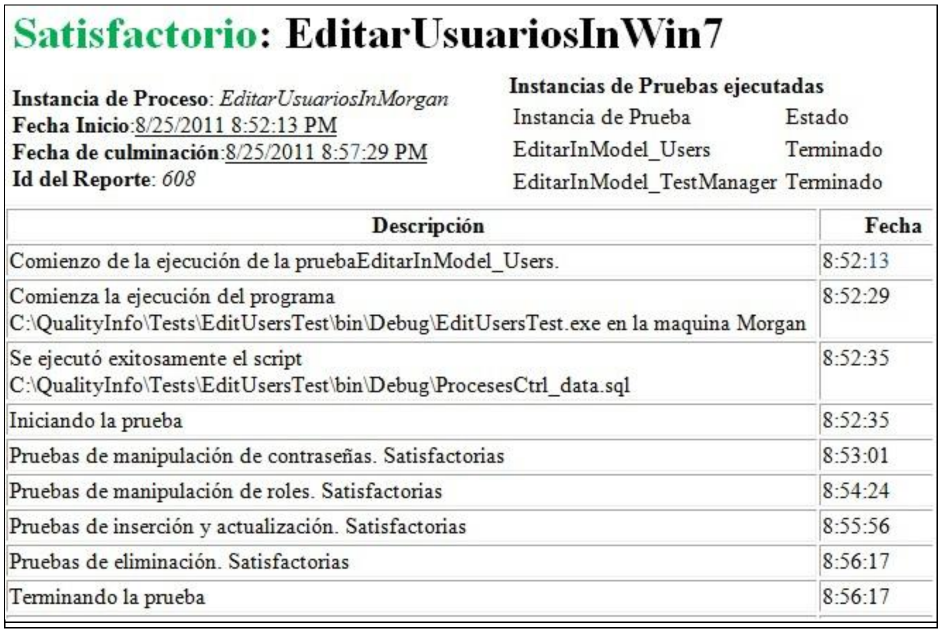
Note that the test scripts are reused in the defined QCPs. The same test components have been incorporated into different QCPs to verify diverse aspects of the system under test. This fact is evidenced in the QCPs to verify the on / off virtual machines and to validate test execution in virtual machines. The two test scripts present in the first process are also included in the second, before and after the test component responsible for running the test on the virtual machine.

It was decided to run the test components twice a week, to perform regression testing from changes made during two or three days. Once the configurations for executions were stored on Quality tool, it was possible to update and redistribute the system modules and test components among the test environment machines. After each QCP execution, the results are mailed to the specialists implicated. The following figure shows a fragment of an Instance of the QCP instance run result. The events generated by the test script can be seen.

Results obtained in the experiment

The registered time of execution of all test suites was approximately three hours. It has been estimated that the execution time of all test scripts performed manually takes 7 and a half hours. The possibility to perform runs outside office hours and the presence of an isolated test lab from developer machines saves development and test time for the work team.

The company selected to do the pilot projects, have followed the Scrum Agile methodology. A monthly record or backlog of the development and test work, as well as the defects detected, has been kept. Tasks are planned to be completed in a similar time span, yielding a deliverable



Satisfactorio: EditarUsuariosInWin7

Instancia de Proceso: *EditarUsuariosInMorgan*
Fecha Inicio:8/25/2011 8:52:13 PM
Fecha de culminación:8/25/2011 8:57:29 PM
Id del Reporte: *608*

Instancias de Pruebas ejecutadas

Instancia de Prueba	Estado
EditarInModel_Users	Terminado
EditarInModel_TestManager	Terminado

Descripción	Fecha
Comienzo de la ejecución de la pruebaEditarInModel_Users.	8:52:13
Comienza la ejecución del programa C:\QualityInfo\Tests\EditUsersTest\bin\Debug\EditUsersTest.exe en la maquina Morgan	8:52:29
Se ejecutó exitosamente el script C:\QualityInfo\Tests\EditUsersTest\bin\Debug\ProcesesCtrl_data.sql	8:52:35
Iniciando la prueba	8:52:35
Pruebas de manipulación de contraseñas. Satisfactorias	8:53:01
Pruebas de manipulación de roles. Satisfactorias	8:54:24
Pruebas de inserción y actualización. Satisfactorias	8:55:56
Pruebas de eliminación. Satisfactorias	8:56:17
Terminando la prueba	8:56:17

Fig. 12. Quality report at QCP Instance run finished.

artifact. To complete a functionality it is necessary to perform two or more tasks, depending on its complexity. Tasks are classified according to the stage where they were planned: tasks planned for each sprint during the pregame to develop the system (base tasks) and tasks arising from any errors detected in a previous cycle (defect tasks).

Fig. 13 illustrates the behavior of the tasks performed during one year since March 2010. The graph contains three series, the number of base tasks; the number of defect tasks and the total resulting from the sum of the number of tasks of both previous classifications. The proposed solution was applied in the month of June. In the figure we can see how in the months of June, July and August the number of base tasks increased because specialists created the artifacts required in the defined process. However, in the months of September, October and November there was only a slight increase in the tasks generated by the defects found after the executions of test suites in the previous months.

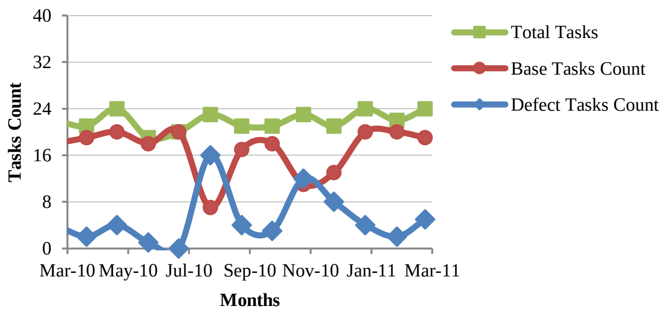


Fig. 13. Graph of the tasks performed before and during insertion of the defined process. Defect tasks are represented by diamonds; base tasks, by circles and the total by squares.

Fig.14 shows the behavior of the work done to develop the pilot project 6 months after that the proposed process was introduced with Quality tool for automation. The graph shows defect tasks have decreased heavily, because the tool helps early detection of errors introduced during implementation. Consequently nonconformities can be discovered and resolved in the same sprint in which they are introduced. From February, the system under test was in the closing stage, for this reason the base tasks also decreased, in turn minimizing the total number of tasks to be performed.

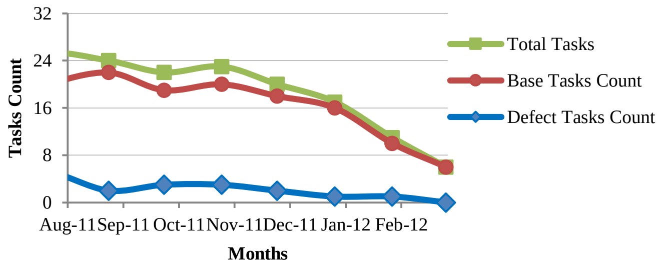


Fig. 14. Graph of the tasks performed 6 months after inserting the proposed process. Defect tasks are represented by diamonds; base tasks, by circles and the total by squares.

Another improvement provided by the proposed solution to the software development process can be seen in the fault detection. The number of errors found was obtained from the backlogs in each sprint. These defects were grouped into two categories: defects detected in functionalities planned in the current sprint (new defects) and failures identified in the current cycle that correspond to functionalities that are considered done in previous cycles (old defects). Fig. 15 describes the conduct of this variable at the same interval of

TABLE II
DEFINED QCPS FOR THE MULTILAYER SYSTEM

Quality Control Process	Size	Test Types Components	Execution Machine
Unit Testing	3/15	Unit Tests	Virtual Machine 1, 2
		Database Consistency Checking	
		Project Build	
Client Web Service Checking	1/4	Web Resources Availability Checking	Virtual Machines 1, 2, 3, Real Machine 2
Virtual Machines Replication	1/1	Integration Tests	Real Machine 2
Machines Edition	3/3	Functional Tests	Virtual Machine 1
Relocate Tests	2/2	Functional Tests	Virtual Machine 1, 2
		Integration Tests	
Timeout expires	1/1	Functional Tests	Virtual Machine 3
Test Edition	3/3	Functional Tests	Machine Real 2
Test Parameters Edition	2/2	Functional Tests	Virtual Machine 2
Process Edition	5/5	Functional Tests	Virtual Machine 1, 2, Real Machine 2
Report Generation	4/4	Functional Tests	Virtual Machine 1
On / off Virtual Machines	2/2	Integration Tests	Real Machine 2
Test run on Virtual Machines	3/3	Integration Tests	Virtual Machine 1
Other operations on virtual machines	1/1	Integration Tests	Real Machine 2
Access Permissions	1/1	Functional Tests	Virtual Machine 1, Real Machine 2
Users Management	2/2	Functional Tests	Virtual Machine 1

Fig. 13 which represents the period before and during implantation of the proposed solution.

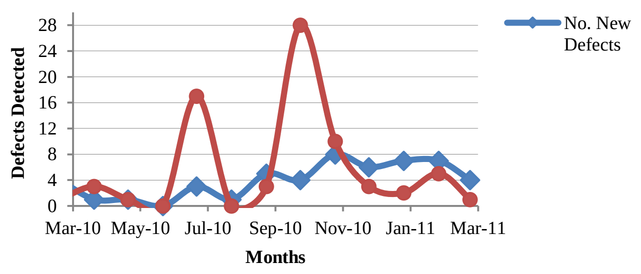


Fig. 15. Graph with the monthly number of defects detected before and during insertion of the proposed process. New defects are represented by diamonds and old defects, by circles.

Above it can be seen that in July, a month after start the solution implantation the system detected an increasing number of defects. However in August few new flaws were found, because the specialists were involved in resolving problems encountered in July, as is outlined in the Fig. 13. In September continues detecting errors due to the creation of new test components. The faults exposed at this stage should have been detected months before the application process. In Fig. 16 can be observed the lines of nonconformities found 6 months after the introduction of the proposed process. At this stage we can see that the number of defects found in features implemented in the current cycle is greater than the number of faults discovered in functionality delivered at earlier sprints.

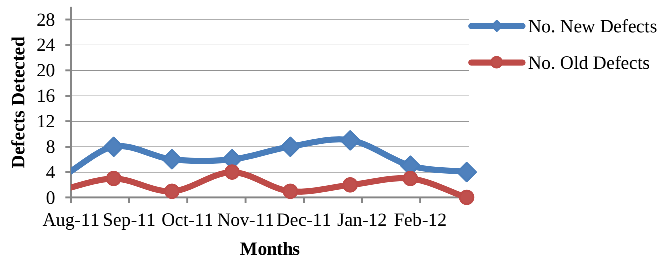


Fig. 16. Graph with monthly the number of defects detected after inserting the proposed process. New defects are represented by diamonds and old defects, by circles.

The most frequent found errors are: Access denied to data; unmanaged exceptions and absent of errors logs, problems with web resources availability, multithreads synchronization issues and timeout expiration. As the errors were detected and solved by developers, unattended executing allows the store and report to stakeholders of the run test results twice a week.

V. CONCLUSION

This paper has detailed a process to standardize the unattended test execution in organizations that develop software product lines. Three stages are defined: Test

Components Generation, Test Environments Creation and Test Execution and Collecting Results. The processes comprise the registration and control of test environments, including machine virtualization. A tool to support the process described has been implemented. This application facilitates the artifacts generation and allows the unattended execution of test components.

The proposed solution adopts the reusability approach proclaimed by the engineering of software product lines. It also promotes the standardization for the test execution of the variations. The process application has reduced the development and testing time, also has provided improvements to the detection of defects in a software company.

ACKNOWLEDGMENTS

This work was supported in part by the Software Security Enterprise, Segurmatica, and the Informatics Studies and Systems Center at the Polytechnic Institute "José Antonio Echevarría." The authors want to thank all workers of the company who participated in the implementation of the proposed solution. We also are very grateful to Heydi Mendez and Annette Morales for their help with translating this paper.

REFERENCES

[1] R. S. Pressman and J. E. Murrieta, *Ingeniería del software, un enfoque práctico*, 6th ed., Mexico, McGraw-Hil Interamericana, 2006, ch.13, pp. 383–414

[2] J. Barnes, *Implementing the IBM® Rational Unified Process® and Solutions: A Guide to Improving Your Software Development Capability and Maturity*. Mexico City, IBM Press, 2007

[3] *Software engineering — Product quality — Part 1: Quality model*, ISO/IEC 9126-1, 2001

[4] R. Pinheiro, K. M. Oliveira, and W. Pereira. "Evaluating the service quality of software providers appraised in CMM / CMMI, *Software Quality Journal*, vol. 17, no. 3, 2009, pp. 283–301; http://link.springer.com/article/10.1007%2Fs11219-008-9065-4

[5] P. Abrahamsson, N. Oza, and M. T. Siponen, "Agile Software Development Methods: A Comparative Review," in *Agile Software Development Current Research and Future Directions*, T. Dingsøyr, T. Dybå and N. Brede, (eds.), Springer, 2010, pp. 31–53

[6] E. Bagheri, F. Ensan, and D. Gasevic, "Decision support for the software product line domain engineering lifecycle," *Automated Software Engineering*, vol. 19, no. 3, 2012 pp. 335–377; link.springer.com/article/10.1007/s10515-011-0099-7

[7] G. K. Hanssen, "Opening Up Software Product Line Engineering," PLEASE'2010 International Workshop, 2010; http://www.idi.ntnu.no/grupper/su/publ/geirkjetil/hanssen-open prodline-please10.pdf

[8] P. A. da Mota Silveira, P. Runeson, I. do C. Machado, E. Santana, S.R. de Lemos, and E. Engstrom, "Testing Software Product Lines," IEEE, vol. 28, no. 5, 2011, pp. 16–20; http://www.computer.org/csdl/mags/so/2011/05/mso2011050016-abs.html

[9] J. Dehlinger and R. R. Lutz, "PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool," *Automated Software Engineering*, vol. 13, no. 1, 2006, pp. 169–193; http://www.cs.iastate.edu/~dehlinge/papers/dehlinger_lutz_AUSE_2006.pdf

[10] A. Bertolino and S. Gnesi, "PLUTO: A Test Methodology for Product Families," *Lecture Notes in Computer Science*, vol. 3014, 2004, pp. 181–197; www.inf.ufpr.br/silvia/topicos/artigostrab10/Bertolino.pdf

[11] E. Uzuncaova, D. Garcia, S. Khurshid, and D. Batory, "Testing Software Product Lines Using Incremental Test Generation," in *Proc. 19th ISSRE*, Washington, DC, 2008, pp. 249–258

[12] A. Edwards, S. Tucker, and B. Demsky, "AFID: an automated approach to collecting software," *Automated Software Engineering*, vol. 17, no. 3, 2010, pp. 347–372. http://link.springer.com/article/10.1007%2Fs10515-010-0068-6#

[13] M. S. Feather and B. Smith, "Automatic Generation of Test Oracles—From Pilot Studies to Application," *Automated Software Engineering*, vol. 8 no. 1, 2001, pp. 31–61. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.7101&rep=rep1&type=pdf

[14] C. Schwarzl and B. Peischl. "Generation of executable test cases based on behavioral UML system models," in *Proc. 5th Workshop on AST '10*, New York, NY, 2010, pp. 31–34

[15] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for GUI-based software applications," *ACM TOSEM*, vol, 16, no. 1, 2007, pp. 4. http://dl.acm.org/citation.cfm?id=1189752

[16] F. Bouquet, C. Grandpierre, B.Legeard, and F. Peureux, "A Test Generation Solution to Automate Software Testing," in Proc. 3rd International Workshop on AST '08, New York, NY, 2008, pp. 45–48

[17] C. Davis, D. Chirillo, D. Gouveia, F. Saracevic, J. B. Bocarsley, L. Quesada, L. B. Thomas, and M. van Lint, *Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource*, 1st ed. Upper Saddle River, N.J: IBM Press, 2009

[18] J. Levinson, *Software Testing with Visual Studio® 2010*, 1st ed. Redwood City, CA: Addison-Wesley Professional, 2011

[19] L. Chang. "Platform-Independent and Tool-Neutral Test Descriptions for Automated Software Testing," in Proc. ICSE, New York, NY, 2000, pp. 713–715

[20] S. D. Burd, G. Gaillard, E. Rooney, and A. F. Seazzu, "Virtual Computing Laboratories Using VMware Lab Manager," in *Proc. 44th HICSS*, Washington, DC, 2011, pp. 1–9.

[21] J. N. Matthews, T. Deshane, W. Hu, E. M. Dow, J. Bongio, P. F. Wilbur, and B. Johnson. *Running Xen: A Hands-On Guide to the Art of Virtualization*, 1st ed. Upper Saddle River, NJ: Prentice Hall, 2008

[22] N. Rice and S.Trefethen, *TestComplete Version 8 Made Easier: Keyword Testing*, Falafel Software Inc., 2012

[23] R. Walters, G. Fritchey, and C. Taglienti. "Common Database Maintance Tasks," in: *Beginning SQL Server 2008 Administration*, New York, NY: Apress L.P., 2009, pp. 225–233

[24] M. Reddy. "Testing," in *API Design for C++*. Burlington, MA: Morgan Kaufmann (Ed), 2011, ch. 10. pp. 218–328