*Serie Científica de la Universidad de las Ciencias Informáticas*
Vol. 9, No. 4, Abril, 2016
ISSN: 2306-2495 | RNPS: 2343
http://publicaciones.uci.cu
Pág. 1-14

Tipo de artículo: Reporte de Caso.
Temática: Calidad de Software.
Recibido: 25/01/2016 | Aceptado: 01/04/2016

# Personalización del proceso de pruebas unitarias empleando la herramienta NUnit

# *Process customizing using the NUnit unit testing tool*

**Susana González Espinosa [1*], Rafael Bello Lara [2]**
1* Facultad 3. Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, km 2½, Torrens, Boyeros, La Habana, Cuba. CP.: 19370.
2 Facultad 3. Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, km 2½, Torrens, Boyeros, La Habana, Cuba. CP.: 19370.
* sespinosa@uci.cu

---

**Resumen:**

El presente trabajo propone una estrategia para realizar pruebas unitarias de forma automatizada, con el objetivo de facilitar la labor de los desarrolladores durante el proceso de construcción de un sistema informático. Se describen brevemente aspectos relacionados con el tema y las herramientas que pueden ser utilizadas. Además, se indica la importancia y trascendencia de las pruebas unitarias, detallando los procesos que deben seguirse en la etapa de pruebas, los roles que intervienen, las actividades que se desarrollan, así como los artefactos de entrada y salida de cada uno de los procesos. De igual forma se expone una panorámica relacionada con los distintos *frameworks* de pruebas automatizadas. Por último, se aplican los procesos descritos para la ejecución de las pruebas unitarias utilizando NUnit en las aplicaciones producidas por el equipo de desarrollo de Registros y Notarías, quedando estas como ejemplo de la utilización de esta nueva estrategia y como validación de la misma.
**Palabras clave:** herramientas de pruebas automatizadas, procesos de pruebas, pruebas de software, pruebas unitarias.

***Abstract:***

*This research proposes a strategy for automated unit testing, in order to facilitate the work of developers during the construction process of a computer system. Aspects of the subject and the tools that can be used are briefly described. It also indicates the importance and significance of unit testing, detailing the processes to be followed in the testing*

*stage, the roles involved, the activities performed and the input and output artifacts of each of the processes. Similarly, an overview regarding the various automated test frameworks is exposed. Finally, apply the processes described to implement unit testing using NUnit in applications produced by the development team of Registries and Notaries are applied, leaving these tests as an example of the use of this new strategy and as validation of the same.*
***Keywords:*** *automated testing tools, software testing, software unit testing; testing processes.*

## 1. Introducción

Las pruebas de software han evolucionado, no sólo permiten depurar el producto, sino que se han convertido en un proceso integrador, automatizable y administrable que se realiza desde el inicio del desarrollo del software y se extiende hasta la entrega final al cliente. En Cuba, desde hace algunos años, se ha comenzado la profundización de pruebas al software para que su funcionamiento sea el esperado por el cliente durante su explotación. La Universidad de las Ciencias Informáticas (UCI) es una de las instituciones proveedoras de software en el país que cuenta con una infraestructura formada por Centros de desarrollo donde se ejecutan proyectos de informatización. En la mayoría de los proyectos que se desarrollan en estos centros no se comprueba si el código fuente del producto posee errores lógicos y suposiciones incorrectas a través de la aplicación de pruebas unitarias, surgiendo el riesgo de alcanzar la fase de integración con un alto grado de incertidumbre sobre la funcionalidad del código como demuestran los resultados de las pruebas de liberación de varios productos de la universidad (CALISOFT, 2012). El costo en tiempo y esfuerzo necesario para reducir la incertidumbre sobre la base de las pruebas manuales es muy elevado para los tipos de desarrollos más comunes. Esto implica la necesidad del uso de herramientas que automaticen el proceso. Sin embargo, los mecanismos y procedimientos necesarios para aplicar pruebas de integración de manera automatizada no son dominados por la mayoría de los desarrolladores.

Cuando se realizan las pruebas de caja negra (ORÉ, 2014) al final del desarrollo y es detectado un fallo, se puede demorar demasiado encontrar dónde se cometió el error, aunque, si se realizan las pruebas de caja blanca (TUYA, et al., 2007) antes de integrar el sistema se pueden detectar fallos en el código y solucionarlos desde la etapa inicial del desarrollo del software. Por lo que se propone que, para llevar a cabo la estrategia con éxito, se debe comenzar a trabajar desde el interior del programa, es decir, desde la partícula más pequeña hasta la máxima expresión que el mismo pueda alcanzar; por lo que se decide comenzar este proceso realizando la prueba de unidad.

La automatización de las pruebas trae numerosas ventajas, pues es una garantía para la calidad del proceso a desarrollar. Es por esto que se recomienda llevar a cabo dicho proceso siempre que sea posible. Las pruebas automatizadas se hacen repetibles, optimizan en gran medida el tiempo, permiten probar cambios en el código y se aseguran de que en ellos no se hayan introducido errores funcionales; simplifican la integración, ya que permiten llegar a esta etapa con un alto grado de seguridad sobre el código; documentan el código; separan la interfaz y la implementación; los defectos están acotados y son fáciles de localizar (DUSTIN, et al., 2000).

El presente trabajo propone una estrategia para realizar pruebas unitarias de forma automatizada, con el objetivo de facilitar la labor de los desarrolladores durante el proceso de construcción de un sistema informático. Se describen brevemente aspectos relacionados con el tema y las herramientas que pueden ser utilizadas. Además, se indica la importancia y trascendencia de las pruebas unitarias, detallando los procesos que deben seguirse en la etapa de pruebas, los roles que intervienen, las actividades que se desarrollan, así como los artefactos de entrada y salida de cada uno de los procesos. De igual forma se expone una panorámica relacionada con los distintos *frameworks* de pruebas automatizadas. Por último, se aplican los procesos descritos para la ejecución de las pruebas unitarias utilizando NUnit en las aplicaciones producidas por el equipo de desarrollo de Registros y Notarías, quedando estas como ejemplo de la utilización de esta nueva estrategia y como validación de la misma.

## 2. Pruebas automatizadas de software

Gerard Meszaros plantea que la razón tradicional por la cual hacer las pruebas es la de asegurar la calidad, este término se conoce como QA, por sus siglas en inglés *Quality Assurance* (MESZAROS, 2007).

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. Es responsabilidad del equipo de desarrollo probar los módulos o subsistemas, asegurándose que cada uno lleve a cabo la funcionalidad para el que fue diseñado.

La metodología robusta RUP (*Rational Unified Process*, por sus siglas en inglés) identifica entre sus etapas la de pruebas, cuyo propósito general es garantizar que el programa no contiene errores de diseño o codificación. En esta etapa no se desea saber si el programa realiza lo que solicitó el usuario debido a que dicha tarea le corresponde a la etapa de implementación, sino que se desea encontrar la mayor cantidad de errores. Todos los programas contienen errores: encontrarlos es cuestión de tiempo. Lo ideal es encontrar la mayoría, si no todos, en esta etapa (CANTONE, 2008).

**Las pruebas deben ayudar a mejorar la calidad:** uno de los principales propósitos de las pruebas es detectar errores en los sistemas; sin embargo, también puede pensarse que ayudan a evitar errores, ejecutándolas de una forma correcta con el personal indicado y siguiendo un buen procedimiento. No se tendrán errores si primero se ejecutan

pruebas de regresión (TUYA, et al., 2007), porque estas señalarán el punto donde se debe verificar el error introducido en el código.

**Las pruebas deben localizar de manera más fácil los errores:** en ocasiones es más costoso prevenir errores que arreglarlos. La existencia de estos puede ser inevitable. Si se hacen muchas pruebas unitarias medianamente pequeñas que verifiquen un simple comportamiento cada una, se podría encontrar el error rápidamente, conociendo cuál es la prueba que falla. Esta es una gran ventaja sobre las pruebas de aceptación (TUYA, et al., 2007), donde solo se indicaría que algún comportamiento esperado no está funcionando correctamente. Las pruebas unitarias por su parte indicarían claramente dónde se está cometiendo el error.

**Las pruebas deben ayudar a entender el Sistema Bajo Prueba (SBP):** no es necesario introducirse en el código del SBP para responder cuál debe ser el resultado para una entrada determinada. Las pruebas, por sí solas, dicen cuál debe ser el resultado esperado. Las pruebas auto-verificadoras indican la salida esperada a través de una o más aserciones (RIVERO, 2008).

**Las pruebas deben ser fáciles de ejecutar:** las pruebas automatizadas permiten que el trabajo sea más rápido, deben ser ejecutadas sin intervención manual, deben contener en su código cada aspecto que necesita ser verificado, deben poder ser ejecutadas varias veces, obteniendo siempre el mismo resultado. Es ideal que las pruebas puedan ser independientes y que puedan ser ejecutadas por sí mismas, así se convertirían en pruebas repetibles sin esfuerzo adicional (RIVERO, 2008).

**Las pruebas deben ser fáciles de escribir y mantener:** las pruebas simples pueden lograrse manteniendo pequeñas pruebas que verifiquen solo una condición a la vez. Cuando se escribe código con el fin de pasar una prueba a la vez, se puede ir agregando porciones pequeñas de comportamiento nuevo al SBP después de pasar cada prueba. La prueba debe ser expresiva, para disminuir la brecha de expresividad puede ser enfocada en crear una librería de métodos útiles para las pruebas que constituyan un lenguaje de dominio específico. El probador podrá expresar los conceptos que desea sin necesidad de traducir sus pensamientos en códigos muy detallados. De esta manera se evita también la duplicación de código en las pruebas, lo cual es un principio no solo aplicable al código de producción (RIVERO, 2008).

**Las pruebas deben requerir el mínimo mantenimiento mientras evoluciona el sistema:** inevitablemente, a medida que evoluciona el sistema pueden existir muchos cambios que afecten las pruebas. Se debe tratar de que el número de pruebas afectadas sea mínimo. Una de estas maneras es evitando el solapamiento de las pruebas. Otra razón puede ser el cambio de ambiente en las pruebas. En lo posible debe probarse el sistema de manera aislada.

Verificar una condición a la vez y utilizar métodos útiles para las pruebas puede ayudar a reducir en mucho el número de pruebas que serán afectadas directamente por algún cambio (MESZAROS, 2007).

Gerard Meszaros, a criterio personal, propone una manera de trabajo: escribir las pruebas antes que el código en producción; una por una de manera usual, aunque podría crearse una lista de pruebas como especie de esqueleto para comenzar; desarrollar de afuera hacia adentro para determinar lo que se necesita de la capa inmediata inferior; verificar el estado y si además, se necesita tener en cuenta la cobertura del código en las pruebas entonces se debe verificar el comportamiento; y realizar diseño de las configuraciones de las pruebas, una por una (MESZAROS, 2007).

**Beneficios de realizar pruebas unitarias.**

Las pruebas unitarias permiten validar una porción del código del sistema que funciona apropiadamente de manera aislada. En caso de la programación de procedimientos se entiende como programa individual, una función o procedimiento. Cuando se está programando orientado a objetos la menor de estas unidades es siempre una clase.

Rob Johnson define las pruebas unitarias como el nivel más fino de granularidad en las pruebas, que verifican una simple unidad de funcionalidad y deben probar que cada método de una clase satisface su contrato documentado (JOHNSON, 2003).

Para llevar a cabo esta prueba se emplea fundamentalmente la técnica de Caja Blanca que es la que verifica que el código funcione correctamente, sin omitir la técnica de Caja Negra que es la encargada de comprobar el funcionamiento de la unidad externamente. Se prueban todos los caminos posibles con el objetivo de descubrir errores en la pequeña unidad que se está probando.

Estas proveen la capacidad para probar un módulo del software de forma reproducible, eficiente y automatizada. Generalmente, se organizan en suites, a manera de una colección de pruebas unitarias que tienen algo en común. El agrupamiento de las pruebas en suites puede estar dado por características o propósitos comunes: ya sea que prueben el mismo módulo, que tengan configuraciones comunes o porque dependan unas de otras (GRAHAM, et al., 1999).

Escribiendo pruebas simultáneamente con código (o incluso antes) no solo es una buena manera de ahorrar tiempo de desarrollo, además es una excelente herramienta para defenderse contra las regresiones; si se realiza al principio del ciclo de desarrollo ayuda también a identificar y resolver problemas de diseño desde el inicio, lo que reduce refactorización.

Los proyectos con múltiples desarrolladores trabajando en paralelo pueden beneficiarse mucho de las pruebas unitarias. Algunos de estos beneficios son (JOSKOWICZ, 2008):

- Ayudan a resaltar los casos donde lo que parecen pequeños cambios producen fallos inesperados.

- Ayudan a especificar cómo una clase se debería comportar.
- Pueden exponer defectos del diseño al inicio del desarrollo.
- Sirven de grandes ejemplos.
- Son un sitio perfecto para los desarrolladores, ya que aprenden a cómo usar el código.

La utilización de la metodología RUP es factible en proyectos de gran tamaño pues tiene un proceso de pruebas bien definido. La ventaja de ser dirigido por casos de uso posibilita el diseño de casos de prueba y la posible automatización de las mismas. Debido a que es iterativo e incremental, las pruebas no se realizan cuando el producto está terminado, sino que este posibilita que en cada iteración pueda ser aplicado al menos un ciclo de pruebas (RUP, 2003). RUP es la metodología más utilizada para el análisis, implementación y documentación de sistemas orientados a objeto y constituye una buena alternativa para grandes proyectos debido a los artefactos que brinda para la documentación.

El proceso de prueba está estrechamente relacionado a la metodología de desarrollo utilizada en el proyecto, de cualquier manera, la estrategia que se presenta a continuación de manera general podría ser adaptada para su utilización bajo cualquier metodología de desarrollo, no obstante en lo adelante se reflejará el comportamiento de la misma siguiendo RUP, ya que esta es la metodología empleada en el proyecto de desarrollo de software Registros y Notarías.

## Propuesta de solución.

Las pruebas están presentes en todo el ciclo de vida de RUP. Durante la fase de Inicio se pueden planificar inicialmente las pruebas a partir de la Especificación de Casos de Uso del Sistema. En la fase de Elaboración, el análisis y el diseño del software posibilitan de forma más específica la determinación de las pruebas a realizar. En esta fase, una vez que se hayan implementado las primeras funcionalidades, se realizan pruebas unitarias y funcionales, aunque el mayor esfuerzo dedicado a las pruebas unitarias es en la fase de Construcción, ya que es donde más código es generado por los desarrolladores de software (JACOBSON, 2003).

Para llevar a cabo el proceso de pruebas unitarias se analizaron los procesos asociados a las pruebas de software propuestos por RUP. De esta forma se propone la realización de 5 procesos como se muestra la Figura 1, los que se consideran necesarios e indispensables para el desarrollo y progreso efectivo de cada una de las acciones a llevar a cabo en la etapa de pruebas.

*Serie Científica de la Universidad de las Ciencias Informáticas*
Vol. 9, No. 4, Abril, 2016
ISSN: 2306-2495 | RNPS: 2343
http://publicaciones.uci.cu
Pág. 1-14


**Figura 1:** Proceso de automatización de las pruebas. (Elaboración propia)

Por tanto, es necesario comprender que para cada nueva integración de funcionalidades que se le realice al sistema en desarrollo, se irán incorporando sucesivamente los nuevos casos de usos, realizando nuevos diseños de casos de pruebas y obteniendo nuevos resultados.

Luego de revisar los diferentes procesos propuestos por RUP que intervienen en la etapa de pruebas, se analizan los artefactos que se deben generar, los roles que deben intervenir en cada uno de los procesos, entradas, actividades y salidas del mismo. Mediante una representación gráfica se explica con más detalle cada proceso (Ver Figuras 2, 3, 4, 5 y 6).

Durante la etapa de pruebas deben intervenir 4 roles fundamentales, los cuales son responsables de diseñar, aplicar y dar seguimiento a las pruebas a realizar (Administrador de pruebas, Analista de pruebas, Diseñador de pruebas y Probador) (KRUCHTEN, 2004). Por las características del proyecto se deciden sumar dos roles que intervendrán igualmente en estos procesos (Responsable de calidad interna del proyecto y Desarrollador de software).

**Responsable de calidad interna:** Es el encargado de planificar y organizar las revisiones, orientando los artefactos a revisar, así como su entrega, también planifica los cursos de capacitación para el equipo y la aceptación del plan de pruebas. Se encarga de conformar el informe único de No Conformidades con todos los defectos encontrados y se lo hace llegar a los responsables de darle solución a los problemas. Este también debe realizar un resumen de los resultados a través de métricas, archivándose los resultados para su posterior uso, finalizando así las revisiones.

**Desarrollador de software:** Participa junto al Analista de Pruebas en el diseño de los casos de prueba para las pruebas de unidad y de integración. También participa en la implementación y ejecución de las pruebas empleando la herramienta NUnit junto al probador y es el encargado de realizar la depuración.

*Serie Científica de la Universidad de las Ciencias Informáticas*
Vol. 9, No. 4, Abril, 2016
ISSN: 2306-2495 | RNPS: 2343
http://publicaciones.uci.cu
Pág. 1-14

Los artefactos son productos tangibles del proyecto, que son producidos, modificados y usados por las actividades. (PRESSMAN, 2005)

Como primer proceso se tiene la planificación de las pruebas la cual consiste en definir los roles y recursos del sistema para un apropiado entorno de pruebas, además de la definición de una estrategia adecuada y la planificación del esfuerzo.

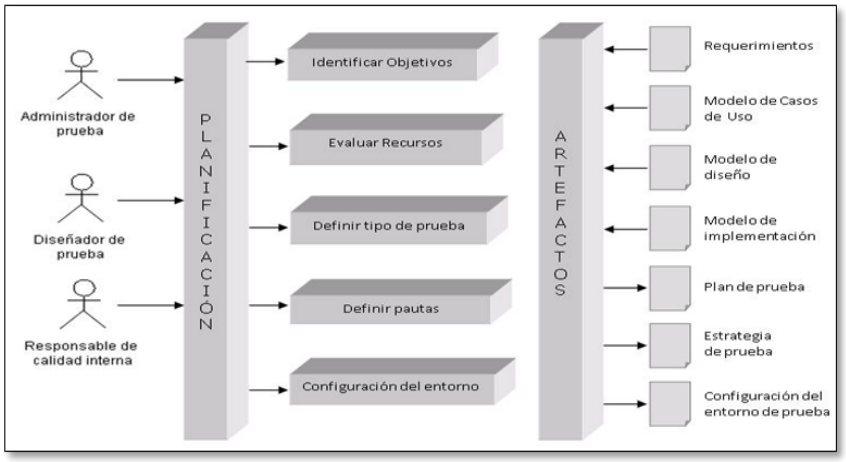


**Figura 2:** Planificación de las pruebas. (Elaboración propia)

Luego se tiene el diseño de pruebas, el cual consiste en identificar los casos de prueba y la estructura de los procedimientos de pruebas.

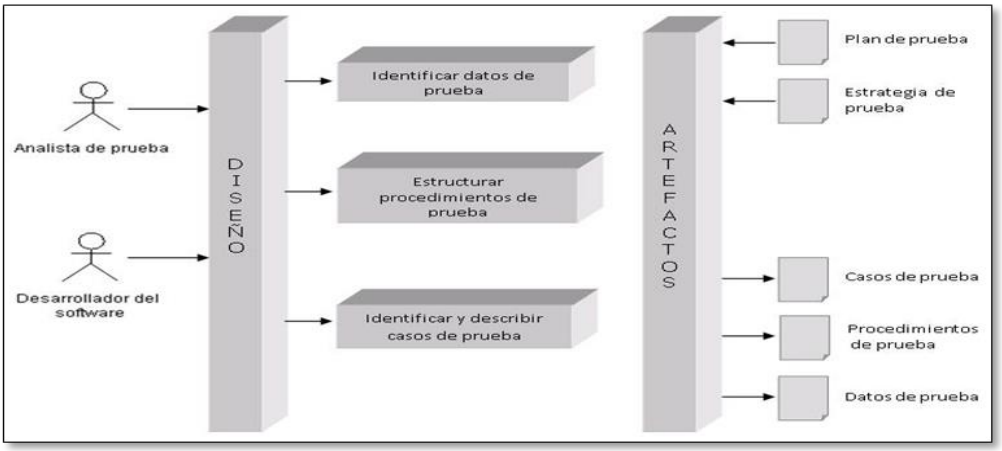


**Figura 3:** Diseño de las pruebas. (Elaboración propia)

Posteriormente se pasa a la implementación de las pruebas la cual consiste en automatizar los procedimientos de prueba obteniéndose los componentes de prueba correspondientes.

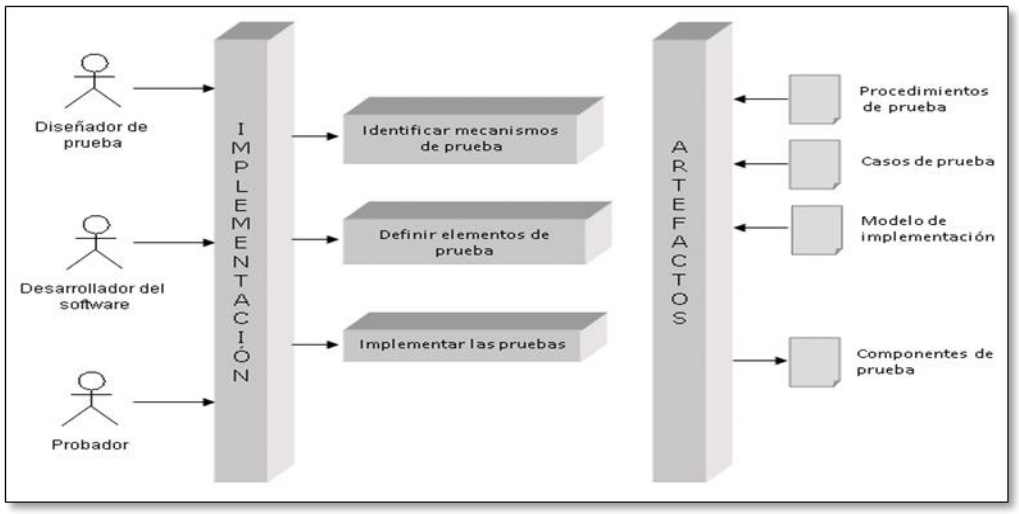


**Figura 4:** Implementación de las pruebas. (Elaboración propia)

Luego se pasa a la ejecución de las pruebas que consisten en comparar el comportamiento esperado del software con su comportamiento real, analizar las diferencias y reportar los resultados.

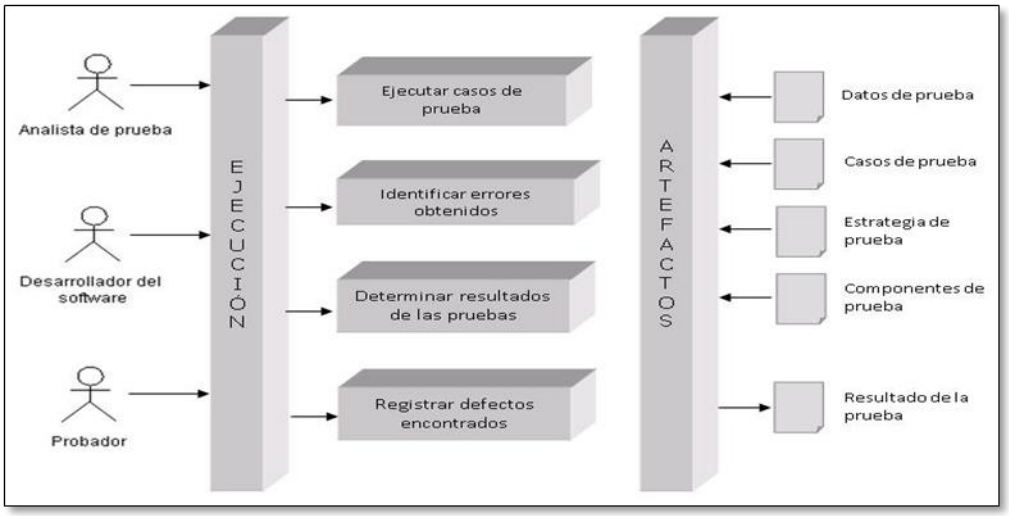


**Figura 5:** Ejecución de las pruebas. (Elaboración propia)

*Serie Científica de la Universidad de las Ciencias Informáticas*
Vol. 9, No. 4, Abril, 2016
ISSN: 2306-2495 | RNPS: 2343
http://publicaciones.uci.cu
Pág. 1-14

Por último, se tiene la evaluación de las pruebas la cual consiste en evaluar los resultados obtenidos y elaborar un documento con los errores detectados, las recomendaciones y observaciones para futuras pruebas a realizar, con vista a las próximas iteraciones a efectuar.

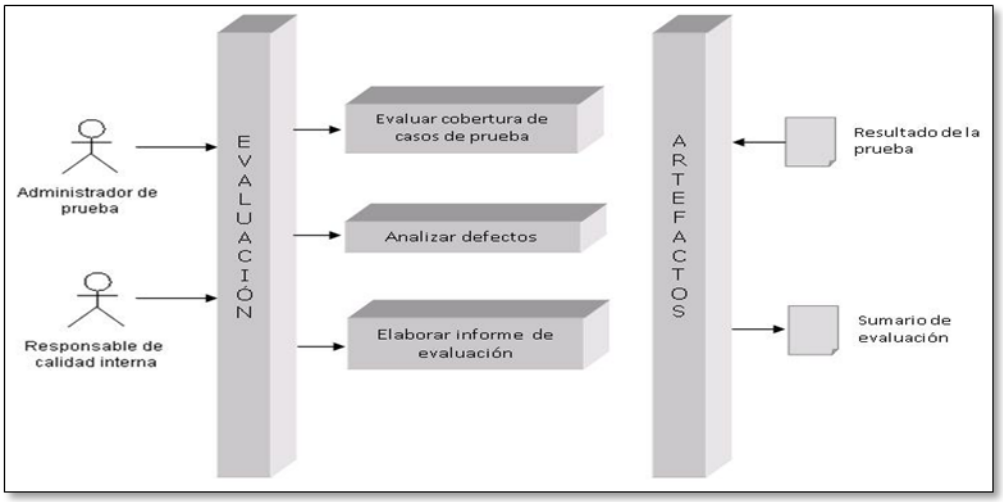


**Figura 6:** Evaluación de las pruebas. (Elaboración propia)

Para obtener un buen resultado a partir de cada uno de los procesos de prueba descritos anteriormente, es necesario tener en cuenta las herramientas para la automatización de las pruebas unitarias ya que son determinantes a la hora de definir el tiempo y el esfuerzo necesario para ejecutar las pruebas.

**Herramientas para la automatización de las pruebas unitarias.**

Existen un grupo de herramientas que ayudan en gran medida al proceso de automatización de las pruebas, facilitando el proceso en sistemas grandes y complejos y reforzando el desarrollo del proceso de pruebas, lo que contribuye a la disminución de tiempo, esfuerzo y costo.

Uno de los primeros *frameworks* desarrollados para realizar pruebas fue JUnit (para aplicaciones en Java), originalmente escrito por Kent Beck y Erich Gamma (MADEYSKI, 2010); en el que se utilizaban lenguajes de programación orientada a objetos (POO) y debido a su éxito se instauró el patrón xUnit que engloba a todos los *frameworks* para realizar pruebas unitarias automatizadas.

La mayoría de los lenguajes de programación, hoy cuentan con al menos una implementación de xUnit, donde se comparten características básicas que permiten lograr las siguientes funcionalidades: Especificar las pruebas como "métodos de prueba". Especificar los resultados esperados dentro del método de prueba en forma de llamadas a

"métodos de aserción". Agregar las pruebas en "suites de pruebas" que puedan ser ejecutadas como una sola operación. Ejecutar una o más pruebas para obtener un reporte de los resultados de la ejecución de las pruebas.

Algunos de los *frameworks* xUnit tienen soporte para descubrir los métodos de prueba o alguna forma de selección para ejecutar subconjuntos de métodos de prueba basados en algún criterio (MESZAROS, 2007).

Entre los *framworks* xUnit, existen versiones para C/C++ (CUnit y CPPUnit), Delphi (DUnit), PHP (PHPUnit), HTML (HTMLUnit), NUnit (plataforma .NET), VBUnit (Visual Basic) entre otros.

Todas estas herramientas ayudan al diseño y ejecución de las pruebas unitarias. Estas constituyen un conjunto de *frameworks* que permiten realizar la ejecución de clases de manera controlada para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Las pruebas unitarias se pueden efectuar de forma manual, pero cuando se van a aplicar a proyectos de gran magnitud esto se vuelve muy engorroso. En ocasiones, se hace muy difícil ejecutar un gran número de pruebas dado el factor tiempo, que es muy importante para el desarrollo total del proyecto. Por esto se considera que es una buena práctica la automatización de las pruebas para cualquier proyecto de software.

NUnit es un *framework open source* de pruebas unitarias para todos los lenguajes de la plataforma .NET que tiene la misma funcionalidad que JUnit. NUnit, como caso particular, es un *framework* desarrollado en C# que ofrece las funcionalidades necesarias para implementar pruebas en un proyecto. Provee una interfaz gráfica para ejecutar y administrar las mismas, informando si una prueba (o un conjunto de pruebas) falló, pasó o fue ignorada. Para ello compara los valores esperados y los valores generados, y si estos son diferentes la prueba no pasa, en caso contrario la prueba es exitosa.

Con NUnit se logra ejecutar pruebas automatizadas fácilmente. Contiene mecanismos para recolectar resultados de manera estructurada, produce varios tipos de reportes a partir de los resultados obtenidos en la ejecución de las pruebas, permite que existan relaciones entre ellas y que unas reutilicen código de otras. Además, soporta la jerarquía entre las pruebas pudiendo establecerse la prioridad y el orden de unas con otras (HUNT, et al., 2004).

NUnit.Forms y NUnit.ASP constituyen una expansión al *framework* núcleo NUnit; los que tienen como objetivo ampliar NUnit de manera que este sea capaz de manejar pruebas de elementos de interfaz de usuario tanto en Windows Forms como en ASP.NET. NUnit posee un conjunto de meta-atributos y aserciones que permiten probar los métodos de una clase especificada. Se puede ejecutar desde la consola o a través de una interfaz gráfica y se puede integrar con el Visual Studio en cualquiera de sus versiones.

*Serie Científica de la Universidad de las Ciencias Informáticas*
Vol. 9, No. 4, Abril, 2016
ISSN: 2306-2495 | RNPS: 2343
http://publicaciones.uci.cu
Pág. 1-14

## 3. Caso de estudio

Con el objetivo de automatizar el Sistema Registral y Notarial venezolano surge el proyecto Registros y Notarías. Para dicha automatización se crean varios sistemas entre los que se encuentra el Sistema Administración Financiera el cual tiene como propósito principal agilizar el proceso financiero y administrativo para los registros y notarías de Venezuela, eliminando los trámites innecesarios y facilitando mecanismos modernos de comunicación interdepartamental, poniendo a disposición de cada puesto de trabajo una función automatizada, que contempla desde la formulación de presupuestos, hasta la obtención de estados financieros finales.

El Sistema Administración Financiera en SAREN (Servicio Autónomo de Registros y Notarías) está conformado por los siguientes módulos relacionados entre sí: Administración, Presupuesto, Contabilidad, Requisiciones, Compras y Servicios, Retenciones, Recaudación, Tesorerías, Fondos en Anticipo y Caja Chica. El SAREN puede estar estructurado por la Unidad Administradora Central (UAC), Unidades Administradoras Desconcentradas (UAD) y Unidades Ejecutoras Locales (UEL), a partir de esta estructura es que se desarrolló el Sistema Administración Financiera del SAREN.

El trabajo desarrollado fue aplicado en el módulo Requisiciones del Sistema Administración Financiera, en el que se decidió emplear la herramienta NUnit que se ajusta a la plataforma .Net para llevar a cabo la automatización de las pruebas de software. Se realizó un análisis de los resultados reales y esperados con el propósito de corregir los defectos encontrados. Mediante la correcta aplicación de esta estrategia se garantiza que se realice la detección de errores en la fase correspondiente, garantizando un mejor aprovechamiento de los recursos para la realización de las pruebas.

A continuación, se muestra una comparación de los resultados obtenidos luego de haber ejecutado las pruebas siguiendo procedimientos diferentes en el módulo Requisiciones. El módulo Requisiciones fue desarrollado en paralelo por dos equipos con las mismas características en cuanto a experiencia y actitudes para el desarrollo de software. El equipo A, siguió el estándar de desarrollo propuesto en el proyecto con la aplicación de las pruebas de forma tradicional utilizando al equipo de calidad interna del proyecto. El equipo B aplicó el procedimiento de prueba descrito en este trabajo. Los resultados en cuanto a la cantidad de defectos detectados se encuentran en la Tabla 1.

Tabla 1. Resultados de la aplicación de la estrategia.

| Equipo | Defectos | Tiempo(días) | Tamaño(CU) | Pruebas realizadas |
|---|---|---|---|---|
| A | 36 | 21 | 15 | Caja Blanca y Caja Negra |
| B | 14 | 3 | 15 | Caja Blanca y Caja Negra |

## 3. Conclusiones.

En este trabajo se desarrolló una estrategia para aplicar las pruebas unitarias, las mismas cuentan con cinco procesos de manera que simplifican las estrategias que se han podido revisar en la bibliografía eliminando un grupo de tareas y actores asociados a modelos de desarrollo de software pesados. La experimentación realizada a través de estudio de casos permitió constatar una reducción importante de los defectos aproximadamente un 61 % y de igual forma una reducción del tiempo necesario para hacer las pruebas de un 85%. Estos resultados avalan la validez del planteamiento realizado, pero en función de la generalización de la propuesta se requieren desarrollar un mayor número de experimentos en el que se utilicen otros proyectos y otras metodologías de desarrollo; en este sentido estará dirigido el trabajo futuro en el que además se pretende desarrollar técnicas inteligentes que permitan hacer recomendaciones sobre el tipo de pruebas, teniendo en cuenta la información disponible en las entidades.

## Referencias

**CALISOFT. 2012.** *Tendencia de las No Conformidades en el Laboratorio Industrial de Pruebas de Software.* 2012.

**CANTONE, DANTE. 2008.** *La Biblia del programador e Implementación y Debugging.* s.l. : MP EDICIONES, 2008.

**DUSTIN, E., RASHKA, J. y al, et. 2000.** *Automated Software Testing: Introduction, management and performance.* s.l. : Addison-Wesley, 2000.

**GRAHAM, D. y FEWSTER, M. 1999.** *Software Test Automation.* s.l. : Addison-Wesley, 1999.

**HUNT, A. y THOMAS, D. 2004.** *Pragmatic unit testing in c# with nunit. The Pragmatic Programmers.* 2004.

**JACOBSON, I. 2003.** *Proceso unificado del desarrollo del software.* 2003.

**JOHNSON, R. 2003.** *Expert One-on-One J2EE Design and Development.* Indiana : Wiley, 2003.

**JOSKOWICZ, J. 2008.** *Reglas y prácticas en eXtreme Programming.* s.l. : Universidad de Vigo, 2008.

**KRUCHTEN, P. 2004.** *The rational unified process: an introduction.* s.l. : Addison-Wesley Professional., 2004.

**MADEYSKI, L. 2010.** *Test-Driven Development.* s.l. : Springer, 2010.

**MESZAROS, G. 2007.** *xUnit Test Patterns - Refactoring Test Code.* s.l. : Addison Wesley, 2007.

**ORÉ, ALEXANDER. 2014.** *CalidadySoftware.com.* [Disponible en: http://www.calidadysoftware.com/testing/pruebas_funcionales.php.].

**PRESSMAN, R. 2005.** *Ingeniería del Software. Un enfoque práctico.* s.l. : Mc Graw-Hill/ Interamericana de España. Quinta Edición., 2005.

**RIVERO, F. 2008.** *Pruebas Unitarias de Software en la Plataforma J2EE. Tesis de Diploma. UCI.* Habana : s.n., 2008.

**RUP, C. 2003.** *Ayuda del Rational.* 2003.

**TUYA, J, RAMOS, I y DOLADO, J. 2007.** *Técnicas cuantitativas para la gestión en la Ingeniería del Software.* s.l. : NETBIBLO, 2007.