Universidad de las Ciencias Informáticas
Facultad 2

Propuesta de Herramientas para Pruebas de Software Automáticas.

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas.

Autores:

- Andrés Luis Licea Castellanos
- Karel Alejandro Charro Pérez

Tutor: Ing. Dariena Ramírez Luján

Cotutor: Ing. Ariagnis Yero Guevara

Ciudad de La Habana, Junio 2009

"Año 51 de la Revolución"

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo, y autorizamos al Grupo de Procesamiento Digital de Imágenes y Señales de la Universidad de las Ciencias Informáticas a hacer uso del mismo, en su beneficio.

Para que así conste firmamos la presente a los ___ días del mes de _____ del año _____.

Autores:

_____ _____

Andrés Luis Licea Castellanos Karel Alejandro Charro Pérez

Tutores:

_____ _____

Ing. Dariena Ramírez Luján Ing. Ariagnis Yero Guevara

Datos de Contacto

Tutores:

Síntesis del Tutor: Ingeniero en Ciencias Informáticas y profesor de Ciencias Informáticas. Graduado por la Universidad de las Ciencias Informáticas en Julio del 2007. Actualmente profesor de la Universidad de las Ciencias Informáticas (UCI). Ha estado vinculado al trabajo productivo e investigativo en temas relacionados con la calidad del software al desempeñarse como Asesora de Calidad. Actualmente trabaja como Analista Principal en uno de los proyectos productivos de la Facultad 2.

Síntesis del Cotutor: Ingeniero en Ciencias Informáticas y profesor de Ciencias Informáticas. Graduado por la Universidad de las Ciencias Informáticas en Julio del 2007. Actualmente profesor de la Universidad de las Ciencias Informáticas (UCI). Ha estado vinculado al trabajo productivo en temas relacionados con las telecomunicaciones.

Dedicatoria

A nuestros padres,

por ser ellos la inspiración de este trabajo.

Agradecimientos

A mis padres por ser lo más grande que me ha dado la vida, por el apoyo, confianza, amor y educación que me han dado siempre bajo cualquier situación.

A mis hermanas por siempre ser ejemplos en mi vida, por darme tanto amor, cariño y por lo mucho que me han ayudado en toda mi vida.

A mi sobrina Lianet por siempre regalarme sus cariños y sonrisas.

A mis abuelos y tíos por haberme dado tanto cariño en la vida, enseñarme y educarme en todo momento.

A Omar Rosales (El Niño) por su cariño, ayuda y confianza en mí desde el primer momento en que nos conocimos.

A Maidolis y Blanco por darme tanta confianza, apoyo y cariño desde el primer momento en que llegué a su casa.

A mi novia Carmen Rosa por estar a mí lado en estos últimos años de mi vida estudiantil dándome tanto amor, momento felices, apoyando e inspirándome confianza en todo momento y ayudándome todo el tiempo en la realización de este trabajo.

A Disney y Luxmi por ser dos hermanos más y darme tanta amistad, cariño y ayuda en todo momento.

A mis amigos de todos los tiempos Daviel, Neury y Manolito por siempre estar en las buenas y las malas brindando amistad y apoyo.

A Edel por su amistad y ayuda todos estos años.

A Lisy por su amistad, ayuda y por tantos momentos de risas y alegrías que compartimos.

A mis amigos de la Peña por compartir tantos momentos de alegría, Guerra, Landy, Carlos, Charro, Santi, Pedry, Barry, Liorge y el Buty.

A Delmys y Liudmila por tanta ayuda que nos brindaron casi sin conocernos, dedicarnos tanto tiempo y bridarnos sus conocimientos.

A Karel Alejandro (El Charro) por haber compartido este trabajo conmigo, por el esfuerzo y dedicación mostrada en especial cuando yo no estuve.

En especial a nuestras Tutoras Dariena y Ariagnis Yero por siempre darnos tantos consejos, ayudarnos en cualquier momento, dedicarnos tanto tiempo y brindarnos siempre sus conocimientos.

Andrés

A mis padres por ser las personas más importante en mi vida y por ayudarme en todo momento.

A mis abuelos Flora y Julio que ya no están conmigo por todo el cariño y amor que me dieron.

A mi hermana por su cariño y porque sin su ayuda no hubiera podido completar el sueño de ser ingeniero.

A mi sobrina Daniela por tener siempre para mí esa hermosa sonrisa.

A mi cuñado José Luis por su apoyo en todo momento y su confianza en mí.

A Elizabeth, María, mi abuela Olga, Diana, Walfrido, Chichi y Galia por su ayuda tanto a mí como a mis padres.

A Juan Miguel, Maikel, Abel Chavez, Jorge, Nelson, Lewis, Kenny, Luis Carlos por pasar tantos buenos momentos con ellos.

A mis amigos de la Universidad Roberto, Andrés, Reiner, El Guerra, Pedro, El Buty, Lio, Landy, Carlos, Santy, Barry, Charly, William, Carmen, Natacha, Lisy, Madelaine y Duany.

A Andrés por haber compartido conmigo la realización de este trabajo.

A las tutoras Dariena y Ariagnis por ser tan pacientes con nosotros y por habernos dedicado tanto de su tiempo.

A Liudmila y Delmys por su ayuda incondicional en todo momento aun cuando no nos conocíamos.

A todas aquellas personas que a lo largo de mi vida han contribuido a este logro.

Karel.

Resumen

Este trabajo propone el uso de las herramientas PushToTest TestMaker y TestNG para su utilización dentro del proceso de pruebas en los Proyectos Productivos del Polo de las Telecomunicaciones. Se describen brevemente aspectos vinculados con este tema como son la calidad del software, tipos de pruebas, niveles de prueba y características de valoración de los resultados.

Se realiza una comparación entre un grupo de herramientas que son utilizadas a nivel mundial en el proceso de ejecución de pruebas automáticas, se justifica la utilización de las herramientas propuestas como herramientas para la automatización del proceso de pruebas en los proyectos productivos del Polo de las Telecomunicaciones. Se realiza una descripción en forma de manual de usuario de las herramientas para que estas fueran más fáciles de utilizar en los Proyecto Productivos.

Se aplica el proceso de pruebas con ambas herramientas en las aplicaciones producidas por los Proyectos Productivos del Polo de las Telecomunicaciones CUBACEL y TIP, quedando estas como un ejemplo más de la utilización de estas nuevas herramientas. Por último se realizó la validación de la propuesta a través del método Delphi contando con el criterio de 7 expertos en el tema de Pruebas de Software.

Índice

Índice de Figura

Índice de Tablas

Introducción

Numerosas empresas productoras de software desarrollan aplicaciones que ponen en juego vidas humanas o grandes inversiones económicas. La rama de la informática que se dedica al estudio de los procesos médicos, el comercio electrónico por Internet, en las instituciones educacionales, en sistemas de transportación como por ejemplo en la aeronáutica con la utilización de procesos automáticos para la navegación, son sólo ejemplos de tales aplicaciones. Desde el punto de vista económico, la detección tardía de un error puede cuestionar decisiones ya tomadas. El costo de estos errores es considerado el mayor en todas las etapas del proceso de desarrollo de sistemas informáticos. Existe, por lo tanto, una real necesidad de que estos softwares sean creados con la mayor calidad posible.

La calidad del software y el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia, son las razones que guían el desarrollo. Esta ha sido estudiada por varios expertos y algunos de ellos la definen como:

"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" [1].

"El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas". [2].

Las Pruebas de Software son un elemento crítico para la garantía de la calidad del software. Ellas son un conjunto de herramientas, técnicas y métodos que hacen la excelencia del desempeño de un programa, así como también la mejor publicidad que una empresa dedicada a la producción de software pueda tener. Se ha desarrollado una gran expectativa sobre el tema de "Cómo probar software de manera eficiente", lo que ha provocado un movimiento de información y de trabajos de investigación al respecto. Algunos de los tipos de Pruebas que más se utilizan son:

- ✓ Prueba de Unidad. Centra el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo.[2]

- ✓ Prueba de integración. Se basa en las Pruebas de conexiones y comunicaciones entre diferentes módulos. Es esencial en sistemas de cliente servidor o red.

- ✓ Prueba de sistema. Es una Prueba de caja negra incluyendo todos los componentes del sistema desde el hardware a la documentación.

- ✓ Prueba de aceptación: Prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

Se estima que las Pruebas de Software representan más de la mitad del costo de desarrollo de un programa. Hacer un sistema magistral cuesta mucho esfuerzo, probarlo cuesta más [3].

La Universidad de las Ciencias Informáticas (UCI) está enmarcada en todo este proceso de aplicación de Pruebas de Software en sus proyectos productivos y dentro de las mismas, el uso de herramientas de Pruebas automáticas de software que agilicen el desarrollo de aplicaciones. En este centro de altos estudios se creó una Dirección de Calidad de Software que son los encargados de controlar la calidad del software a nivel de universidad, el mismo está compuesto por los grupos de calidad de cada una de las facultades que están formados a su vez por los aseguradores de calidad de cada uno de los proyectos como se muestra en la siguiente figura.

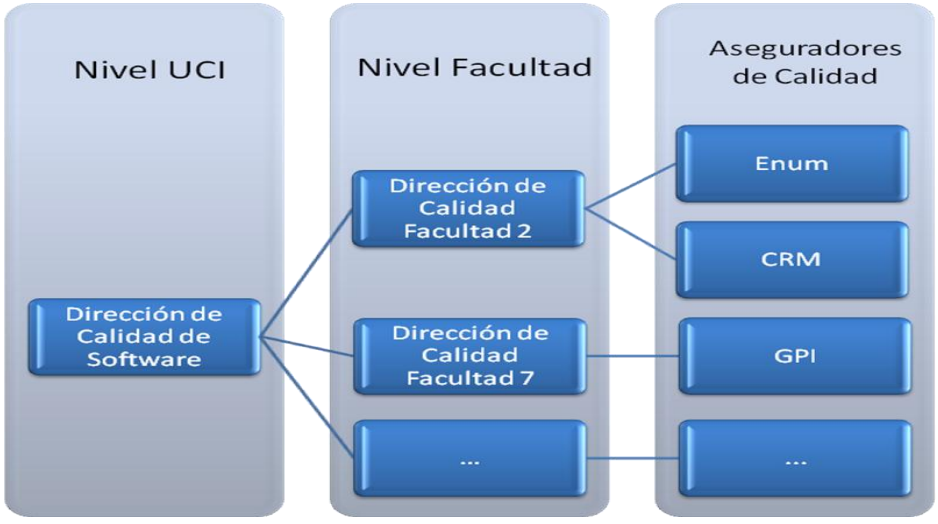


Fig. # 1 Estructura de la Dirección de Calidad de Software.

El software, después de ser creado, pasa por una serie de procesos que garantizan la calidad final del mismo

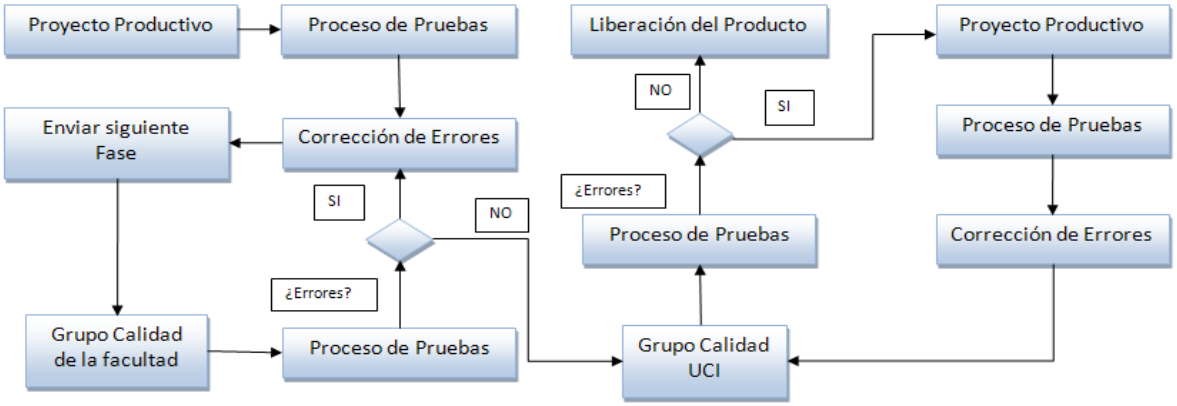


Fig. # 2 Proceso de revisión del software en la Universidad de las Ciencias Informáticas.

Este ciclo presenta una serie de deficiencias ya que al producto se le realizan, en la mayoría de los casos, Pruebas manualmente, y en algunas ocasiones los desarrolladores tienen muy poco tiempo para realizar un programa y por tanto se afecta el tiempo que se debe dedicar para probar las funcionalidades del producto, además la simulación de un escenario real aumenta el empleo de recursos. Otro problema relacionado con ello sería la precisión en cuanto a los resultados ya que el método manual puede traer consigo errores derivados del factor humano.

A pesar de que con el uso de herramientas de Pruebas automáticas de software se disminuye en el costo de la ejecución de las pruebas, el tiempo de realización de las mismas y se ahorra en conceptos de personal, en la Facultad 2, que desarrolla software relacionados con las Telecomunicaciones, en sus proyectos productivos no se ejecutan este tipo de pruebas e incluso los jefes de proyectos no tienen todo el conocimiento de qué Pruebas automáticas pueden aplicarle a los productos antes de pasar por la fase de liberación. Tampoco existe un estudio sobre las Herramientas Automáticas de Pruebas de Software que les permita conocer a ellos cuáles son las más óptimas según la tecnología manejada en la producción.

Por todo lo anteriormente planteado el problema científico que se presenta es: ¿Cómo optimizar el proceso de Pruebas de Software en los proyectos productivos del polo de telecomunicaciones? Para dar respuesta a esta problemática tomamos como objeto de estudio el proceso de Prueba de Software. Se ha definido como campo de acción: las

Herramientas para Pruebas de Software Automáticas, trazándose como objetivo general la Propuesta de Herramientas para Pruebas de Software Automáticas en correspondencia con las tecnologías del Polo de Telecomunicaciones de la Facultad 2.

Del objetivo general se derivaron los siguientes objetivos específicos que ayudan a darle un eficiente cumplimiento:

- ✓ Identificar herramientas para Pruebas de Software automáticas según cada una de las tecnologías usadas en el polo de Telecomunicaciones.
- ✓ Proponer herramientas para Pruebas de Software automáticas.

Siendo las tareas de la investigación para dar cumplimiento a los objetivos específicos las siguientes:

- ✓ Realizar estudio detallado de los diferentes tipos de Pruebas de Software automáticas que existen.
- ✓ Realizar estudio detallado de las diferentes tecnologías usadas en los proyectos productivos del polo de Telecomunicaciones.
- ✓ Identificar herramientas para Pruebas de Software automáticas según cada una de las tecnologías usadas en el polo de Telecomunicaciones.
- ✓ Proponer Herramientas para Pruebas de Software Automáticas.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción

Este capítulo muestra los temas fundamentales que se abordan a lo largo de la investigación. Se mencionan conceptos de calidad, pruebas y herramientas automáticas de pruebas, así como, otros aspectos relacionados con estos temas.

1.2 Calidad de Software

Primeramente para poder referirse al término de Calidad de Software es necesario remitirse al significado de calidad. Esta palabra puede adquirir múltiples interpretaciones, ya que todo dependerá del nivel de satisfacción o conformidad del cliente. Sin embargo, la calidad es el resultado de un esfuerzo arduo, donde se trabaja de forma eficaz para poder satisfacer el deseo del consumidor. Dependiendo de la forma en que un producto o servicio sea aceptado o rechazado por los clientes, podremos decir si éste tiene o carece de calidad.

En los libros se han propuesto muchas definiciones de calidad del software. Por lo que a esta investigación respecta, la calidad del software se define como:

“Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente. “

No hay duda de que esta definición puede ser modificada o ampliada. De hecho, no tendría fin una discusión sobre una definición formal de calidad del software.

La anterior definición permite hacer hincapié en tres puntos importantes:

- ✓ Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.
- ✓ Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software. Si no se siguen esos criterios, casi siempre habrá falta de calidad.

- ✓ Existe un conjunto de requisitos implícitos que a menudo no se mencionan (por ejemplo: el deseo por facilitar el uso y un buen mantenimiento). Si el software se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del software queda en entredicho [2].

1.3 Pruebas de Software

En todo proceso de desarrollo de aplicaciones es indispensable la presencia de un proceso de Pruebas de Software que coexista y se integre con este primero para garantizar así el buen funcionamiento y la calidad del producto final. Para lograr lo antes expuesto se debe partir del concepto de que las mismas desempeñan un papel fundamental en esta disciplina.

Las Pruebas de Software no son más que el proceso de ejecutar un sistema o componente, para medir y mejorar su calidad, bajo condiciones específicas, con la intención de encontrar errores, observando y grabando los resultados, y haciendo una evaluación de algunos aspectos del sistema o componente en cuestión [4].

Es importante resaltar que no siempre es necesario tener una aplicación funcionando para realizarle las evaluaciones pertinentes y lograr así su mejor funcionamiento, sino que si se lleva a cabo un proceso de prueba bien diseñado, que tenga pronosticado probar en cada una de las etapas del desarrollo, se eliminan muchos de los errores que posteriormente provocarían grandes gastos económicos, de tiempo y de esfuerzo humano.

Toda prueba de software desempeña un papel fundamental en el desarrollo de cualquier tipo de aplicación, pero si se estudia la mejor forma de hacerlo, siguiendo los pasos de acuerdo con los especialistas en el tema, se incrementan las posibilidades de que esta llegue a un feliz término y arroje resultados más cercanos a los esperados, permitiendo así, realizar a posteriori un mejor análisis de la situación. Para ilustrar mejor esta situación, se presentan a continuación algunos objetivos, características y principios con que deben contar las Pruebas de Software.

1.3.1 Objetivos de las Pruebas de Software

Éstos son algunos de los objetivos que debe perseguir todo diseño y ejecución de Pruebas de Software [5]:

- ✓ Probar si el software no hace lo que debe.

- ✓ Probar si el software hace lo que no debe, es decir, si provoca efectos secundarios adversos.
- ✓ Descubrir un error que aún no ha sido descubierto.
- ✓ Encontrar el mayor número de errores con la menor cantidad de tiempo y esfuerzo posibles.
- ✓ Mostrar hasta qué punto las funciones del software operan de acuerdo con las especificaciones y requisitos del cliente.

1.3.2 Características de las Pruebas de Software

Roger Pressman [2], Kaner, Falk y Nguyen [6] sugieren algunas de las características que debe tener una «buena prueba»:

- ✓ Una buena prueba tiene una alta probabilidad de encontrar un error. El ingeniero de software debe tener un alto nivel de entendimiento de la aplicación a construir para poder diseñar casos de prueba que encuentren el mayor número de defectos.
- ✓ Una buena prueba no debe ser redundante. Uno de los objetivos de las pruebas es encontrar el mayor número de errores con la menor cantidad de tiempo y esfuerzo posibles, por lo cual no se deben diseñar casos de prueba que tengan el mismo propósito que otros, sino que se debe tratar de diseñar el menor número de casos de prueba que permitan probar adecuadamente el software y optimizar los recursos.
- ✓ Una buena prueba debería ser la mejor de la cosecha. La limitación en tiempo y recursos puede impedir que se ejecuten todos los casos de prueba de un grupo de pruebas similares por lo cual en estas situaciones se debería seleccionar la prueba que tenga la mayor probabilidad de descubrir errores.
- ✓ Una buena prueba no debería ser ni demasiado sencilla ni demasiado compleja.

1.3.3 Principios de las Pruebas de Software

A continuación se mencionan algunos de los principios básicos que guían las pruebas del software [2] [7]:

- ✓ A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.

- ✓ Deberían planificarse mucho antes de que empiecen. La planificación de las pruebas puede empezar tan pronto como esté completo el modelo de requisitos y se tenga consolidado el modelo de diseño.

- ✓ Deberían empezar por «lo pequeño» y progresar hacia «lo grande». Las primeras planeadas y ejecutadas se centran generalmente en módulos individuales del programa. A medida que se avanza en éstas, el enfoque de las pruebas cambia en un intento de encontrar nuevos errores relacionados con la integración de estos módulos y finalmente con la interacción del sistema completo.

- ✓ No son posibles las pruebas exhaustivas. El número de permutaciones de caminos para incluso un programa de tamaño moderado es demasiado grande. Por lo cual, es imposible ejecutar todas las combinaciones de caminos durante las mismas. Sin embargo, es posible elegir y ejecutar una serie de caminos lógicos importantes que permitan probar adecuadamente el software.

- ✓ Para ser más eficaces, deberían ser realizadas por un equipo independiente. El ingeniero de software que creó el sistema no es el más indicado para realizar las pruebas debido a que consciente o inconscientemente puede omitir casos de prueba importantes que conlleven a descubrir nuevos errores. Por consiguiente, es recomendable organizar un grupo de trabajo independiente para las mismas que suministre una visión más objetiva del software.

- ✓ Se debe inspeccionar a fondo los resultados de cada prueba.

- ✓ Examinar un programa para ver que haga lo que se espera, es sólo la mitad de la batalla; la otra mitad es ver si hace lo que no se espera.

- ✓ No se debe hacer una planificación de los esfuerzos de prueba bajo la suposición de que no se encontrarán errores.

- ✓ La probabilidad de que existan más errores en una sección de programa es proporcional al número de éstos que se hayan encontrado en esa sección.

- ✓ Son una tarea extremadamente creativa e intelectualmente desafiante.

1.4 Procesos de Prueba de Software

Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software debe ir acompañado de una actividad que garantice la calidad. El proceso de Pruebas de Software es una parte importante en la obtención de productos con la calidad requerida y representa una revisión final de las especificaciones, del diseño y de la codificación.

Con el objetivo de encontrar el mayor número posible de errores, las pruebas deben conducirse sistemáticamente, y los casos de prueba deben diseñarse utilizando técnicas definidas. El software debe probarse desde dos perspectivas diferentes [8]:

- ✓ La lógica interna del programa se comprueba usando técnicas de diseño de casos de prueba de caja blanca.

- ✓ Los requisitos del software se comprueban utilizando técnicas de diseño de casos de prueba de caja negra.

En ambos casos, se intenta encontrar el mayor número de errores con la menor cantidad de esfuerzo y tiempo.

El Proceso de Pruebas de Software se encuentra regido por los casos de prueba, que tienen el propósito o la intención de descubrir un error, de hecho un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces. La prueba demuestra hasta qué punto las funciones del software parecen funcionar de acuerdo con las especificaciones y parecen alcanzarse los requisitos de rendimiento. Además los datos que se van recogiendo a medida que se lleva a cabo la prueba proporcionan una buena indicación de la fiabilidad del software y, de alguna manera, indican la calidad del software como un todo. Pero, la prueba no puede asegurar la ausencia de defectos; solo puede demostrar que existen defectos en el software [2].

A nivel internacional las instituciones y empresas dedicadas a la industria del software emplean en su actividad de desarrollo metodologías. Estas metodologías establecen un conjunto de actividades que definen cómo se debe hacer el software, quién debe hacer cada actividad, cuándo hacerla y qué se debe hacer. En la actualidad existe gran cantidad

de metodologías orientadas al proceso de desarrollo de software, siendo una de las más significativas *Rational Unified Process (RUP)*,

RUP se caracteriza por ser Dirigido por casos de uso donde los casos de uso definen lo que el usuario desea a partir de la captura de requisitos y la modelación del negocio. Es Centrado en la Arquitectura, característica que brinda una visión completa del sistema, se describen los procesos del negocio que son más importantes, para comprenderlo, desarrollarlo y producirlo de una forma eficaz. Iterativo e Incremental donde cada fase se desarrolla en iteraciones, de forma tal que se pueda dividir en pequeños proyectos mejorando su comprensión y desarrollo.

1.4.1 Flujo Trabajo de Prueba en RUP

En el flujo de trabajo de prueba se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema a ser entregadas a terceros.

Los objetivos de las pruebas de acuerdo a lo planteado en la Metodología de Desarrollo de Software Rational Unified Process son [9]:

- ✓ Planificar las pruebas necesarias en cada iteración, incluyendo las de integración y las de sistema. Las pruebas de integración son necesarias para cada construcción dentro de la iteración, mientras que las de sistema son necesarias sólo al final de la iteración.

- ✓ Diseñarlas e implementarlas creando los casos de pruebas que especifican qué probar, creando los procedimientos de pruebas que especifican cómo realizarlas y creando, si es posible, componentes de pruebas ejecutables para automatizarlas.

- ✓ Realizar las diferentes pruebas y manejar los resultados de cada una sistemáticamente. Las construcciones en las que se encuentran defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los defectos importantes puedan ser arreglados.

El Flujo de Trabajo de Prueba dentro de RUP está compuesto por el papel que éstas desempeñan en la vida del software, los artefactos que utilizan, como son los modelos de pruebas; los casos de pruebas; los procedimientos; los componentes; los planes de

prueba; los defectos y la evaluación de las mismas; además, se detallan los trabajadores involucrados en este proceso como son: el diseñador de pruebas, el ingeniero de componentes, el ingeniero de pruebas de integración y el ingeniero de pruebas del sistema. También se abordan los flujos de trabajo que este proceso contiene como: planificar las pruebas; diseñarlas; implementarlas; realizar las pruebas de integración y las de sistema y, por último, evaluar las pruebas; lo cual se detallará a continuación [2].

Fig. # 3 Los trabajadores y artefactos involucrados en las pruebas [9].

1.4.1.1 El papel de la prueba en el ciclo de vida del software

Durante la fase de inicio los ingenieros de prueba se van poniendo al corriente de la naturaleza general del sistema propuesto, van considerando qué pruebas requerirá y van desarrollando algunos planes provisionales de prueba. En la fase de elaboración el objetivo es asegurarse que los subsistemas de todos los niveles y de todas las capas funcionen, solo se pueden probar los componentes ejecutables. Sin embargo, las pruebas se llevan a cabo sobre todo en la fase de construcción, cuando el resultado de la implementación es sometida a pruebas de unidad, integración y de sistema. Esto quiere decir que la realización de las mismas se centra en las fases de elaboración, cuando se prueba la línea base ejecutable de la arquitectura, y de construcción, cuando el grueso del sistema está implementado. Durante la fase de transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y a las pruebas de regresión (Ver figura 4).

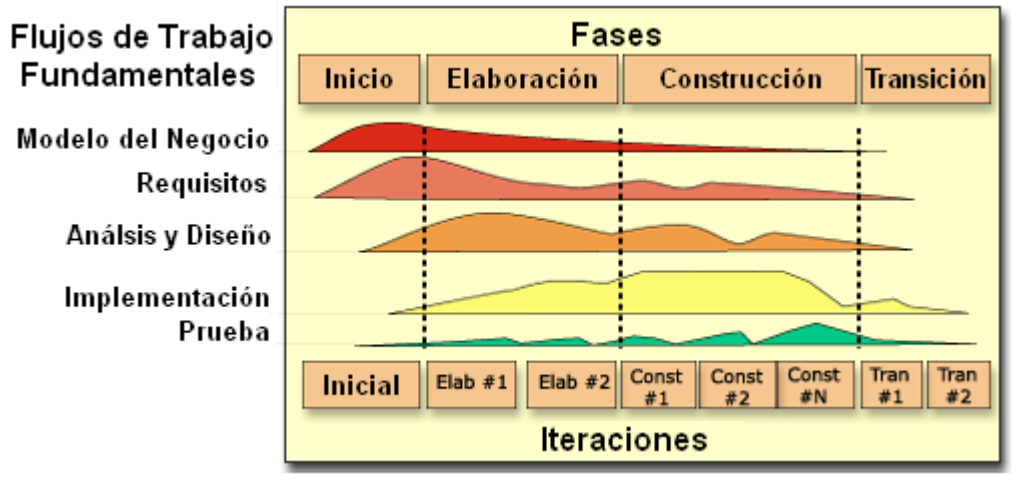


Fig. # 4 Flujos de trabajo fundamentales de RUP [9].

Es importante mantener el modelo de prueba a lo largo del ciclo de vida del software completo, aunque el modelo de prueba cambia constantemente debido a:

- ✓ La eliminación de los *casos de pruebas* obsoletos (y los correspondientes procedimientos y componentes de éstas).
- ✓ El refinamiento de algunos casos de prueba en *casos de pruebas* de regresión.
- ✓ La creación de nuevos *casos de uso* para cada nueva construcción.

1.4.1.2 Artefactos del Flujo de Trabajo de Prueba

Los artefactos que RUP propone para llevar a cabo el proceso de prueba son los siguientes [9]:

- ✓ *Modelo de pruebas:* describe principalmente cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. Además describe cómo han de ser probados los aspectos específicos del sistema; por ejemplo, si la interfaz de usuario es utilizable y consistente o si el manual de usuario del sistema cumple con su cometido.
- ✓ *Caso de prueba:* especifica una forma de probar el sistema, incluyendo la entrada o resultados con la que se ha de verificar y las condiciones para esto.

- ✓ *Procedimiento de prueba:* específica cómo realizar un caso de prueba, varios o parte de éstos. Por ejemplo, puede ser una instrucción para un individuo sobre cómo ha de realizar un caso de pruebas manualmente, o puede ser una especificación de cómo interactuar manualmente con una herramienta de automatización de pruebas para crear componentes ejecutables.

- ✓ *Componente de prueba:* automatiza un procedimiento de prueba, varios o parte de ellos. Puede ser grabado por una herramienta de automatización. Se utiliza para probar los componentes en el modelo de implementación, proporcionando entradas de prueba, controlando y monitorizando la ejecución de los componentes y, posiblemente, informando de los resultados de las pruebas.

- ✓ *Plan de prueba:* describe las estrategias, recursos y planificación de la prueba. Incluye la definición del tipo de prueba a analizar para cada iteración y sus objetivos, el nivel de cobertura de ésta y de código necesario además del porcentaje de pruebas que deberían ejecutarse con un resultado específico.

- ✓ *Defecto:* es una anomalía del sistema y puede ser utilizado para localizar cualquier aspecto que los *desarrolladores* necesiten registrar como síntoma de un problema en el sistema que se necesita controlar y resolver.

- ✓ *Evaluación de prueba:* este artefacto se encarga de la evaluación de los resultados de los esfuerzos de prueba, tales como la cobertura del caso de prueba, la cobertura de código y el estado de los defectos.

1.4.1.3 Trabajadores del Flujo de Trabajo de Prueba

Los trabajadores que se encuentran en esta fase de Prueba de RUP [Jacobson1999] son:

- ✓ *Diseñador de pruebas:* es el responsable de la integridad del modelo de pruebas, asegurando que el mismo cumple con su propósito, además planea las pruebas, decidiendo los objetivos apropiados y la planificación de las mismas. Esta persona se encarga de seleccionar y describir los casos y procedimientos de prueba correspondientes a las necesidades pertinentes, incluyendo en su labor, la evaluación de los resultados de las de integración y de sistema una vez ejecutadas.

- ✓ *Ingeniero de componentes:* es el responsable de los *componentes de pruebas* que automatizan algunos de los procedimientos. No todos los procedimientos de pruebas pueden ser automatizados, porque la creación de dichos componentes puede necesitar de sustanciales habilidades como programador.

- ✓ *Ingeniero de pruebas de integración:* es el responsable de realizar las pruebas de integración que se necesitan para cada construcción producida en el flujo de trabajo de implementación, además se encarga de documentar los defectos en los resultados de pruebas de integración.

- ✓ *Ingeniero de pruebas de sistema:* es el responsable de realizar las pruebas de sistema necesarias sobre una construcción que muestra el resultado de una iteración completa, además se encarga de documentar los defectos en los resultados de pruebas de sistema.

1.4.1.4 Actividades del Flujo de Trabajo de Prueba

Dentro de los flujos de trabajo que se encuentran descritos en RUP existen diversas actividades, como son [9]:

- ✓ *Planificar prueba:* es planificar los esfuerzos de prueba en una iteración describiendo una estrategia, estimando los requisitos para el esfuerzo de la misma, por ejemplo: los recursos humanos y sistemas necesarios.

- ✓ *Diseñar prueba:* tiene como propósito identificar y describir los casos de prueba para cada construcción, además de identificar y estructurar los procedimientos de prueba, especificando cómo realizar los casos de prueba.

- ✓ *Implementar prueba:* es automatizar los procedimientos de prueba creando *componentes de prueba* en caso de ser posible, pues no todos se pueden automatizar.

- ✓ *Realizar pruebas de integración:* se encarga de realizar las pruebas de integración necesarias para cada una de las construcciones creadas en una iteración y de recopilar los resultados.

- ✓ *Realizar pruebas de sistema:* tiene como objetivo realizar las pruebas de sistema necesarias en cada iteración y recopilar los resultados. Éstas se realizan de igual

forma que las pruebas de integración y empiezan cuando estas últimas indican que el sistema satisface los objetivos de calidad de integración fijados en el *plan de prueba* de la actual iteración.

- ✓ *Evaluar prueba:* es evaluar los esfuerzos de prueba en una iteración, comparando los resultados obtenidos con los objetivos trazados en el *plan de prueba*. A partir de esto, los diseñadores de pruebas preparan métricas que les permiten determinar el nivel de calidad del software y qué cantidad de ellas es necesario realizar.

1.5 Niveles de Prueba

Las pruebas del software se encuentran en cada etapa del desarrollo del software. En la medida que el trabajo de los desarrolladores se incrementa, va aumentando el volumen de la aplicación, las pruebas van cambiando de estrategias y técnicas, presentándose como una nueva etapa, estas están definidas en niveles que tienen nuevos objetivos, entornos y resultados. Estos niveles especifican qué tipos y métodos de pruebas se deben utilizar en cada uno de ellos y cuáles son sus objetivos.

Los niveles de prueba son:

- ✓ Unidad

Se prueba la interfaz del módulo para asegurar que la información fluya de forma adecuada hacia y desde la unidad de programa que está siendo probada. Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las condiciones límites para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento. Se ejercitan todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Finalmente, se prueban todos los caminos de manejo de errores.

Tipos:

- ➢ Caja Negra

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

- Caja Blanca

Se comprueba los caminos lógicos del software proponiendo casos de prueba que se ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

- ✓ Integración

Es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es coger los módulos probados mediante la prueba de unidad y construir una estructura de programa que está de acuerdo con lo que dicta el diseño.

Tipos:

- Integración Descendente

Es un planteamiento incremental a la construcción de la estructura de programas. Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando por el módulo de control principal (programa principal). Los módulos subordinados (subordinados de cualquier modo) al módulo de control principal se van incorporando en la estructura, bien de forma primero-en-profundidad, o bien de forma primero-en-anchura.

- Integración Ascendente

La prueba de la integración ascendente, como su nombre indica, empieza la construcción y la prueba con los módulos atómicos (es decir, módulos de los niveles más bajos de la estructura del programa). Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los

módulos subordinados siempre está disponible y se elimina la necesidad de resguardos.

- Pruebas de Regresión

  Las pruebas de regresión son volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios (debidos a las pruebas o por otros motivos) no introducen un comportamiento no deseado o errores adicionales y que no ocasionen efectos colaterales.

  Se pueden hacer manualmente, volviendo a realizar un subconjunto de todos los casos de prueba o utilizando herramientas automáticas de reproducción de captura.

- Pruebas de Humo

  Las pruebas de humo son un método de prueba de integración que es comúnmente utilizada cuando se ha desarrollado un producto software empaquetado. Son diseñadas como un mecanismo para proyectos críticos por tiempo, permitiendo que el equipo de software valore su proyecto sobre una base sólida.

  El progreso es fácil de observar. Cada día que pasa, se integra más software y se demuestra que funciona.

✓ Sistema

La prueba de sistema, realmente, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar el sistema sobre el hardware en que será instalado posteriormente. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

Tipos:

- Funcionales

Enfocadas a los requisitos funcionales del software. Comprueban el correcto comportamiento de las acciones que debe realizar el sistema

- No funcionales

Enfocadas a los requisitos no funcionales del proyecto.

- ✓ Pruebas de Resistencia o Estrés

Ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Por ejemplo: ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos.

- ✓ Pruebas de Volumen

Someten a la aplicación a volúmenes pesados de datos y tienen como objetivo demostrar que el programa no puede manejar el volumen de datos. Puesto que las pruebas de volumen pueden requerir una cantidad de recursos considerables, en términos de cantidad de máquinas y de tiempo por parte de las personas, no se puede ir demasiado lejos. Sin embargo, todo programa debe ser expuesto a pruebas de volumen

- ✓ Pruebas de Rendimiento o Carga

Verificar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado y la capacidad del sistema para manejar volúmenes de datos extremos de acuerdo al tiempo de respuesta establecido para el sistema.

- ✓ Pruebas de Seguridad

Las pruebas de seguridad intentan verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de accesos impropios. Durante la prueba de seguridad, el responsable de la prueba desempeña el papel de un individuo que desea entrar en el sistema y debe intentar conseguir las claves de acceso por cualquier medio, puede atacar al sistema con software diseñado para romper cualquier defensa.

- ✓ Liberación

  - ➢ Prueba el software funcionando como un todo. Trata de probar que los objetivos para los que fue construida la aplicación no se cumplen en su totalidad y que por tanto el software no está en condiciones de ser presentado al cliente y hay que hacer modificaciones a la aplicación. Se realiza cuando el software funciona como un todo y está prácticamente listo para ser presentado al cliente.

- ✓ Aceptación

Prueba el software funcionando como un todo, se realiza dándole un uso real a la aplicación y es realizada por parte de los clientes, los errores que se encuentren en la misma son reportados como defectos.

Tipos:

- ➢ Pruebas alfa

  Las realiza el usuario en presencia de personal de desarrollo del proyecto haciendo uso de una máquina preparada para tal fin usando el software de forma natural.

- ➢ Pruebas beta

  Las realiza el usuario después de que el equipo de desarrollo les entregue una versión casi definitiva del producto y no están presentes los desarrolladores.

1.6 Herramientas de automatización de las pruebas

Para comenzar a hablar del uso de herramientas para la realización de pruebas automatizadas, es necesario el conocimiento de dos conceptos básicos [8]:

- ✓ Prueba manual*:* "Es aquella prueba realizada por una o más personas que interactúan directamente con el sistema. Estas personas verifican si los resultados obtenidos son válidos o no".

- ✓ Prueba automática*:* "Es aquella realizada por un programa o herramienta que prueba el sistema sin necesidad de la interacción de una persona. La herramienta

suministra una serie de valores de prueba, o acciones de prueba al sistema y verifica los resultados devueltos por éste con los resultados esperados".

Las Pruebas de Software se pueden ejecutar de cualquiera de estas dos formas. Toda forma automática trae grandes beneficios, pues simplifica el trabajo y el esfuerzo humano necesario para realizar cualquier tipo de tarea, además de aportar velocidad, eficiencia, precisión, aunque generalmente ante estas ventajas, se le anteponen ciertas desventajas, como por ejemplo, una gran cantidad de aplicaciones que contienen elementos que no son compatibles con las herramientas que se utilizan para ponerlas a prueba, en consecuencia esto requiere la búsqueda de soluciones creativas para que las herramientas se adapten a la aplicación, lo cual constituye un obstáculo al que se tendrá que enfrentar el equipo de trabajo.

Según la metodología RUP las herramientas de prueba se pueden categorizar según las funciones que realicen. Algunas designaciones de funciones típicas para herramientas son:

- ✓ **Herramientas de Adquisición de Datos** que adquieren datos para utilizar en las tareas de prueba. Los datos se pueden adquirir mediante la conversión, la extracción, la transformación o la captura de datos existentes, o mediante la generación de guiones de uso o especificaciones suplementarias.

- ✓ **Herramientas Estáticas de Medida** que analizan información contenida en los modelos de diseño, el código fuente u otros orígenes fijos. El análisis produce información en el flujo lógico, el flujo de datos o la métrica de calidad, como la complejidad, el mantenimiento o las líneas de código.

- ✓ **Herramientas Dinámicas de Medida** que realizan un análisis durante la ejecución del código. Las medidas incluyen la operación de tiempo de ejecución del código, como la memoria, la detección de errores y el rendimiento.

- ✓ **Simuladores o Controladores** que realizan tareas que, por cuestiones de tiempo, gastos o seguridad no están disponibles para las pruebas.

- ✓ **Herramientas de Gestión de Pruebas** que ayudan en la planificación, el diseño, la implementación, la ejecución, la evaluación y la gestión de tareas de prueba o productos de trabajo.

Además las herramientas de prueba suelen caracterizarse como cajas blancas o cajas negras en función de cómo se utilicen, o la tecnología y los conocimientos necesarios para utilizarlas.

- ✓ Las **Herramientas de Caja Blanca** dependen del conocimiento del código, los modelos de diseño y otro material de origen para implementar y ejecutar las pruebas.

- ✓ Las **herramientas de Caja Negra** dependen de los guiones de uso o la descripción funcional del destino de la prueba.

Las Herramientas de Caja Blanca saben cómo procesa la solicitud el destino de la prueba, mientras que las herramientas de caja negra dependen de las condiciones de entrada y de salida para evaluar la prueba.

Además de las amplias clasificaciones de herramientas que se presentaron antes, las herramientas también se pueden clasificar según la especialización.

- ✓ Las **Herramientas de Grabación y Reproducción** combinan la adquisición de datos con la medida dinámica. Los datos de prueba se adquieren durante la grabación de sucesos (conocida como implementación de la prueba). Más tarde, durante la ejecución de la prueba, los datos se utilizan para reproducir el script de prueba, que se utiliza para evaluar la ejecución del destino de la prueba.

- ✓ Las **Herramientas de Métrica de Calidad** son herramientas de medida estática que realizan un análisis estático de los modelos de diseño o código fuente para establecer un conjunto de parámetros que describen la calidad del destino de la prueba. Los parámetros pueden indicar fiabilidad, complejidad, mantenimiento u otras medidas de calidad.

- ✓ Las **Herramientas de Supervisión de la Cobertura** indican la completitud de la prueba mediante la identificación de la cantidad de destino de la prueba cubierta, en alguna dimensión, durante la prueba. Las clases típicas de cobertura son guiones de uso (basados en requisitos), nodo o ramificación lógica (basada en código), estado de los datos y puntos de función.

- ✓ Los **Generadores de Guiones de Prueba** automatizan la generación de los datos de prueba. Utilizan o bien una especificación formal de las entradas de datos del

destino de la prueba o bien los modelos de diseño y el código fuente para producir datos de prueba que prueben las entradas nominales, las entradas de errores y los guiones de límite.

- ✓ Las **Herramientas del Comparador** comparan los resultados de la prueba con los resultados de referencia e identifican las diferencias. Los comparadores se diferencian en su especificación para formatos de datos particulares. Por ejemplo, pueden basarse en píxeles para comparar imágenes de mapa de bits o en objetos para comparar las propiedades o los datos del objeto.

- ✓ Los **Extractores de Datos** proporcionan entradas para los guiones de prueba de orígenes existentes, incluidos bases de datos, secuencias de datos de un sistema de comunicación, informes o modelos de diseño y código fuente.

1.7 Tabla comparativa

A continuación se aprecian dos tablas que resumen una parte del estudio realizado, con el afán de mostrar claramente las bondades de algunas de estas herramientas y facilitar la comparación entre las mismas.

Los aspectos a tener en cuenta para realizar dicha comparación son:

- ✓ *Tipos de pruebas:* se analizan los tipos de pruebas que se pueden automatizar con la herramienta. Es un parámetro importante pues permite conocer si la herramienta se adapta al enfoque de automatización que se espera obtener, o sea, para ver si es útil según el plan de automatización elaborado.

- ✓ *Modo de adquisición:* se analiza la manera en que se puede acceder a la herramienta, si es de forma gratuita o bajo el pago de alguna licencia.

- ✓ *Tipo de Reportes:* se analiza la forma en que la herramienta presenta las salidas de las pruebas realizadas, sobre todo, hacia qué formatos exporta esta información. Por ejemplo si permite hacerlo en forma de texto, de tablas, de gráficos, hacia formatos amigables como son: pdf, doc, xls, HTML. Ésta es una característica importante a analizar, sobre todo en las herramientas que permiten realizar Pruebas de Estrés.

- ✓ *Soporte para AJAX:* analiza si la herramienta cubre la necesidad de encontrar defectos en sistemas implementados con este tipo de tecnología, pues dada su novedad se hace de vital importancia este análisis.

- ✓ *Integración con entornos de desarrollo:* se analiza si la herramienta brinda la posibilidad de integrarse con uno o varios entornos de desarrollo, pues a partir del que sea usado por el equipo de trabajo se escogen las herramientas para la realización de las pruebas.

- ✓ *Bibliografía:* se analiza la existencia de materiales de estudio para comprensión del uso de la herramienta.

Aunque en las tablas se hacen alusión a solo 3 herramientas de cada tipo de pruebas, existen en la actualidad muchas más, pero las escogidas son, a consideración de esta investigación, las más completas.

Aspectos\Herramientas	PushToTest TestMaker	Jakarta JMeter	QALOAD de Compuware
Tipos de Prueba que realizan.	Funcionales, Carga, Estrés, Volumen, Regresión y monitoreo de la aplicación.	Funcionales, Carga, Estrés y Regresión.	Carga, Estrés y Monitoreo de la aplicación
Modo de Adquisición.	Gratuita	Gratuita	Compra de una licencia
Tipo de Reportes.	Reporte HTML que contiene una serie de gráficos y tablas	Tablas de resultados y gráficos	Muestra los resultados de las pruebas en una amplia variedad de informes y gráficas
Bibliografía.	Posee una amplia documentación en el sitio de la aplicación. Cuando es instalado cuenta con una gran cantidad de ejemplos	Existe una amplia documentación sobre la utilización el modo de empleo de la herramienta y los requerimientos no funcionales que ella precisa.	Posee tutoriales pero estos son comercializados bajo la licencia.

Soporte de Ajax.	Sí	No	Sí
Integración con entornos de desarrollo.	No	No	No

Tabla # 1 Comparación de las herramientas de Caja Negra.

Aspectos\Herramientas	TestNG	JUnit	JTest
Tipos de Prueba que realizan	Unidad, Funcionales, Integración, Regresión y pruebas de tiempo de los algoritmos	Unidad, Funcionales, Integración, Regresión.	Unidad, Funcionales, Integración, Regresión y pruebas de Base de Datos
Modo de Adquisición	Gratuita	Gratuita	Compra de una licencia.
Reportes	Generación de un archivo HTML, Consola.	Consola.	Genera reporte HTML y XML.
Bibliografía	Posee amplia bibliografía así como libros de manera gratuita	Posee amplia bibliografía así como libros de manera gratuita	Los libros se obtienes mediante el pago de la licencia.
Integración con entornos de desarrollo	Eclipse, NetBeans, IDEA, Intellij, Maven	Eclipse, NetBeans, IDEA, ANT, JDeveloper	Eclipse, IDEA, Intellij, ClearCase, RAD.

Tabla # 2 Comparación de las herramientas de Caja Blanca.

Analizando el resultado del estudio de las herramientas, se llegó a la conclusión, de que las que no son de libre distribución ofrecen una serie de características adicionales que las hacen mucho más robustas, como son las generaciones de casos de prueba o los diferentes tipos de reportes que poseen, por citar algunos ejemplos. Aunque siempre tendrán como principal limitación su modo de adquisición.

1.8 Validación

La validación se refiere a la construcción de un modelo correcto. Es el proceso de determinar si el modelo, como abstracción, es una buena representación del sistema. Usualmente la validación se consigue a través de la calibración del modelo, en un proceso iterativo de comparación del comportamiento del modelo con el del sistema y usar las diferencias entre ambos para mejorar el modelo. Este proceso se repite hasta que el modelo se considera aceptable. Muchos otros autores tienen conceptos diferentes sobre validación entre los que se pueden encontrar:

- ✓ Roger S Pressman, según su criterio, la validación es el conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente.
- ✓ Determina si los requisitos y el sistema o software construidos al final cumplen con su uso proyectado.
- ✓ Es un proceso para determinar si los requisitos y el sistema o el producto de software, tal como se han construido, cumplen con su uso específico previsto.

1.8.1 Métodos de validación

Existe una variedad de métodos de validación los cuales brindan la posibilidad de validar una propuesta, dentro de ellos los más utilizados son los que se muestran a continuación [10]:

- ✓ **Comparación de los resultados de salida del modelo con los del sistema real**

  Este método se podrá aplicar en aquellos casos en los que el sistema exista y se pueda experimentar con él, de forma que se obtengan datos de salida del mismo. Este método consiste en ejecutar el modelo y obtener una serie de datos de salida y comparar éstos, mediante algún método estadístico, con resultados que se tengan del sistema. Debemos comparar dos conjuntos de datos, de alguna forma, para determinar si el modelo es una representación adecuada del sistema real. Una primera aproximación sería utilizar uno de los test estadísticos clásicos (como Chicuadrado o Kolmogorov-Smirnov para dos muestras) para determinar si las

distribuciones subyacentes a los dos conjuntos de datos se pueden ver como la misma.

- ✓ **Test de Turing**

  Alan Turing sugirió este método como un test de inteligencia artificial. En este test, a un experto, o grupo de expertos, se le presentan resúmenes o informes de resultados de ejecución del sistema y del modelo, a los que se les ha dado el mismo formato. Estos informes se reparten aleatoriamente a los ingenieros y administradores del sistema, para ver si son capaces de discernir cuáles son los reales del sistema y cuales la imitación resultado de la simulación. Si los expertos no son capaces de distinguir entre ambos, se puede concluir que no hay evidencias para considerar inadecuado al modelo. Si descubren diferencias las respuestas sobre lo que encuentran inconsistente se puede utilizar para realizar mejoras en el modelo.

  Se puede considerar que este método es el inverso al método de Delphi. En el test de Turing se consulta a los expertos para ver si son capaces de identificar las respuestas del sistema, mientras que en el de Delphi se pregunta a los expertos para que predigan las respuestas del sistema.

  Aunque este test parece muy intuitivo, hay muy pocos informes de su uso, ya que requiere un esfuerzo considerable para formatear las medidas de ejecución del sistema a la hora de crear el informe que se da a los expertos. Otra dificultad está en ajustar las medidas del sistema real ya que en ellas intervienen elementos que no se han considerado en el modelo. Por último este test requiere un análisis estadístico por parte del grupo de expertos para determinar si hay diferencias significativas entre el informe real y el simulado.

- ✓ **Validación de comportamientos en casos extremos**

  Ocasionalmente se puede observar el comportamiento del sistema bajo condiciones extremas.

  Esta es una situación ideal para recoger datos de las medidas de ejecución del sistema real de forma que luego se puedan comparar con los resultados de la simulación, una vez que se ejecute el modelo bajo situaciones similares. También es posible que los expertos del sistema puedan predecir el comportamiento del

sistema bajo condiciones extremas y utilizar estas predicciones para validar el modelo.

- ✓ **Validación Interna y Externa**

  La validación interna es un ejercicio teórico que asegura que la propuesta tiene una caracterización numérica apropiada de la propiedad que pretende medir. Esto es por supuesto un pre-requisito para demostrar la utilidad de dicha propuesta (validación empírica).

  La validación externa se lleva a cabo para demostrar con evidencia real que una propuesta es útil en el sentido de que está asociada con alguna característica externa del software tal como la usabilidad, la mantenibilidad, etc.

- ✓ **Método Delphi**

  Este método se utiliza cuando no se tienen datos o se dispone de muy pocos, acerca del sistema que se está considerando. En este método, se selecciona un grupo de expertos los cuales deben llegar a un consenso en las respuestas que den acerca de una serie de preguntas que se les plantean. En un entorno de simulación los expertos pueden ser los administradores y usuarios del sistema y las cuestiones son acerca del comportamiento del sistema bajo ciertas condiciones de operación. El método Delphi excluye las discusiones cara a cara entre los miembros del grupo.

  En este método se les plantea al grupo una serie de cuestiones

  - Se envía un cuestionario a cada miembro del grupo. Para la validación de un modelo de simulación, las cuestiones podrían tratar sobre las respuestas del sistema real ante ciertas entradas o cambios en su estructura.

  - Basándose en las respuestas dadas a las cuestiones planteadas, se elaboran nuevos cuestionarios que van centrándose en temas más específicos.

  - Los nuevos cuestionarios se envían al grupo junto con las respuestas obtenidas a las cuestiones en rondas anteriores.

Estos tres pasos se repiten hasta que el analista consiga de los expertos una predicción de la respuesta de sistema.

Una crítica de este método es que consume mucho tiempo. No necesariamente debe suponer un tiempo adicional en el proyecto de simulación, ya que se puede ir realizando paralelamente al desarrollo del modelo. Su costo puede resultar caro, pero no mucho más que otros métodos de validación. En general, aunque se critique que los métodos de validación son caros y consumen tiempo, más costoso es construir un modelo que no sea válido.

La calidad de los resultados depende, sobre todo, del cuidado que se ponga en la elaboración del cuestionario y en la elección de los expertos consultados. Siempre se comenzaría este proceso enviando un modelo a los posibles expertos con una explicación breve sobre los objetivos del trabajo y los resultados que se desean obtener.

El método de validación seleccionado fue el Delphi ya que existen la suficiente cantidad de expertos para validar la propuesta. Además, la validación no se enfoca solo en los resultados obtenidos en la aplicación de pruebas a los proyectos productivos, sino que la propuesta cumpla con todas las características y funcionalidades que se hayan hecho mención anteriormente.

1.9 Enfoque de la Investigación

La investigación está enfocada en el polo productivo de telecomunicaciones por lo tanto es de vital importancia el conocimiento de las características de los software que tienen que ver con la rama de las telecomunicaciones, ya que esto sería un punto importante a tener en cuenta durante el proceso de propuesta de las herramientas para la realización de las pruebas automáticas.

1.9.1 Características de los Software de Telecomunicaciones

Los Softwares producidos en el Polo de Telecomunicaciones tienen características particulares, sobre las cuales debemos de tener conocimiento a la hora de realizar una selección de herramienta para la realización de pruebas automáticas. Las siguientes características ayudan a realizar este proceso de selección.

- ✓ **Grandes**: Decenas o Centenares de líneas de código fuente

  Consecuencias:

  - El diseño lo debe abarcar un grupo numeroso de personas

  Necesidades:

  - Interfaces claras
  - Diferentes perfiles de diseñadores
  - Esfuerzo considerable de Gestión

- ✓ **Complejos**: Exhiben algún grado de concurrencia y están constituidos por un número elevado de módulos

  Consecuencias:

  - Necesidad de Técnicas de análisis y especificación formal en el proceso de descomposición modular.

  Necesidades:

  - Manejo de diferentes niveles de abstracción
  - Jerarquización (Con el objetivo de disminuir la complejidad observable)
  - Independencia entre módulos
  - Reutilización(El punto de partida es un desarrollo existente)

- ✓ **Larga Vida**: El tiempo de explotación es muy grande (15 a 25 años)

  Consecuencias:

  - Necesario controlar la evolución

  Necesidades:

  - El entorno cambiará durante ese periodo de tiempo y el sistema debe adaptarse
  - Mantenibilidad

- Modularidad
- Buena documentación

✓ **Intensivo en Datos:** Manejo de gran volumen de datos.

Consecuencias:

- Las técnicas de especificación y diseño deben permitir una rica variedad de definiciones de datos.

Necesidades:

- Disponibilidad de librerías
- Recuperación de módulos
- Soporte de cambios on-line

✓ **Encastrado**: Implantado sobre sistemas Hardware/Software especiales

Consecuencias:

- Implica el desarrollo hardware/software específico

Necesidades:

- Diseño independiente de la implementación
- Técnicas de especificación común hardware/software
- Herramientas de generación de código cruzado

1.10 Conclusiones

En el capítulo se han abordado los elementos teóricos sobre los cuales se sustentará la Propuesta de Herramienta para Pruebas Automáticas de Software. Se definen los conceptos generales de pruebas, sus tipos, su automatización, etc. De la investigación realizada se puede concluir que no existe una receta única para enfrentar esta tarea en la industria de software, su implementación está condicionada por diversos factores que van desde las capacidades de los recursos humanos hasta los recursos económicos disponibles.

Capítulo 2: Descripción de la Propuesta de Solución

2.1 Introducción

Una manera de optimizar un proceso de Pruebas de Software es la introducción de herramientas que ayudan a agilizarlo y hacerlo menos tedioso. Dada la existencia de un gran número de dichas herramientas el equipo de pruebas necesita criterios sobre los cuales establecer una comparación y así poder seleccionar el software de prueba que más se adapte a sus necesidades. En el presente capítulo se describe la forma de selección de dicho software teniendo en cuenta las características del software de telecomunicaciones mostradas en el epígrafe 1.9.1, las tecnologías utilizadas en el polo de telecomunicaciones y las ventajas y desventajas de las herramientas mostradas en la tabla del epígrafe 1.7.

2.2 Ventajas y desventajas del uso de herramientas para automatizar Pruebas de Software

Para comenzar el tema de las pruebas automatizadas es necesario conocer una serie de ventajas y desventajas que acarrea el uso de este recurso, con el objetivo de seleccionar o no diferentes alternativas y vías de solución a problemas que puedan existir. A continuación se verán algunas de ellas:

Ventajas:

- ✓ Rapidez en la ejecución de pruebas de regresión. En todas las metodologías de desarrollo de software el ciclo codificación-prueba-corrección se repite un número ilimitado de veces. La prueba de una funcionalidad específica tiene que ser ejecutada muchas veces durante el desarrollo, esto impide que ante la corrección de errores encontrados, otros no sean introducidos. Este proceso de repetición de pruebas es conocido como pruebas de regresión. Esta práctica es fundamental debido a las modificaciones que sufren los sistemas durante su elaboración. Si esta tarea se encuentra automatizada debido a su previa ejecución en versiones anteriores del producto, sólo se necesita seleccionarla y volverla a ejecutar con un mínimo esfuerzo manual.

- ✓ La ejecución de un mayor número de pruebas en una unidad de tiempo finita. Esta característica hace que el producto final sea mucho más confiable y por supuesto tenga mucha más calidad.

- ✓ Simulación de condiciones reales de explotación. Existen pruebas de vital importancia que no es factible realizarlas de forma manual. Tomando como ejemplo una prueba de estrés en la que se quiere simular el uso de una aplicación por 500 usuarios de manera concurrente. Como es de suponer es costoso reunir 500 computadoras y 500 personas para realizar esta labor, sin embargo existen herramientas con las que se simula esta situación.

- ✓ Programación de la ejecución de pruebas en horarios no laborales como la noche y los fines de semana. Muchas herramientas brindan prestaciones que permiten la programación automática de un sinnúmero de pruebas, utilizando así el horario de trabajo para idear posteriores planes de prueba.

Desventajas:

- ✓ Por lo general una gran cantidad de aplicaciones contienen elementos que no son compatibles con las herramientas que se utilizan para ponerlas a prueba, en consecuencia esto requiere la búsqueda de soluciones creativas para que éstas se adapten a la aplicación, lo cual constituye un obstáculo al que se tendrá que enfrentar el equipo de trabajo.

- ✓ Poco conocimiento de lenguajes de scripting por parte de los probadores. Éste puede ser un punto determinante en el proceso de automatización de pruebas pues una gran cantidad de herramientas utilizan estos lenguajes para el mantenimiento de las mismas.

- ✓ Si se automatiza el caos sólo se logra un caos mucho peor. No es recomendable la práctica de las pruebas automáticas por equipos que tengan mal organizada la realización de éstas. Ejemplos de malas prácticas son: poca o inconsistente documentación, pruebas que no son muy efectivas en la búsqueda de defectos, etcétera.

- ✓ Problema del mantenimiento. Cuando los sistemas sufren algún tipo de cambio, como ya se ha visto anteriormente, las pruebas también tienen que cambiar. El

esfuerzo empleado en la actualización de las mismas es a veces causa de abandono de la iniciativa de su automatización y de un retorno a la ejecución manual.

2.3 Proceso de Selección de Herramientas para Automatizar las Pruebas

En el proceso de selección de herramientas se evalúan cuáles son las apropiadas para las necesidades del equipo de trabajo. Para que el proceso tenga éxito no se puede comenzar la búsqueda centrándose en las herramientas que existen, lo primero y más importante, es identificar los requerimientos, características de los proyectos productivos, en este caso los enmarcados en polo de telecomunicaciones; metodología de desarrollo, tipos de prueba empleados, lenguajes de programación, framework de desarrollo que utilizan estos proyectos, los cuales fueron obtenidos a través de una entrevista realizada a los líderes de cada proyecto del Polo de Telecomunicaciones de la Facultad 2, así como las características de los software de este Polo y la disponibilidad de los distintos tipos de herramientas.

Para la realización de la selección de las herramientas se debe tener en cuenta también, que no se cuenta con el conocimiento de ninguna herramienta que sea capaz de realizar todos los tipos de pruebas, por lo que la selección propone el uso de estas en conjunto para la ejecución de las pruebas. Además se realiza especial énfasis en aquellas que sean software libre y gratis. Aunque nunca se debe dejar pasar por alto la calidad de las herramientas que no son de de libre distribución, ya que estas presentan una mayor bibliografía, brindan mayor facilidad para la realización de los pruebas e incluso algunas poseen la funcionalidad de creación de casos de prueba, lo que constituye un elemento que carecen las herramientas libres encontradas.

La mala elección de las herramientas tiene efectos indeseables que impactan de forma significativa en la realización de las pruebas, un ejemplo de esto son las dificultades técnicas que surgen a la hora de hacer funcionar la herramienta en el entorno, los usuarios encuentran dificultad en su uso, lo cual retrasa su trabajo lejos de hacerlo más ameno.

2.3.1 Resultados de las Entrevistas Realizadas a los Líderes de Proyectos del Polo de Telecomunicaciones

Por la necesidad de tener conocimiento de las características de los proyectos productivos del polo de telecomunicaciones para tener en cuenta estas características en el proceso de selección de las herramientas, así como, la necesidad de conocer el nivel de conocimiento y empleo de herramientas automáticas para Pruebas de Software se realizó una entrevista a los líderes de estos proyectos.

La entrevista constó de 6 preguntas previamente elaboradas (Ver Anexo # 1), en la primera se aborda acerca de la metodología de desarrollo que se aplica en el proyecto. En la segunda y tercera pregunta se hace alusión a cuáles pruebas aplican y desde que flujo de trabajo se empiezan a realizar, la cuarta pregunta se relaciona con el uso de pruebas automáticas conociendo de esta forma si son empleadas en el proyecto y cuales se han utilizado y por último en la quinta y sexta pregunta se trata el tema de qué tipo de aplicaciones se realizan en el proyecto el líder entrevistado, así como, el lenguaje de programación con que desarrollan el proyecto. De esta manera queda concebida la entrevista que permitió la obtención de la información necesaria para la realización de la propuesta de solución.

Las entrevistas realizadas a los diferentes líderes de proyectos del polo de telecomunicaciones arrojaron como resultado que en todos se realizan pruebas automáticas aunque no las mismas, en algunos casos es empleada la herramienta JUnit en el flujo de trabajo de implementación y en otros casos el JMeter durante el flujo de trabajo Prueba, de igual forma todos utilizan metodología RUP, así como, en la mayoría se programa en lenguaje Java para la realización de aplicaciones Web.

Teniendo en cuenta todo lo anteriormente planteado, se realizó una propuesta de herramientas que se espera sea utilizada por los líderes de proyecto para la realización de sus pruebas automáticas. La selección se realizó teniendo en cuenta las características que presenta el software de telecomunicaciones y a su vez los resultados de la entrevista realizada, por ser estos aspectos primordiales para la realización de la misma.

2.4 Propuesta de Herramientas

Al llevar a cabo la incorporación de herramientas para pruebas automatizadas en las etapas de desarrollo del proceso de software es necesario tener en cuenta todos los aspectos mencionados en los epígrafes anteriores, pues éstos constituyen la base para lograr este objetivo. Como conclusión de un estudio realizado en materia de herramientas para la automatización de pruebas, las características de los software del Polo de Telecomunicaciones y los resultados de la entrevista realizada a los líderes de los proyectos del mismo, se propone el uso de dos herramientas para la realización de pruebas automáticas, por las funcionalidades que brindan son las que mejor se adaptan a los entorno de desarrollo de los proyectos productivos del polo de telecomunicaciones.

Las dos herramientas seleccionadas pertenecen al software libre. La propuesta no comprende la automatización de las etapas de Requisitos y Análisis-Diseño, pues no se cuenta con herramientas que automaticen la revisión de los requisitos, ni el proceso de generación de casos de pruebas funcionales a partir de la descripción expandida de los casos de uso. Los siguientes epígrafes ilustran cómo se propone la automatización de las pruebas en las etapas de implementación y prueba.

2.4.1 Flujo de Trabajo de Implementación

En este Flujo de Trabajo las principales actividades relacionadas a las Pruebas de Software son: las pruebas de Caja Blanca. Este conjunto de pruebas permite la detección temprana de errores que con la aplicación de otros tipos de pruebas resultaría muy difícil, además se obtiene como resultado un código de mejor calidad, lo que sin dudas redundará, entre otras cosas, en un mejor mantenimiento de éste.

2.4.1.1 Pruebas de Unidad

Las pruebas de unidad se encargan de probar el código fuente; tienen como objetivo fundamental probar el funcionamiento de pequeñas unidades de código impidiendo que los errores se propaguen hacia el proceso de integración y otras etapas posteriores. Existe una gran variedad de herramientas para la automatización de pruebas de unidad, pero se hará especial énfasis en aquellas que son libres y de código abierto.

Basándose en los aspectos expuestos anteriormente, solo constituyen herramientas de caja blanca libres y de código abierto JUnit y TestNG. En algunos proyectos del polo de telecomunicaciones la herramienta que se usa es JUnit, que fue diseñado expresamente

para desarrollar pruebas de unidad. Las pruebas de Unidad son una parte importante de las pruebas, pero hay otros aspectos de las pruebas que con JUnit no se realizan de forma fácil, como la regresión, la integración, o las pruebas funcionales. Así pues, si bien es fácil probar con clases JUnit, cuando se trata de probar aplicaciones o sistemas enteros, este no ofrece una gran flexibilidad. Aunque sigue siendo la herramienta de pruebas de unidad más popular dentro del lenguaje JAVA, viene con una serie de limitaciones que imponen restricciones a los desarrolladores.

TestNG o Test Next Generation es una herramienta de código abierto diseñada para pruebas de unidad, aunque se puede utilizar para realizar Pruebas de Regresión, Integración, Pruebas de Tiempo de los Algoritmos, Funcionales. Fue creada para superar las principales limitaciones de JUnit y proporciona características adicionales necesarias para probar la última generación de aplicaciones desarrolladas en JAVA. TestNG también facilita la migración progresiva de JUnit. E incluso se pueden ejecutar una combinación de pruebas JUnit y TestNG.

Otra de las ventajas que presenta TestNG con respecto a sus predecesores es el agrupamiento de las pruebas, los métodos de prueba pueden pertenecer a uno o varios grupos. Estos grupos se utilizarán en tiempo de ejecución para determinar qué métodos deben ser invocados. Este recurso permite seleccionar cuáles se ejecutarán en un momento dado y cuáles no, sin necesidad de comentar el código. Además no hay necesidad de heredar de otra clase, ni implementar una interface para la creación de las pruebas y los métodos pueden ser nombrados de la forma que el usuario desee. Permite que los métodos que serán probados puedan contar con parámetros, siendo esta una de las principales desventajas de JUnit, y estos puedan ser obtenidos de forma dinámica u obtenidos de una base de datos, hoja de cálculo o cualquier otro archivo de almacenamiento. También existe una gran variedad de herramientas que hacen más fácil la conversión de clases de JUnit a TestNG. Cuenta además con los plugins en IDEs, como Eclipse, IntelliJ, IDEA, y NetBeans.

Para la automatización de las pruebas de unidad se ha seleccionado una herramienta libre de código abierto llamada **TestNG.**

¿Como diseñar casos de Pruebas de Unidad?

A la hora de diseñar los distintos casos de prueba para la prueba de unidad, específicamente, para una Prueba de Caja Blanca se pueden crear de distintas formas, en esta investigación se explicará el Método de Camino Básico. El Método del Camino Básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Antes de considerar el método de camino básico se debe introducir una sencilla notación para la representación del flujo de control, denominada grafo de flujos y una métrica para la obtención de la complejidad ciclomática.

El grafo de flujos representa el flujo de control lógico mediante la notación ilustrada Fig. # 5.

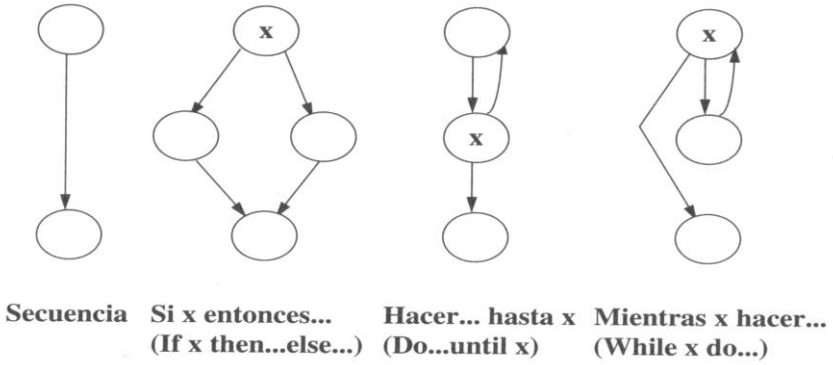


Fig. # 5 Notación del grafo de flujo.

Cada círculo del grafo de flujo es denominado nodo y corresponde a una o más sentencias sin estructuras condicionales o ciclos y las flechas que unen los nodos, están definidas como aristas, que son las que siguen la lógica del método que se analiza y deben siempre terminar en un nodo, incluso si éste no representa ninguna sentencia procedimental.

Para la creación del grafo a partir del código fuente de una función o método del programa resulta del cambio de las estructuras representadas en la Fig. # 5 por sus homólogos en el código, o sea, todas las acciones que no incluyan condicionales y ciclos serán

colocadas en los nodos, posteriormente se colocan las aristas uniendo cada nodo según indica la lógica del programa.

La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto del método de prueba del camino básico, el valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición. En términos del grafo de flujo, un camino independiente está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino. (Ver Fig. # 5)

La complejidad se puede calcular de tres formas:

- ✓ El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
- ✓ La complejidad ciclomática, V (G), de un grafo de flujo G se define como V (G)=A-N+2 donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.
- ✓ La complejidad ciclomática, V (G), de un grafo de flujo G también se define como V (G) =P+ 1 donde P es el número de nodos predicado contenidos en el grafo de flujo G. Nodo predicado: Está caracterizado porque dos o más aristas que emergen de él.

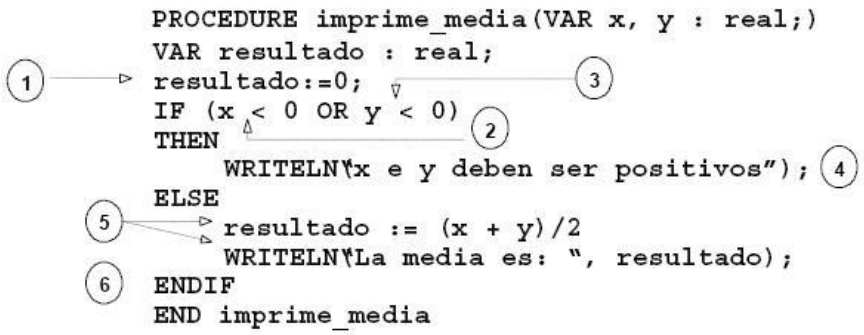
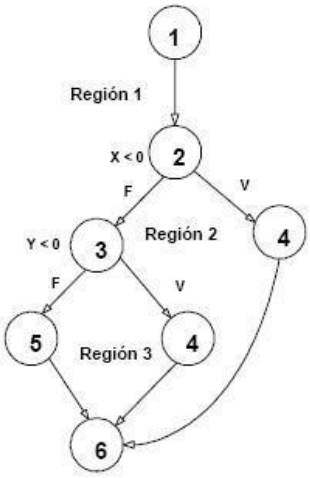


Fig. # 6 Método para calcular el grafo de flujos



V(G) = 2+1 = 3. Por lo tanto, hay que determinar tres caminos independientes.
Por ejemplo:

Camino 1: 1-2-3-5-6
Camino 2: 1-2-4-6
Camino 3: 1-2-3-4-6

Fig. # 7 Ejemplo de grafo de flujo

Conociendo estos conceptos ya se puede proceder a la creación de los casos de prueba, que se puede dividir en los siguientes pasos:

- ✓ Usando el diseño o el código como base, se dibuja el correspondiente grafo de flujo.
- ✓ Se determina la complejidad ciclomática del grafo de flujo resultante.
- ✓ Recorrer el grafo de flujo para encontrar los caminos independientes de la función, hasta lograr que su número sea igual a la complejidad ciclomática y además que cada camino independiente cumpla con la definición antes planteada.
- ✓ Se preparan los casos de prueba que forzarán la ejecución de cada camino del conjunto básico.

2.4.1.2 Utilización de TestNG para Pruebas de Unidad

Para usar **TestNG** se adiciona el plugins al IDE con se va a utilizar y posteriormente solo hay que añadir el .jar, que se encuentra disponible en el sitio www.**testng**.org, en el camino donde se encuentran las clases (classpath) del proyecto que se desea probar y luego comenzar a implementar los caso de prueba.

Para mostrar cómo se realiza el uso de esta herramienta se utilizó como ejemplo el fragmento de código correspondiente a la clase “Ordenar” y dentro de ella se procederá a la comprobación del método “Intercalar”.

```
    public int [] Intercalar( int [] a1, int [] a2)
    {
/*1*/     int i = 0;
/*1*/     int j = 0;
/*1*/     int k = 0;
/*1*/     int [] a3 = new int [a1.length + a2.length];
/*2,3*/   while (( i < a1.length ) && ( j < a2.length ) )
        {
/*4*/         if (a1[i] < a2[j])
/*5*/             a3[k++] = a1[i++];
/*6*/         else a3[k++] = a2[j++];
         }
/*7*/    if (j == a2.length) //¿El 2° array no tiene datos?
/*8*/         while (i < a1.length)
/*9*/             a3[k++] = a1[i++];
/*10*/   else  while (j < a2.length)
/*11*/            a3[k++] = a2[j++];
/*12*/   return a3;
      }
```

Fig. # 8 Método Intercalar.

Ahora se aplica los pasos definidos en el epígrafe anterior para determinar la realización de los casos de prueba de unidad y se construye el grafo pertinente.

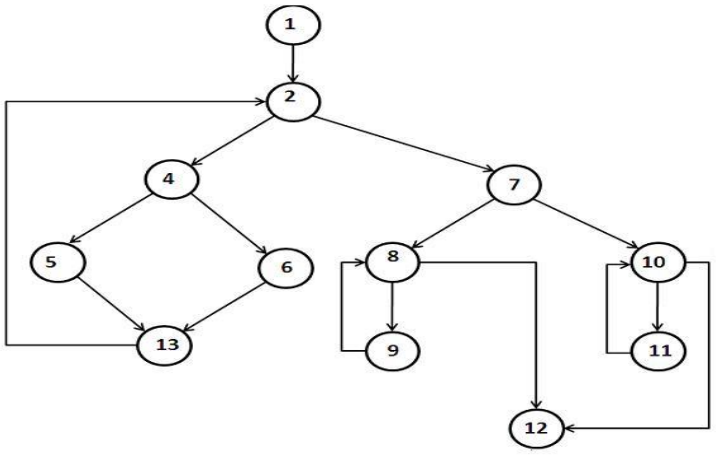


Fig. # 9 Grafo de Flujos

Posteriormente se calcula la complejidad ciclomática de dicho grafo, de cualquiera de las 3 formas antes explicadas, con ello se obtiene la cantidad de caminos independientes existentes en el grafo.

Cálculo de la complejidad ciclomática	Caminos Encontrados

V (G)=A-N+2. A = número de aristas (16). N = numero de nodos (12). V (G)= 16 – 12 + 2. V (G) = 6.	✓ **1, 2, 4, 5, 13, 2, 7, 8, 9, 8, 12.** ✓ **1, 2, 4, 6, 13, 2, 7, 8, 9, 8, 12.** ✓ **1, 2, 4, 5, 13, 2, 7, 10, 11, 10, 12.** ✓ **1, 2, 4, 6, 13, 2, 7, 10, 11, 10, 12.** ✓ **1, 2, 7, 8, 9, 8, 12.** ✓ **1, 2, 7, 10, 11, 10, 12.**

Tabla # 3 Descripción del cálculo de complejidad ciclomática

Una vez que se han hallado los caminos necesarios a ejecutar se deben buscar los datos correctos para garantizar dicho recorrido y posteriormente implementar una clase Java que contenga los casos de pruebas con sus datos requeridos.

Los conceptos utilizados en esta documentación para la definición de clases y métodos son los siguientes:

- ✓ Una suite está representado por un archivo XML. Puede contener una o más pruebas y está definida por la etiqueta <suite>.
- ✓ Una prueba está representada por la etiqueta <test> dentro de la suite y puede contener una o más clases TestNG.
- ✓ Una clase TestNG es una clase Java que contiene al menos una anotación TestNG. Es representada por la etiqueta <class> dentro de la suite y puede contener uno o más métodos de ensayo.
- ✓ Un método de prueba es un método Java anotado por @Test en su fuente.

La clase ComprobarUnit1 (Ver Fig. # 10) está compuesta por el método setUp (), es el cargado de inicializar el objeto de tipo Unit1, que será al cual se le aplicará la prueba. Para garantizar que este método sea invocado antes que cualquier método de prueba de la clase se le coloca la anotación @BeforeTest. Las anotaciones se declaran, como se ve antes, utilizando el carácter “@” y detrás el nombre de la anotación, ellas se ocupan de la configuración y la organización de la clase de prueba.

```
public class ComprobarUnit1
{
    private Unit1 objeto;

    @BeforeTest
     public void setUp()
     {
       // code that will be invoked when this test is instantiated
        objeto = new Unit1();

     }
    @Test(groups = { "fast" },dataProvider = "test1")
     public void ProbarIntercalar( int [] a1, int [] a2)
     {
        int [] respuesta = objeto.Intercalar(a1, a2);
        for (int i = 0; i < respuesta.length; i++)
        {
            Assert.assertEquals(respuesta[i], i+1);
        }
```

Fig. # 10 Contenido de la clase TestNG para las pruebas.

Además se expone el código del método de prueba ProbarIntercalar. En su implementación se realiza una llamada al método que se necesita probar, Intercalar, y posteriormente se comprueba el resultado del mismo usando la función Assert definida por el framework TestNG. Se tiene la anotación @Test, utilizada para identificar los métodos de prueba, a esta anotación se le definen algunos parámetros, como “groups” que es utilizado para agrupar los métodos de pruebas en función de las necesidades del probador y “dataprovider” que es de donde obtendrá los parámetros para su ejecución.

Para la obtención de los datos de prueba que cubran los distintos caminos definidos anteriormente se usa un método con la anotación @DataProvider. Este devuelve un arreglo de arreglos de objetos que contienen los parámetros, en caso que devuelva más de un juego de parámetros, el método se ejecuta nuevamente. En el caso de la Fig. # 11 se devuelven 4 juegos de datos para la ejecución del método ProbarIntercalar.

```
@DataProvider(name = "test1")
public Object[][] createData1()
{
    return new Object[][] {
          { new int[]{2,3,4},new int[]{1}},
          { new int[]{1},new int[]{2,3,4}},
          { new int[]{1,2,3,4},new int[]{}},
          { new int[]{},new int[]{1,2,3,4}}
          };
}
```

Fig. # 11 Ejemplo de la creación de parámetros

Antes de poder ejecutar las pruebas, sin embargo, se debe configurar TestNG utilizando un archivo XML. La sintaxis de este archivo es muy simple, y se presenta en la Fig. # 12. Este archivo comienza por la definición de la suite,"Corte3", compuesto por una única prueba, "PruebaUnidad", que esta implementada en la clase "ComprobarUnit1".

```
<suite name="Corte3">
  <test verbose="2" name="PruebaUnidad" annotations="JDK">
    <classes>
      <class name="Clases.ComprobarUnit1"/>
    </classes>
  </test>
</suite>
```

Fig. # 12 Ejemplo de una suite de prueba.

Una vez que se han creado clases que contienen anotaciones TestNG (y / o) una o más suite (archivos .XML), se debe crear la configuración de lanzamiento TestNG. Se selecciona el comando Run / Open Run Dialog... y se busca dentro de la opción TestNG una nueva configuración, luego se selecciona el proyecto donde se encuentra la prueba y se selecciona la opción suite y se coloca la dirección donde se encuentra la suite. Solo resta aplicar los cambios y correr la prueba. (Ver Fig. # 13 y 14)

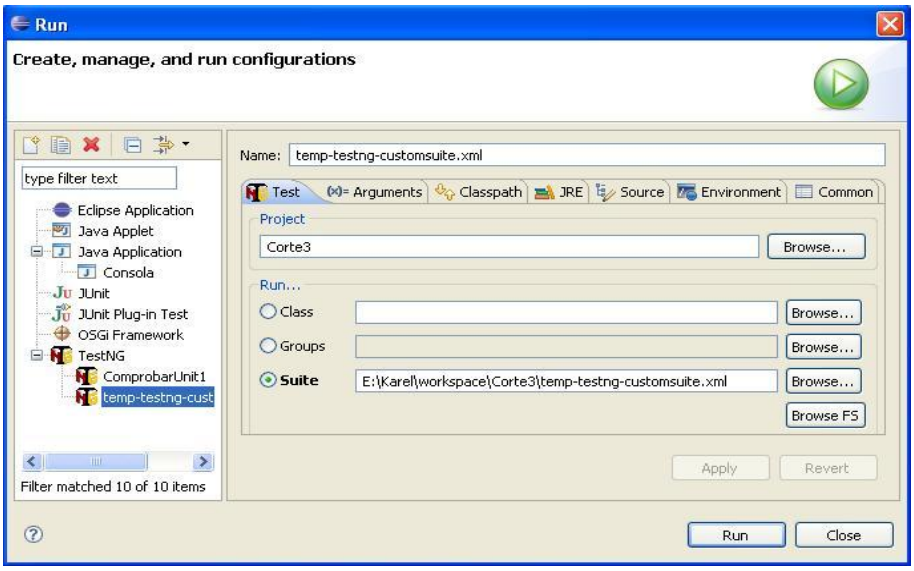


Fig. # 13 Configuración del lanzamiento de la suite desde eclipse.

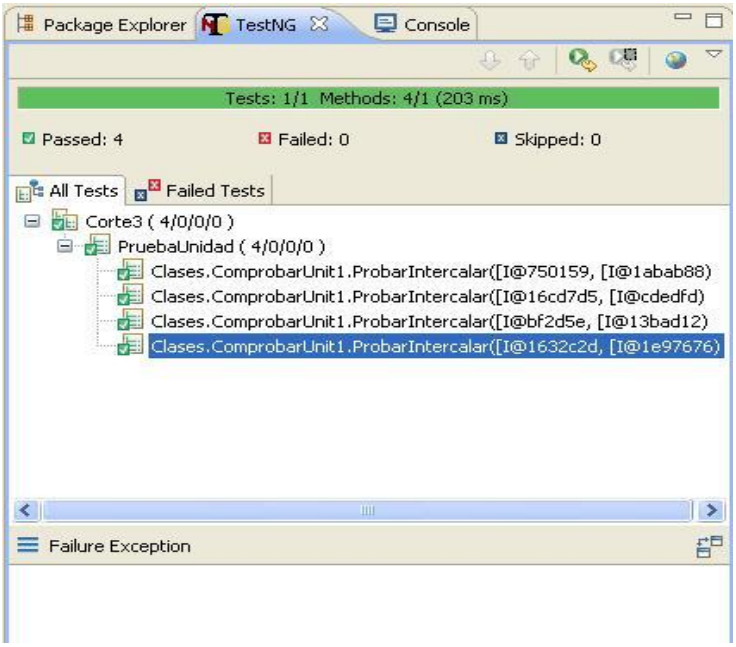


Fig. # 14 Resultado de ejecutar la prueba.

Test	Methods Passed	Scenarios Passed	# skipped	# failed	Total Time	Included Groups	Excluded Groups
Clases.ComprobarUnit1	1	4	0	0	0.1 seconds		

Class	Method	# of Scenarios	Time (Msecs)
Clases.ComprobarUnit1 — passed			
Clases.ComprobarUnit1	ProbarIntercalar (fast)	4	32

Fig. # 15 Ejemplo de reporte HTML

2.4.2 Flujo de Trabajo de Prueba

En el epígrafe 1.4.1 del presente trabajo se exponen las actividades que se realizan durante el flujo de trabajo de prueba, así como los trabajadores que intervienen en este flujo y los artefactos que se generan. Seguidamente se exponen la herramienta propuesta para automatizar las Pruebas Funcionales, de Sistema, Volumen y Regresión.

2.4.2.1 Pruebas Funcionales

Durante el proceso de investigación y búsqueda de herramientas se obtuvo como resultado que la mayoría de ellas son de software propietario. Siendo JMeter y

PushToTest TestMaker las aplicaciones software libre y de libre distribución más completa encontradas.

JMeter es una herramienta creada 100% en JAVA, que permite realizar Pruebas Funcionales, Carga, Estrés, sobre aplicaciones web. Además, con ella se pueden realizar test a servidores FTP, JDBC, JNDI, LDAP, SOAP/XML-RPC. Adicionalmente, JMeter puede realizar Pruebas de Regresión a los diferentes sistemas y presenta una gran variedad de informes con los resultados de los test.

PushToTest TestMaker es una plataforma que su funcionamiento se basa en la utilización de los diferentes test de unidad para rehusarlos como Pruebas de Integración, Funcionales, Carga, Estrés, Volumen, Regresión y posee un servicio de vigilancia que se encarga de monitorear el funcionamiento de la aplicación web cada un tiempo determinado por el usuario. Posee una gran variedad de gráficas para la visualización de los resultados de los test, así como un monitoreo del funcionamiento de los recursos (RED, CPU, Memoria) de los nodos que realizan la prueba. Las mismas pueden ser repartidas en diferentes nodos de ejecución y así lograr un escenario más real donde probar las aplicaciones web.

También puede ser usado en conjunto de otros lenguajes muy utilizados en la actualidad como son: Java, .NET, Jython, Groovy, PHP, Ruby para la construcción de los casos de pruebas. Además soporta Servicio Orientado a Arquitectura (SOA), AJAX, Web 2.0, y servicios REST usando sus protocolos nativos (HTTP, HTTPS, SOAP, XML-RPC, y los protocolos de correo, SMTP, POP e IMAP). Los datos para la realización de las pruebas pueden ser obtenidos a través de bases de datos o archivos de texto.

Esta herramienta cuenta con asistentes (wizards), como el soapUI 2.0.2 que es usado para la construcción de pruebas funcionales y de carga a los distintos servicios web que hayan sido creados para ser usados por las aplicaciones web. Además cuenta con grabadores (recorders), como el TestGen4Web o el Selenium, que facilitan la construcción de los diferentes casos pruebas, aún cuando no se cuenten con ninguna experiencia como programadores o aseguradores de calidad. Cuando se instala la aplicación posee un servidor web con el cual se pueden dar los primeros pasos dentro del mundo de las pruebas. http://localhost:8080/BrewBizWeb/login.html; este solo puede ser usado con la aplicación **PushToTest TestMaker** en ejecución.

La propuesta de herramienta elegida para esta investigación fue **PushToTest TestMaker.**

2.4.2.1.1 Instalación de PushToTest TestMaker.

Para realizar la instalación de esta aplicación es preciso tener instalado previamente en la computadora la versión 1.6 o mayor de Java Runtime Enviroment (JRE), contar con Mozilla Firefox 2 o mayor ya que es el encargado del soporte de los grabadores y para realizar la instalación del software de prueba se debe ejecutar los archivos:

- ✓ **TestMaker_Install**: este fichero instala el entorno de PushToTest TestMaker. Por defecto también instala un TestNode local para operar las pruebas desde la misma máquina, un Monitor, también local para los recursos del sistema y los plugins para Firefox Selenium y TestGen4Web.
- ✓ **TestNode_Install**: este fichero instala un TestNode, que es el entorno remoto que recibe comandos desde el TestMaker y ejecuta una prueba.
- ✓ **Monitor_Install**: este fichero instala el PushToTest Monitor para observar el CPU, la red y la utilización de la memoria cuando una prueba se está ejecutando.

La carpeta de instalación de PushToTest TestMaker está compuesta por los siguientes archivos:

- ✓ **example_agents**: posee una gran cantidad de ejemplos demostrando las potencialidad de la herramienta
- ✓ **docs**: contiene documentación sobre el uso de los ejemplos encontrados en la carpeta example_agents y muestra entre otros temas como debe ser analizados lo resultados de cada prueba, la metodología para el uso y las preguntas más frecuentes.
- ✓ **lib**: contiene librerías de código que soportan las operaciones de la herramienta.
- ✓ **license.txt**: es la licencia para la herramienta.
- ✓ **TestMaker.bat**: inicia la herramienta para el sistema Windows.
- ✓ **TestMaker.sh**: inicia la herramienta para los sistemas UNIX, Linux, y Mac OS X.

- ✓ **QuickStart.html**: revela las notas asociadas a la herramienta, cambios de último minuto y adiciones que pudiera tener la misma.

Luego se añade una variable del sistema (“Mi PC -> clic derecho->propiedades->Variables de entorno-> Nueva variable del sistema”) con el nombre de “JAVA_HOME” y su valor es la dirección de donde se encuentra la maquina virtual de java o jre. Solo falta ejecutar el archivo TestMaker.bat.

Utilización de PushToTes TestMaker para Pruebas Funcionales

Para la ejecución de la aplicación bastaría con ejecutar el archivo TestMaker.bat si nos encontramos usando el Sistema Operático Windows o el TestMaker.sh si usamos alguna distribución de Linux, Unix y Mac Os X, esto mostraría la interfaz ilustrada en la Fig. # 16.

La interfaz cuenta con 5 áreas principales:

- ✓ Controles: que no son más que lo botones que son usados para la ejecución de cada uno de los escenarios de prueba.
- ✓ Output: Consola que es la encargada de mostrar todos lo sucesos que ocurren durante el uso de la aplicación.
- ✓ Buffer: Muestra todos aquellos archivos que han sido abiertos recientemente, así como, las direcciones de carpetas usadas.
- ✓ Location: Muestra la dirección de un archivo o carpeta.
- ✓ Editor: Región de color amarillo que muestra el contenido de las carpetas o ficheros cuya dirección sea colocada en Location.

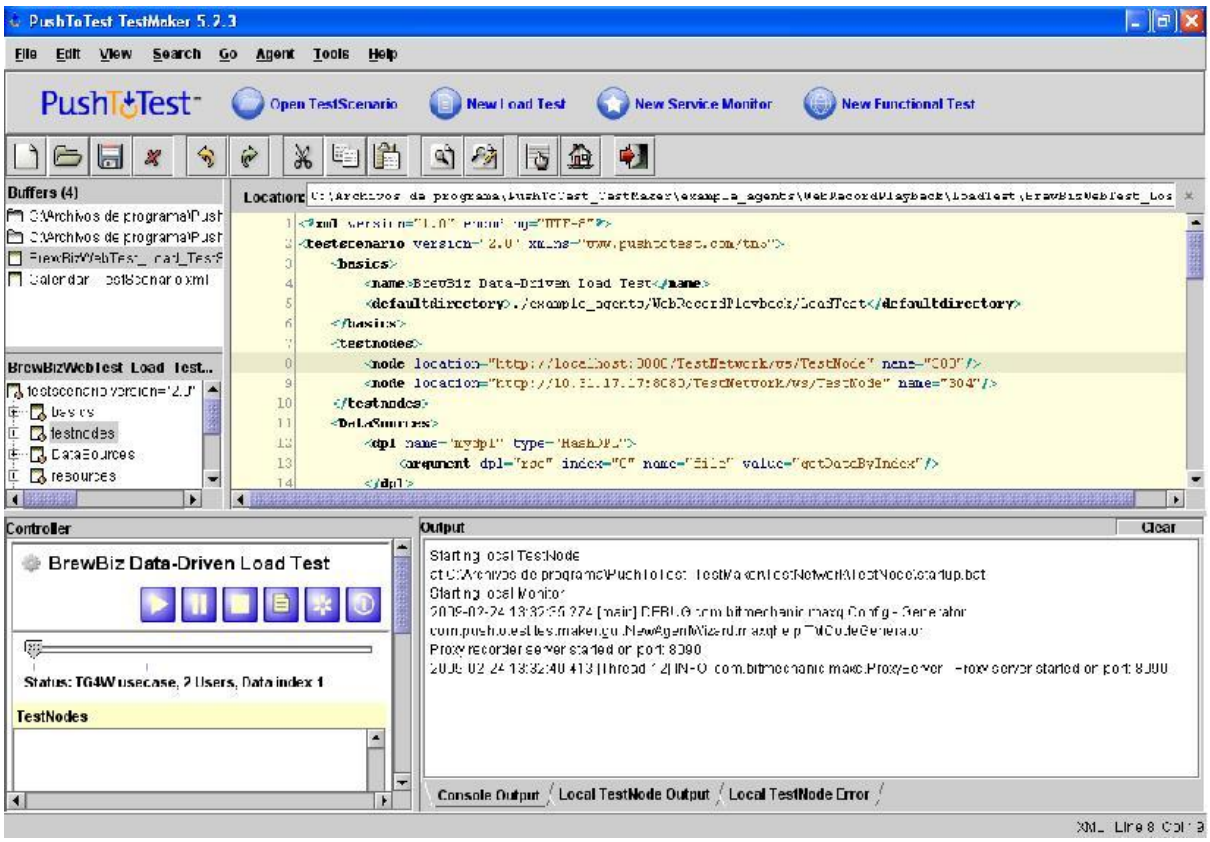


Fig. # 16 Interfaz de la herramienta PushToTest TestMaker.

En esta investigación, debido a la similitud en el uso y a las funcionalidades que brindan, el TestGen4Web y el Selenium, solamente nos referiremos al primero de estos. Selenium es considerada mucho más potente que el TestGen4Web, debido a que soporta el trabajo con aplicaciones que han sido desarrolladas usando Ajax. Para ejemplificar el uso de estas herramientas usaremos la aplicación que brinda por defecto cuando es instalado el PushToTest TestMaker, BrewBizWeb. BrewBiz Inc. es un fabricante de cerveza, que ofrece a sus clientes una aplicación web (BrewBizWeb) que permite el login de sus usuarios, la búsqueda de producto y otras funcionalidades.

El caso de prueba usado para la ejemplificación sería acceder a la página web para confirmar que los productos que serán vendidos se encuentran dentro del inventario. Primero es necesario que el usuario realice el login, luego de ser correcto buscaría por el identificador (un número) el producto y la aplicación le mostraría una lista con todos los productos que poseen ese identificador y los precios de los mismos. Luego se asegurará de que los identificadores mostrados son lo que se habían pedido en realidad y luego se cerraría la sesión del usuario. Un ejemplo muy sencillo y fácil, pero muy explicativo para

comprender la lógica del funcionamiento tanto del PushToTest TestMaker y del asistente TestGen4Web.

Iniciamos la aplicación usando como navegador web Mozilla Firefox, es necesario porque el asistente TestGen4Web está diseñado para este navegador como se explicó anteriormente. Una vez inicializada la aplicación se usa el TestGen4Web como grabador que muestra la siguiente interfaz.

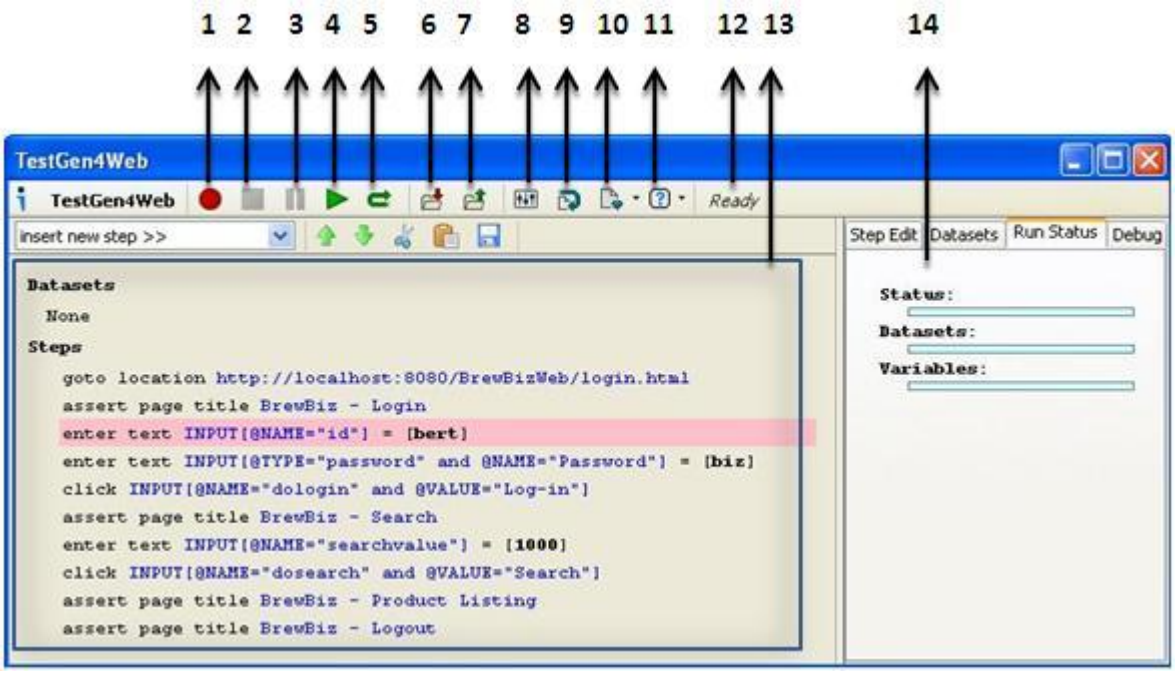


Fig. # 17 Interfaz de la herramienta TestGen4Web desde Mozilla

Descripción de la herramienta TestGen4Web	
1-Recorder: Comenzar la grabación	2-Stop: Finalizar la grabación
3-Pause: Pausar la grabación	4-Play: Ejecutar la grabación
5-Loops: Cantidad de repeticiones	6-Save: Salvar la grabación
7-Open: Abrir la grabación existente	8-Preferencias
9- Reset: Reiniciar la grabación realizada	10-Abrir grabaciones usadas recientemente

11-Ayuda	12-Mensaje que varía durante la ejecución de la prueba
13-Editor: Donde se muestra el código de la prueba	14-Tabs que muestran la conexión de la Base de datos (si existe), opciones para modificar algún paso, estado de ejecución de la prueba y permite debuguear código JavaScript

Tabla # 4 Descripción de la herramienta TestGen4Web.

Luego de obtener la grabación, dentro del editor del TestGen4Web (marcado con el número 13), se salva con el nombre Ejemplo.testgen4web para el uso posterior con los escenarios de PushToTest TestMaker.

Los escenarios para pruebas que propone PushToTest TestMaker son archivos de extensión "XML". El contenido se encuentra estructurado por etiquetas. Todo el contenido debe encontrase embebido dentro de la etiqueta **"<testscenario>"**,y dentro de ella se encuentran otras como **"<basics>"**, **"<testnodes>"**, **"<resources>"**, **"<test>"** y otras que serán explicadas a medida que sean usadas.

Para la creación de los escenarios de prueba primeramente debe ser modificado dentro de la etiqueta **"<testscenario>",** la etiqueta **"<basics>"** que es la encargada de definir tanto el nombre de la prueba como la dirección de donde se encuentran los archivos que serán usados en esta.

```
<basics>
    <name>Ejemplo de Prueba Funcional</name>
    <defaultdirectory>D:\Ejemplo_Prueba_Funcional</defaultdirectory>
</basics>
```

Fig. # 18 Muestra la configuración de la etiqueta basics.

Luego se configura la etiqueta **"<testnodes>"**, que es la encargada de definir desde donde aplicaremos la prueba y el nombre que se le pondrá al nodo.

```
<testnodes>
    <node location="http://localhost:8080/TestNetwork/ws/TestNode" name="localhost"/>
</testnodes>
```

Fig. # 19 Muestra la configuración de etiqueta TestNode

Posteriormente se modifica la etiqueta **"<resources>"**, que identifica los archivos que serán cargados de forma dinámica por PushToTest TestMaker.

```
<resources>
    <testgen4web path="Ejemplo.testgen4web"/>
</resources>
```

Fig. # 20 Muestra la configuración de etiqueta Resources

Por último la etiqueta **"<test>"**, que dentro de ella la etiqueta **"<run>"** cuya sentencia ejecuta cada fichero grabado; en su propiedad **name** se define el nombre de la prueba, en **testclass** el nombre del fichero con extensión **"testgen4web"**, **method** y **langtype** se llenan de la forma ilustrada.

```
<test>
    <run langtype="testgen4web"
        method="TestGen4Web" name="testweb" testclass="Ejemplo.testgen4web"/>
</test>
```

Fig. # 21 Muestra la configuración de etiqueta Test.

Posteriormente se salva la configuración del escenario y se ejecuta la prueba pulsando el botón Run del controlador.

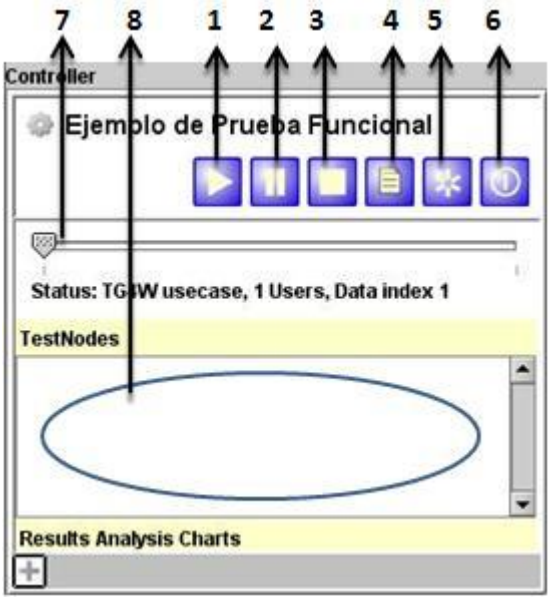


Fig. # 22 Ambiente del controlador.

Descripción del controlador.

1-Run: Ejecuta la prueba.	2-Pause: Detiene momentáneamente la ejecución de la prueba.
3-Stop: Detiene la prueba.	4-Edit: Permite editar el fichero XML.
5-Expand: Maximiza o minimiza el controlador.	6-Close: Cierra el controlador.
7-Status: Barra de estado de la prueba.	8- Muestra los nodos que están usando en la prueba.

Tabla # 5 Descripción del Controlador.

Al finalizar la prueba se maximiza la ventana del controlador (usando el botón 5) y este muestra los resultados.

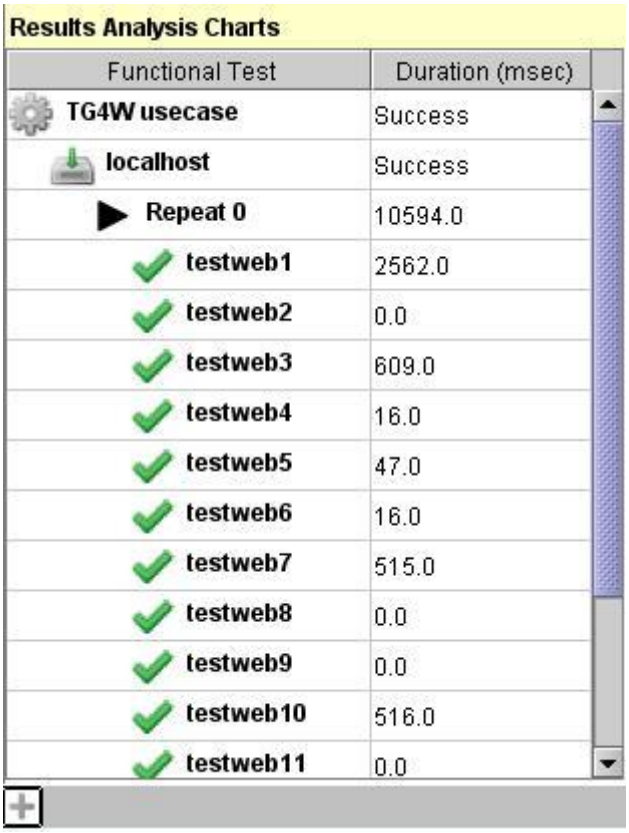


Fig. # 23 Resultados de la prueba funcional

El resultado de la prueba está compuesto por la comprobación, en cada uno de los nodos utilizados, de cada uno de los pasos grabados previamente con el TestGen4Web,

representados por testweb1, testweb2 y así sucesivamente y el tiempo que demoró en ejecutar cada uno de ellos en milisegundos. De no poder realizar algún paso lo muestra en color rojo intenso.

PushToTest TestMaker también cuenta con la posibilidad de ampliar aún más el análisis de las aplicaciones web, ya que posee Librerías de Producción de Datos (Data Production Library [DPL]) que son las encargadas de leer datos de archivos de valores separados por comas (Comma Separated Value [*.csv]). Si en el ejemplo anterior sobre la realización de pruebas funcionales solo se podían realizar esos test usando lo valores establecidos por la grabación; ahora se tiene la posibilidad que esos valores sean obtenidos de un archivo de extensión *.csv. Por lo que permitiría realizar la prueba siempre con juegos de datos diferentes y así potenciar más la calidad de las pruebas.

Para hacer uso de esta técnica primeramente se debe crear un archivo de valores separados por coma. Este puede ser creado en cualquier editor de texto u hoja de cálculo, donde la primera línea contiene el nombre de las columnas que será usada para mapear los valores dentro de la grabación. El archivo que será nombrado como **data.csv** quedaría de la siguiente forma:

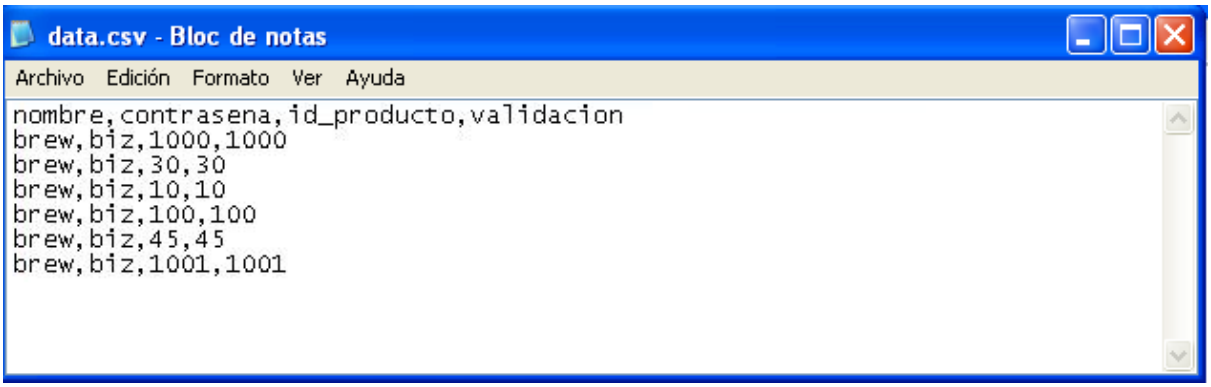


Fig. # 24 Estructura del archivo data.csv usando el Bloc de Notas de Windows

Luego se debe modificar algunos pasos dentro de la grabación para que esta sea capaz de vincularse con los valores de data.csv. Para modificar bastaría con seleccionar el paso donde se encuentra una entrada de datos y este valor sería remplazado por el nombre de la columna a que se quiere que esté asociada (Ver Fig. #25).

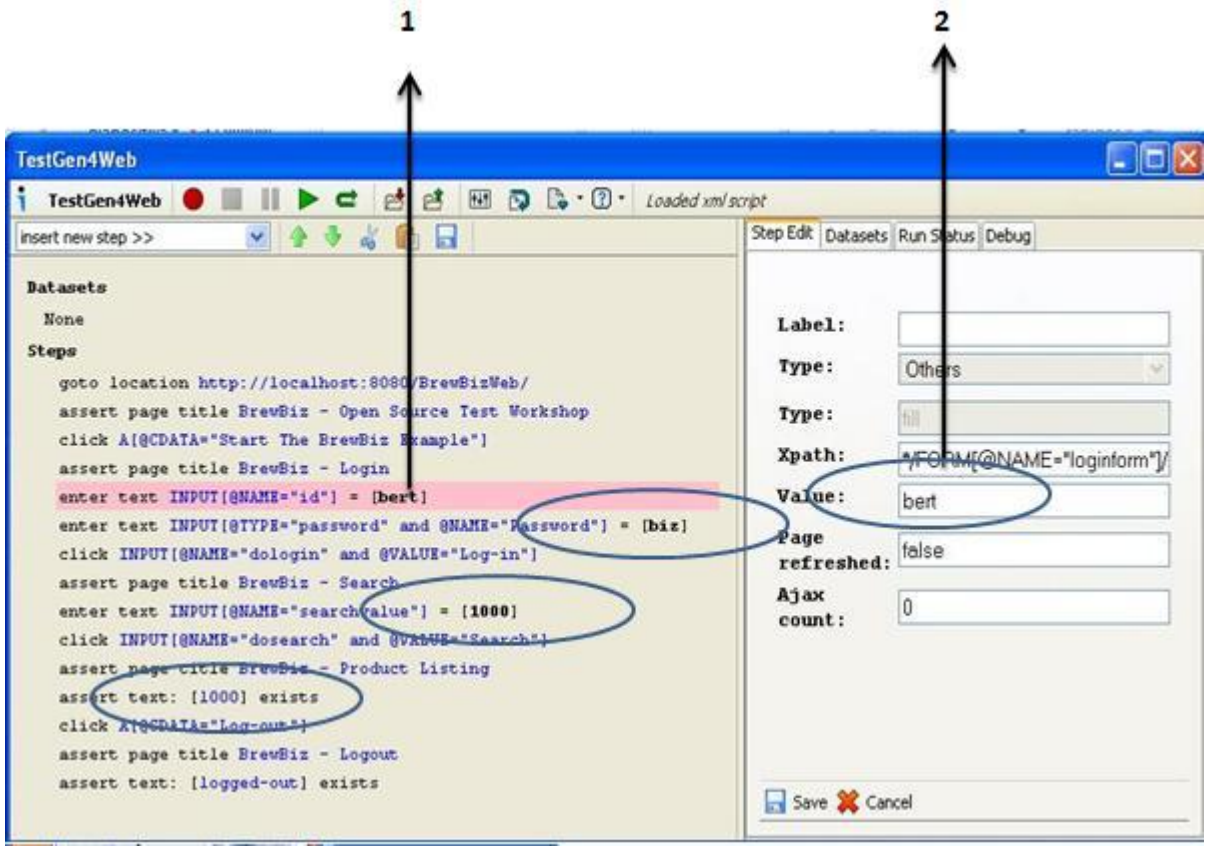


Fig. # 25 Muestra como debe ser modificada la grabación.

1-Se selecciona el paso de la grabación que se desea modificar (se muestra en color rosa).	2- En la pestaña “Step Edit” se cambia el valor por el nombre de la columna a la cual se desea vincular. En este caso en específico se cambia el Value: “bert” por “nombre”.

De manera similar se cambiarán los demás valores encerrados en óvalos (Ver Fig. # 25) por sus columnas contrasena, id_producto, y validación en el mismo orden en que aparecen.

Posteriormente se edita como se hizo anteriormente el escenario de pruebas de **PushToTest TestMaker** y modificamos la etiqueta **“<DataSources>”** que se encarga de comunicarle a **PushToTest TestMaker** que va a trabajar con un nuevo DPL para la realización de pruebas de forma dinámica.

```
<DataSources>
  <dpl name="mydpl" type="HashDPL">
    <argument name="file" dpl="rsc" value="getDataByIndex" index="0"/>
  </dpl>
</DataSources>
```

Fig. # 26 Muestra la configuración de etiqueta DataSource.

Se añade la dirección de **data.csv** a la etiqueta"**<resources>"**:

```
<resources>
    <data path="data.csv"/>
    <testgen4web path="Ejemplo.testgen4web"/>
</resources>
```

Fig. # 27 Muestra la configuración de etiqueta resources.

Se cambia el contenido de la etiqueta **"<test>"** por:

```
<run name="testweb" testclass="Ejemplo.testgen4web" method="testgen4web" langtype="testgen4web">
  <argument dpl="mydpl" name="DPL_Properties" value="getData"/>
</run>
```

Fig. # 28 Muestra la configuración de etiqueta test.

Posteriormente dentro de la etiqueta **"<functionaltest repeat="1">"**, se establece un nuevo número de repeticiones, casi siempre mayor o igual que la cantidad de juegos de datos que contiene **data.csv;** esto posibilitaría que se ejecutara al menos una vez por cada juego de datos. Para este ejemplo se tomaran 6 repeticiones. Se salva el editor y se ejecuta la prueba.

Con esto culmina esta segunda variante de ejecución de pruebas funcionales a las aplicaciones web. La metodología de realización de las mismas usando Selenium sería prácticamente igual, cambiaría solamente en que habría que usar una grabación de dicha herramienta, las extensiones *.testgen4web serían cambiados por *.selenium y la forma en que PushToTest TestMaker reconoce como trabajar con TestGen4Web deben ser cambiadas por las que identifican a Selenium.

2.4.2.2 Pruebas No Funcionales

2.4.2.2.1 Pruebas de Carga

PustToTest TestMaker ejecutó en el ejemplo anterior la prueba funcional simulando un solo usuario. Sin embargo esta podría convertirse en una prueba de carga si se ejecuta simulando más de un usuario realizando la prueba a la vez.

2.4.2.2.2 Utilización de PustToTest TestMaker para Pruebas de Carga

Para la creación de un escenario de pruebas de carga se selecciona en la barra de botones la opción Nueva Prueba de Carga (**"**New Load Test**"**), este brinda la posibilidad de guardar el escenario, que al igual que el escenario visto anteriormente es de extensión **.**XML. Las diferencias que existen entre el escenario de pruebas de carga y los escenarios de pruebas funcionales son dentro del archivo **.**XML**,** ya que las grabaciones pueden ser usadas por ambos tipos de pruebas. Además la interfaz del controlador posee los mismos botones, pero en su parte inferior muestran las gráficas (**"**Charts**"**) con el resultado de las pruebas.

Dentro del escenario de pruebas de carga, la configuración de algunas etiquetas es la misma al ejemplo mostrado en las pruebas funcionales, las etiquetas son: **"<basics>"**, **"<testnodes>"**, **"<resources>"**, **"<test>"** y **"<DataSource>".** Ahora dentro de la etiqueta **"<crlevels>"** se debe definir la cantidad de usuarios que van a ser simulados al ejecutar la prueba o sea, en este caso especifico, Fig. # 29, PustToTest TestMaker ejecutará la prueba para 3 niveles de concurrencia de usuarios distintos, para 2, 4 y 8 usuarios:

```
<crlevels>
    <crlevel value="2"/>
    <crlevel value="4"/>
    <crlevel value="8"/>
</crlevels>
```

Fig. # 29 Muestra la configuración de etiqueta crlevels.

Posteriormente se ejecuta la prueba presionando el botón Run del controlador y este en su parte inferior genera una serie de gráficos con los resultados. Estos representan: el Índice de Escalabilidad (**"Real Time Scalability Index"**), Desarrollo Bajo Carga (**"Performance Under Load"**), la Distribución de Transacciones (**"Transaction**

Distribution"), el Uso de Recursos (**"Resource Monitoring"**). Además existen otros que se encuentran predefinidos, que son usados como plantillas, también le da la posibilidad al usuario de crear nuevos tipos de gráficos y de personalizar los existentes. Esta funcionalidad se encuentra dentro de la opción Configuración de los Gráficos de Análisis de los Resultados ("**Result Analysis Chart Settings**") que se encuentra dentro de la opción Herramientas (**"Tool"**) de la barra de menú de **PushToTest TestMaker**.

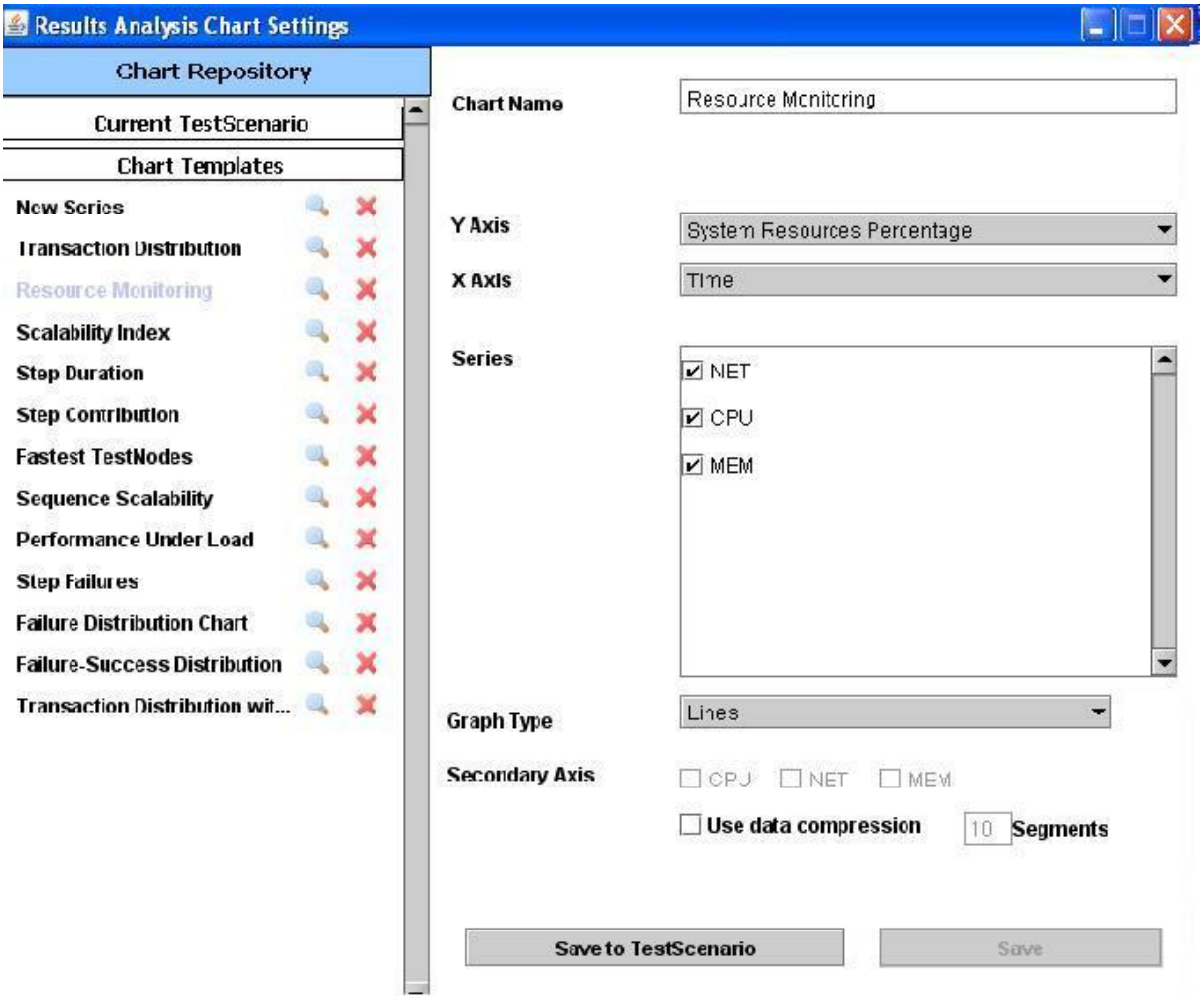


Fig. # 30 Muestra la interfaz para configurar los gráficos.

Un elemento importante es como se interpretan los resultados de cada una de los gráficos. Debido a que PustToTest TestMaker cuenta con una gran variedad de gráficos solo se explicará el que es generado por defecto cuando se realiza una prueba de carga: El Índice de Escalabilidad. El Índice de Escalabilidad muestra la capacidad que posee el sistema de dar respuestas al número creciente o decreciente de peticiones concurrentes, por lo tanto indica como reaccionara este cuando se encuentre en explotación. Está

compuesto, el eje vertical, por las transacciones por segundo (**"Transaction Per Second [TPS]"**) y el eje horizontal por los grupos de usuarios concurrentes que serán simulados en la prueba, en el caso especifico de la prueba que será realizada, los grupos son 2, 4, 8.

Realizando un análisis del gráfico, se debe aclarar que cuando las TPS aumentan en un mismo valor con el aumento de las peticiones concurrentes, el sistema posee buena escalabilidad, en otras palabras, si se simula un escenario cualquiera y se obtiene un índice de escalabilidad de 0.55 TPS para un grupo de 2 usuario, el resultado óptimo de simular un grupo de 4 usuarios sería el doble o más de 0.55 TPS, o sea mayor o igual a 1.1 TPS. En otro caso, de las TPS seguir constantes en los distintos tipos de usuarios significa que la aplicación posee un problema de indexación de la base de datos, un caché de datos o una conexión de red saturada. En caso contrario de disminuir de forma proporcional se produce el efecto de cuello de botella que es cuando el software posee algún factor que retrasa el tiempo de respuesta del mismo. Los resultados obtenidos en la ejecución de la prueba fueron Fig. # 31.

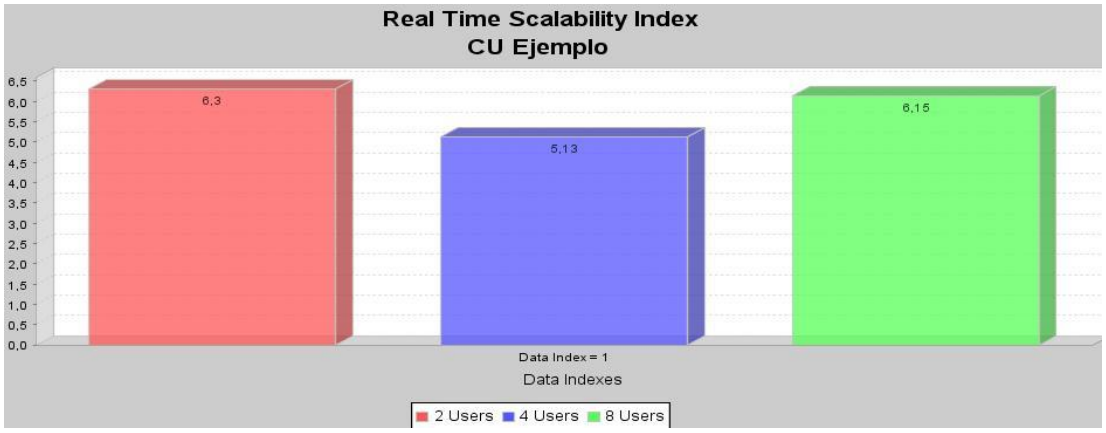


Fig. # 31 Muestra el resultado de la prueba

Al someter a la aplicación bajo la carga en aumento de usuarios concurrentes (UC) pueden ocurrir 3 cosas:

- ✓ El servidor toma menos tiempo (en promedio) para responder a una mayor cantidad de UC. En esta condición cada UC termina primero que sus homólogos de menor concurrencia, registra la respuesta (una transacción), y hace su próxima solicitud mucho antes. Las TPS aumentan en relación con los de una menor cantidad de UC.

- ✓ El servidor toma el mismo tiempo (en promedio) para responder a una mayor cantidad de usuarios concurrentes. Cada usuario concurrente consume la misma cantidad de tiempo que los UC de menor cantidad. Las TPS aumentan proporcionalmente con el aumento de la cantidad de UC.
- ✓ El servidor toma más tiempo (en promedio) para responder a una mayor cantidad de usuarios concurrentes. Los usuarios simulados demoran más que los UC de menos cantidad simulados por lo que deja sin oportunidad al servidor de manejar más solicitudes. Por lo tanto las TPS disminuyen proporcionalmente con el aumento del tiempo de respuesta.

A continuación se muestran las posibles situaciones según los valores de las TPS en los resultados de los diferentes test.

Sucesos	Posible Problema
Aumentan los UC con la disminución de las TPS.	Verificar el tiempo promedio de respuesta
Aumentan los UC con el aumento de las TPS.	La cantidad de usuarios no es lo suficientemente alto.
Aumentan los UC con un pequeño cambio de las TPS.	La cantidad de usuarios es demasiado alta. Ejecutar la prueba con la cantidad de usuarios al 50%.
El CPU del servidor en uso al 95% y aumentan los UC con el aumento de las TPS.	Se debe considerar como el número máximo de peticiones posibles a manejar.
El CPU de los nodos de prueba a un 95% de utilización y aumentan los UC con la disminución de las TPS.	El número de UC es demasiado alto para el número de nodos encargados de generar los UC. Adicionar más nodos a la ejecución de la prueba
El CPU de los nodos de prueba a un 15%	La red posiblemente se encuentra saturada.

de utilización, el CPU del servidor en uso al 30% y aumentan los UC con un pequeño cambio de las TPS.	Chaquear la utilización del ancho de banda y considerar el uso de una red más rápida.

Tabla # 6 TPS en los resultados de los diferentes test.

2.4.2.2.3 Pruebas de Estrés

En el epígrafe 1.5 se exponen los distintos niveles de prueba y dentro de ellos las diferentes pruebas que se pueden aplicar dentro de cada unos de estos niveles. Aunque existen marcadas semejanzas entre los conceptos de prueba de carga y prueba de estrés, se diferencian principal en el objetivo de cada una, mientras que las pruebas de carga se encargan de probar el comportamiento del sistema bajo una carga soportable, las pruebas de estrés son dedicadas a someter al sistema a una carga tan grande que sea capaz de colapsar el mismo.

A la hora de diseñar poseen implementaciones casi parecidas si antes simulamos las pruebas de carga para 2, 4, 8 usuarios, para este tipo de prueba es necesario simular un número de usuarios realmente grande, tomaremos como ejemplo la cifra de 60 usuarios.

2.4.2.2.4 Utilización de PustToTest TestMaker para Pruebas de de Estrés

Para implementar el ejemplo de uso de la aplicación **PushToTest TestMaker** se puede usar la configuración de las etiquetas del ejemplo de pruebas de carga ya que lo único que se debe modificar es la etiqueta **"<crlevels>"** y añadir dentro de esta el siguiente código (Ver Fig. # 32).

```
<crlevels>
    <crlevel value=" 60"/>
</crlevels>
```

Fig. # 32 Muestra la configuración de etiqueta crlevels.

Para este ejemplo se uso la simulación de una cantidad de 60 usuarios. Las pruebas de estrés son útiles para conocer cuál es el límite de carga soportado por la aplicación, con ese índice se define si el sistema cumple con sus objetivos o no.

2.4.2.2.5 Pruebas de volumen

Con **PushToTest TestMaker** se puede realizar también este tipo de pruebas, ya que puede ser fácilmente configurada para realizarlas. A la hora de ejecutar estas pruebas lo que se pretende es que la aplicación maneje una gran cantidad de datos, o que realice actividades de peso, como la interacción con la base de datos. Se tiene que tomar en cuenta también la cantidad de usuarios que serán simulados, ya que se debe recordar siempre no exceder el límite de usuarios para no caer en Estrés.

Además se puede usar otra de las potencialidades de esta herramienta como lo es la realización de pruebas distribuidas en nodos. Esto permite realizar la prueba desde distintas nodos u ordenadores de forma simultánea.

2.4.2.2.6 Utilización de PustToTest TestMaker para Pruebas de Volumen

La configuración del escenario que será usado constituye el mismo que fue usado para las pruebas de carga o estrés y se realizarán pequeñas modificaciones para adecuarlas a este tipo de pruebas en especifico. El número de usuarios que será simulado debe ser un valor que este cerca del límite que soporta la aplicación, se utilizará para este ejemplo 30 usuarios, que se configuran de la misma forma que en los tipos de pruebas anteriores.

Para la creación de las pruebas distribuidas usaremos dos nodos de ejecución de pruebas. Para poder usar otro nodo este debe tener instalado previamente **PushToTest TestMaker**. Luego modificamos la etiqueta "**<testnode>**" agregando el nuevo nodo de la siguiente manera (Ver Fig. # 33).

```
<testnodes>
    <node location="http://localhost:8080/TestNetwork/ws/TestNode" name="localhost"/>
    <node location="http://10.31.17.209:8080/TestNetwork/ws/TestNode" name="nodo2"/>
</testnodes>
```

Fig. # 33 Muestra la configuración de etiqueta testnodes.

2.5 Conclusiones

Durante el desarrollo del presente capítulo se han propuesto las herramientas insertadas dentro de los flujos de trabajo de implementación y prueba del proceso de desarrollo de software, propiciando la temprana detección de defectos que a la postre serían difíciles de

encontrar y corregir. También se ha descrito el proceso de instalación y utilización de las herramientas propuestas, mostrando ejemplo del uso de las mismas.

Capítulo 3: Validación de Resultados

3.1 Introducción

En este documento se ha demostrado la necesidad de la utilización de herramientas para la realización de Pruebas de Software automáticas. En este capítulo se describirá cómo se ha aplicado el proceso y la utilización de las herramientas propuestas en el capítulo 2, las cuales son el TestNG y PustToTest TestMaker, el análisis de los resultados obtenidos durante la aplicación de estas herramientas a dos de los proyectos del polo de telecomunicaciones y así como el resultado del proceso de validación de la propuesta.

3.2 Planificación y Concepción de las Pruebas de Unidad, Integración y Regresión

3.2.1 Estrategia de Pruebas

Objetivo

Las pruebas de Unidad se encargan de probar el código fuente, tienen como objetivo fundamental encontrar errores en el funcionamiento de pequeñas unidades de código, impidiendo que los estos se propaguen hacia el proceso de integración y otras etapas posteriores. Las pruebas unitarias además facilitan la integración ya que permiten llegar a la fase de integración con un alto grado de seguridad de que el código está funcionando correctamente y sirven incluso como documentación del código ya que ahí se puede ver cómo utilizarlo. El objetivo principal de las pruebas de integración es detectar las fallas de interacción entre las distintas clases que componen el sistema. Las pruebas de regresión, son necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

Técnica

La técnica aplicada para las pruebas de unidad es la de caja blanca, utilizando el método del camino básico, esta técnica brinda la posibilidad de saber cuán compleja es una estructura procedimental, elemento utilizado para hallar una serie de caminos que ejecuten por lo menos una vez cada sentencia del programa y además cada condición en su vertiente verdadera y falsa. Para la realización de estas pruebas se utiliza la

herramienta de automatización TestNG, la cual ha sido seleccionada como candidata para realizar estas por las características que ella presenta y por las opciones que brinda para su manipulación.

Entorno de Prueba

El entorno de prueba es un ambiente similar a donde se realiza la aplicación, debido a que es una herramienta que constituye de gran ayuda para al desarrollador, y es él, el encargado de revisar el resultado de su trabajo. El entorno de prueba será el mismo para la aplicación de las pruebas a los dos proyectos, debido a que el código a ser probado será muy semejante, se realizará con la misma herramienta en iguales condiciones de trabajo.

Para la realización de los test de prueba se necesitan los siguientes datos recogidos en el artefacto *Especificación de Recursos*. Se deben acondicionar los recursos tanto de software como de hardware.

Tarea	Recurso	Cantidad de Personas	Fecha Limite
Elaboración del plan de pruebas	**1PC. M Office**	**2**	**29/04/09 (1)** **30/04/09 (2)**
Realizar los test en la herramienta	**1PC. 1Gb de Ram. M Office. Java Virtual Machine 1.3 o superior.** **Se necesita además que se encuentre corriendo el servidor de la aplicación.**	**2**	**04/05/09 (1)** **05/05/09 (2)**
Ejecución de las pruebas	**1PC. 1Gb de Ram. M Office. Java Virtual Machine 1.3 o superior.** **Se necesita además que se encuentre corriendo el servidor de la aplicación.**	**2**	**04/05/09 (1)** **05/05/09 (2)**
Análisis de los resultados	**1PC. M Office**	**2**	**04/05/09 (1)** **05/05/09 (2)**

Tabla # 7 Especificación de recursos para las pruebas de los proyecto Cubacel y TIP.

En el artefacto *Especificación de Recursos* los datos son para los dos proyectos a los cuales se les aplicaron las pruebas, en la columna de la fecha límite se encuentran dos fechas, la fecha "(1)" hace referencia a la del proyecto Cubacel y la fecha "(2)" a la fecha que se realizó la actividad en el proyecto TIP.

3.2.2 Proceso de Prueba

El proceso de automatización de las pruebas consta de 3 etapas fundamentales.

- ✓ Confección de los test.
- ✓ Ejecución de las pruebas y recolección de los resultados.
- ✓ Análisis de los resultados.

Confección de los Test

La confección de los diferentes test se realizaron usando la técnica de camino básico explicada en el capítulo 2, además se realizó especial énfasis en la capa de acceso a datos, la cual ocupa un lugar cimero dentro del correcto funcionamiento de una aplicación, para el funcionamiento adecuado de las capas superiores es necesario que las funcionalidades que brinda el acceso a datos se encuentre lo mejor depuradas posible para que no se propaguen los posibles errores hacia niveles superiores

Ejecución de las pruebas y recolección de los resultados

Para la realización de los distintos tipos de prueba se comprobó que todos los recursos de software se encontraran en perfecto estado o sea, que se encontrara instalado el eclipse, junto con el plugin que soporta la ejecución de las pruebas TestNG, que estuviera disponible el servidor de bases de datos. Las pruebas se realizaron de forma repetitiva para lograr una mayor veracidad en los resultados de cada uno de los diferentes test.

Casos de Prueba

Con tal objetivo se diseñaron casos de pruebas que permitieran verificar las diferentes funcionalidades del software, enfocándose en aquellos métodos implementados en el acceso a datos.

Calendario y plazos

En el cronograma asociado a estas pruebas se definen las tareas que se realizaron acompañadas de sus fechas en la que deben ser ejecutadas.

A continuación se muestra el Cronograma de la aplicación de las pruebas de Unidad a los proyectos Cubacel y TIP:

Inicio	Fecha(Inicio-Fin)	Tiempo de Desarrollo(min)
Realizar los test en la herramienta	**04/05/09**	**20**
Ejecución de las pruebas	**04/05/09**	**1**
Análisis de los resultados	**04/05/09**	**10**
Total	**04/05/09**	**31**

Tabla # 8 Cronograma del Proceso de Pruebas (Unidad) Automatizadas de Cubacel.

Inicio	Fecha(Inicio-Fin)	Tiempo de Desarrollo(min)
Realizar los test en la herramienta	**05/05/09**	**20**
Ejecución de las pruebas	**05/05/09**	**1**
Análisis de los resultados	**05/05/09**	**10**
Total	**05/05/09**	**31**

Tabla # 9 Cronograma del Proceso de Pruebas (Unidad) Automatizadas de TIP.

3.2.3 Realización de los Test

La realización de los test usando la herramienta TestNG quedó conformada por una sola suite de prueba que estaba compuesta clases de prueba que contenían los casos de pruebas que se habían implementado. Se mostrarán los casos de pruebas del Portal Web TIP, para los casos de prueba del Portal Web Cubacel (ver Anexos #4).

Caso de prueba: Listar países

```
@ContextConfiguration(locations = {
            "/cu/uci/tip/pmp/dataacces/TIPdao/config/dataaccess-
config.xml",
            "/cu/uci/tip/pmp/business/config/business-config.xml" })
public class BaseDaoTestCase extends
```

```
            AbstractTransactionalTestNGSpringContextTests {

      ... //Configuración del DataSource..

}
public class CountryDAOTest extends BaseDaoTestCase {

      ...

      @Test
      public void selectAll() {
            String query = "select * from tb_ncountry ";
            List<Map<String, Object>> list =
            simpleJdbcTemplate.queryForList (query);
            System.out.println ("Listado de países existentes:");
            for (Map<String, Object> map: list) {

                  System.out.println (map.get ("country_name"));
            }
      }
```

En este caso de prueba se espera un listado completo de los países existentes en la correspondiente tabla de la base de datos.

Caso de prueba: Seleccionar país por ID

```
@BeforeMethod
@Timed(millis = 500)
public void init()
{
      simpleJdbcTemplate.update("insert into tb_ncountry (pk_country_id,
      country_code, country_name) values (?,?,?)",3, 21, "Argentina");
}

@Test
@Timed(millis = 500)
public void selectCountryById()
{
      String query = "select country_name from tb_ncountry where
      pk_country_id =?";
      String name = (String) simpleJdbcTemplate.queryForObject (query,
                        String.class, 3);
      assertEquals ("Argentina", name);
}
```

Seleccionar País por ID está compuesto por el método `init()`que se encarga de adicionar a la base de datos un nuevo país. Al estar precedido por la anotación `@BeforeMethod` se ejecutara antes que el cualquier método de prueba, con la restricción de que la operación se debe realizar en un plazo de tiempo inferior al pasado por valor a la anotación `@Timed`.

Una vez ejecutado este método de ejecuta el método de prueba `selectCountryById()`, que se encarga de corroborar si el nombre del país del código que se provee coincide con el valor esperado. Si demora más de 500 milisegundo la prueba lanzará una excepción mostrando el fallo de la prueba.

Caso de prueba: Seleccionar provincias dado un ID de país

```
public class ProvinceManagerImplTest extends BaseDaoTestCase
{
...
@Test
@Timed (millis = 500)
public void selectProvinceByCountryID () throws Exception
{
      List<Province> list = provinceManagerImpl.getProvincesByCountryID
(1);

      assertEquals (super.countRowsInTable ("tb_nprovince"),list.size());

      for (Province province: list)
            System.out.println (province.getName ());
}
```

En el caso de prueba anterior se integran las clases `ProvinceManagerImpl` validad que la cantidad de provincias con id = 1 sea igual a la cantidad de elementos que contiene la tabla de la base de datos `"tb_nprovince"` usando la función de TestNG *`assertEquals`*. El método debe ejecutarse en un plazo de tiempo inferior al pasado por valor a la anotación. Posteriormente se muestran los nombres de las provincias que se encuentran en la lista.

3.2.4 Análisis de los resultados de las Pruebas

Al ejecutar los métodos de pruebas antes explicados y otros métodos implementados se generó la siguiente Fig. #. 34 en la consola sobre los distintos resultados de las pruebas en el proyecto TIP. (Ver Anexo 5 para las pruebas del proyecto Cubacel).

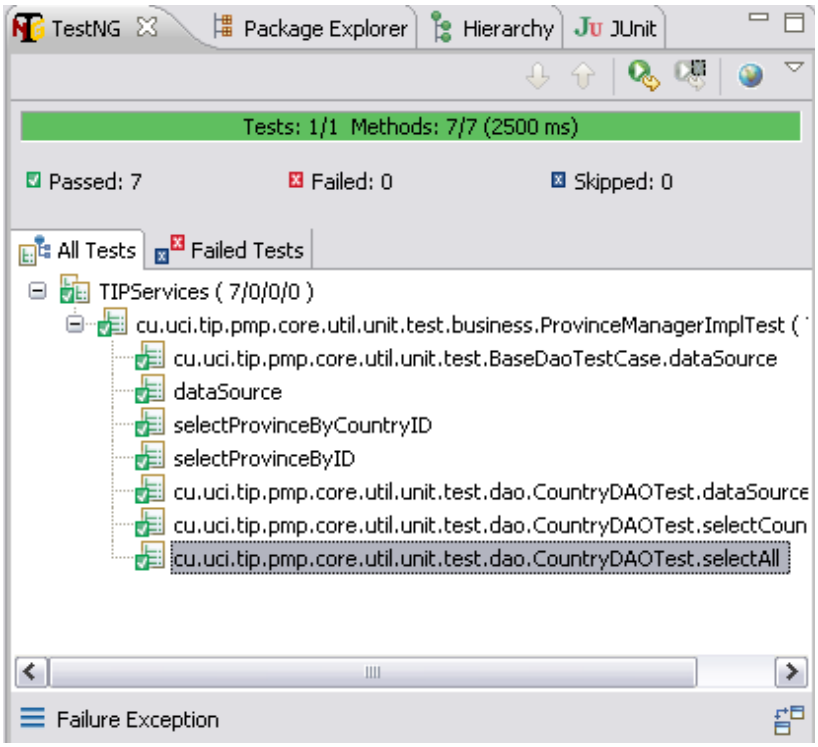


Fig. # 34 Resultados de las pruebas.

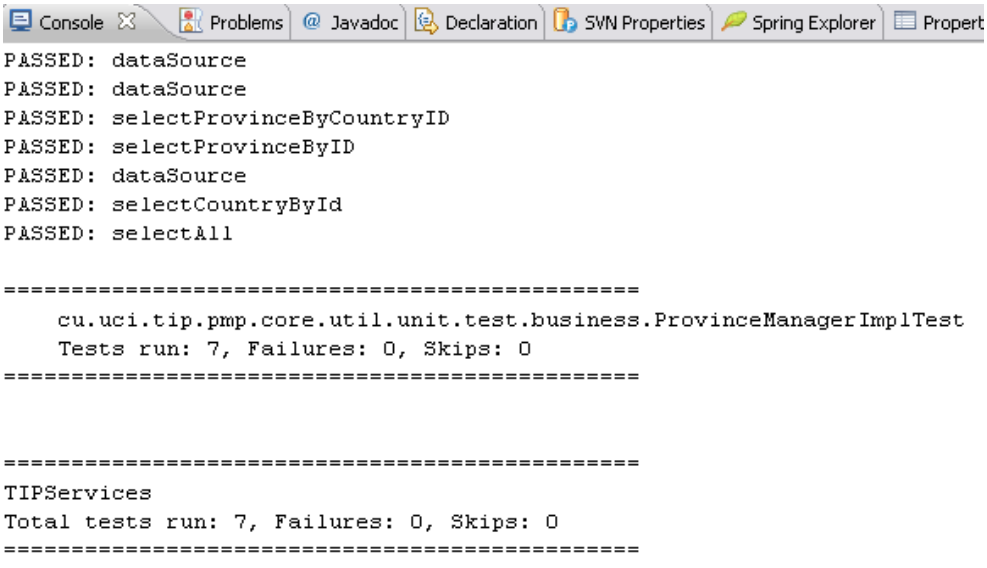


Fig. # 35 Resultados de la Prueba en la Consola de Ejecución.

Otro de los reportes generados es de tipo HTML donde recoge los tiempos de ejecución de cada uno de los métodos de prueba así como otra información necesaria sobre la Suite de prueba ejecutada, los métodos ejecutados y la estructura de las clases que forman parte de la Suite.

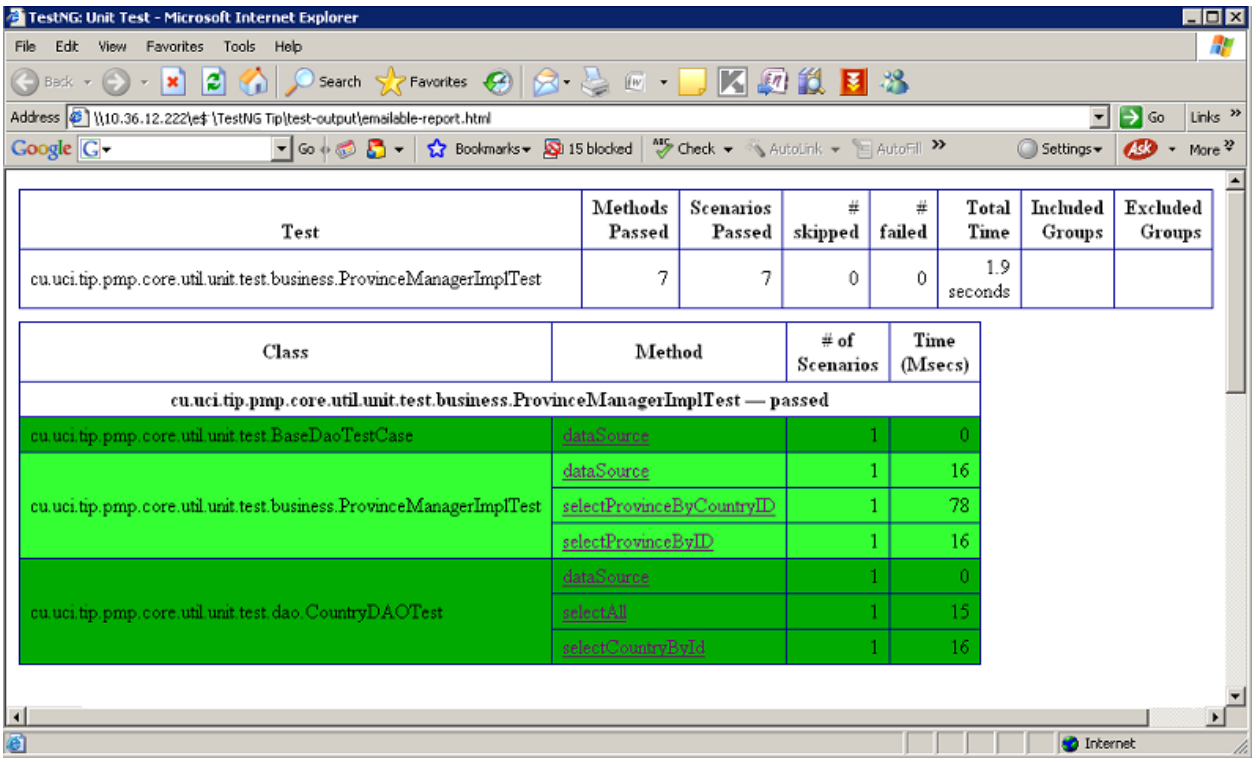


Fig. # 36 Reporte HTML de la prueba ejecutada.

Realizando un análisis de los resultados de las pruebas de Unidad, arrojó que no se encontraron errores en ninguno de los casos de pruebas que le fueron aplicados al software. Además el tiempo consumido por cada uno de los métodos de prueba es bastante pequeño, por lo que demuestra la eficiencia de la aplicación cuando se utiliza para gestión de información proveniente de la base de datos, también muestra el desempeño positivo que tendrá cuando se encuentre en explotación.

3.3 Planificación y Concepción de las Pruebas de Funcionales, Carga, Estrés y Volumen

3.3.1 Estrategia de Pruebas

Objetivos

Para las pruebas de carga, estrés y volumen el objetivo es obtener un índice de resultados de comportamiento del sistema en determinadas condiciones, variándose las condiciones en dependencia del tipo de prueba que se quiera hacer. Al realizar estas pruebas de la aplicación se obtienen resultados de un comportamiento que puede ser

muy cercano a un comportamiento real y en dependencia con esto el equipo de trabajo podría hacer cambios favorables para mejorar el comportamiento de la aplicación.

Técnica

Para las pruebas de carga, estrés y volumen la técnica utilizada es la de caja negra. Estas pruebas necesitan que se realice una entrada de datos y la validación de estos datos contra resultados esperados es por ello que se define como técnica utilizada para este proceso de pruebas la de caja negra. La herramienta que se utilizará para la realización de estas pruebas será PustToTest TestMaker, la cual fue seleccionada en el capítulo 2 del presente trabajo, por sus características y opciones que brinda.

Entorno de Prueba

El entorno de prueba es un ambiente más o menos similar al ambiente donde se realiza la aplicación, no tiene que tener una total semejanza con el ambiente de producción sino que solo es factible simular en una variedad de grados de aproximación, ya que en ocasiones cuesta mucho hacer una semejanza total del ambiente de producción.

Para la grabación de los escenarios de prueba se necesitan los siguientes datos recogidos en el artefacto *Especificación de Recursos*. Se deben acondicionar los recursos tanto de software como de hardware. El entorno de prueba será el mismo para la realización de las pruebas tanto al proyecto Cubacel como TIP, debido a la semejanza que existe entre ambas aplicaciones.

Tarea	Recurso	Cantidad de Personas	Fecha Limite
Elaboración del plan de pruebas	**1PC. M Office**	**2**	**29/04/09 (1)** **30/04/09 (2)**
Realizar los test en la herramienta	**1PC. 1Gb de Ram. M Office. Java Virtual Machine 1.3 o superior. TestMaker.** **Navegador IExplorer.** **Se necesita además que se encuentre corriendo el servidor de la aplicación.**	**2**	**02/05/09 (1)** **03/05/09 (2)**

Ejecución de las pruebas	**1PC. 1Gb de Ram. M Office. Java Virtual Machine 1.3 o superior. TestMaker.** **Navegador IExplorer.** **Se necesita además que se encuentre corriendo el servidor de la aplicación.**	**2**	**02/05/09 (1)** **03/05/09 (2)**
Análisis de los resultados	**1PC. M Office**	**2**	**02/05/09 (1)** **03/05/09 (2)**

Tabla # 10 Especificación de recursos para las pruebas de los proyecto Cubacel y TIP.

En el artefacto *Especificación de Recursos* los datos son para los dos proyectos a los cuales se les aplicaron las pruebas, en la columna de la fecha límite se encuentran dos fechas, la fecha "(1)" hace referencia a la del proyecto Cubacel y la fecha "(2)" a la fecha que se realizó la actividad en el proyecto TIP.

3.3.2 Proceso de Prueba

El proceso de automatización de las pruebas consta de 4 etapas fundamentales.

- ✓ Grabación de los escenarios de prueba.
- ✓ Confección de los test.
- ✓ Ejecución de las pruebas y recolección de los resultados.
- ✓ Análisis de los resultados.

Grabación de los Escenarios de Prueba

La grabación de los escenarios de prueba se conformó por 2 grabaciones independientes por cada proyecto productivo a los cuales fueron aplicadas las pruebas, para la realización de las mismas, se solicitó un representante de cada equipo de desarrollo con conocimientos sobre las páginas críticas del módulo. Esta grabación se realizó utilizando el grabador explicado en el capitulo anterior TestGen4Web.

Habiendo previamente configurado el navegador se accedió a la dirección del servidor web al que se le realizan las pruebas y se navegó a través de todos sus módulos.

Confección de los Test

Para la creación de las diferentes tipos de test, se utilizaron las grabaciones de los escenarios explicados anteriormente y se modifica el archivo XML según el tipo de pruebas a realizar y las necesidades que se necesite probar.

El archivo XML está compuesto de la siguiente manera:

- ✓ <basics>
- ✓ <testnodes>
- ✓ <DataSources>
- ✓ <resources>
- ✓ <loadtest> ó <functionaltest>
- ✓ <crlevels>
- ✓ <run>
- ✓ <chart>

La etiqueta “loadtest”, “crlevels” y “chart” se usa para los tipos de pruebas de carga, estrés y volumen en tanto “functionaltest” para las pruebas funcionales.

Descripción de los elementos de prueba

Los elementos que componen la prueba son configurados de la forma que se explica en el capitulo anterior. El número de usuarios concurrentes que se simulará para cada uno de los portales Web será, para Cubacel de 2, 4, 8, 16, 20, 30, 34, 40, 46 usuarios y los mismos para TIP. Los valores por defectos para las peticiones HTTP fueron:

- ✓ Dirección IP o nombre del servidor donde se encuentra el Portal Web.
- ✓ Tipo de protocolo que se utiliza.
- ✓ Puerto por el que se accede.

Los demás elementos de prueba se añaden como se explica en el epígrafe **2.4.2.2** donde se describe el uso de la herramienta PustToTest TestMaker.

Ejecución de las pruebas y recolección de los resultados

Para la realización de los distintos tipo de prueba se comprobó que todos los recursos de software como de hardware se encontraron en perfecto estado o sea, que la aplicación web se encontrara en funcionamiento, que estuviera disponible el servidor web, que no existieran problemas con la red. Las pruebas se realizaron de forma repetitiva para lograr una mayor veracidad en los resultados de cada uno de los diferentes test.

Casos de Prueba

La prueba se realizó con un único caso de prueba siguiendo la estructura planteada en la confección de los test. Este test recoge toda la actividad del portal web.

Calendario y Plazos

En el cronograma asociado a estas pruebas se definen las tareas que se realizaron acompañadas de sus fechas en la que deben ser ejecutadas. A continuación se muestra el Cronograma de los portales Web de Cubacel y TIP:

Inicio	Fecha(Inicio-Fin)	Tiempo de Desarrollo(min)
Elaboración de las grabaciones de prueba	**02/05/09**	**4**
Realizar los test en la herramienta	**02/05/09**	**5**
Ejecución de las pruebas	**02/05/09**	**10**
Análisis de los resultados	**02/05/09**	**20**
Total	**02/05/09**	**39**

Tabla # 11 Cronograma del Proceso de Pruebas Automatizadas de Cubacel.

Inicio	Fecha(Inicio-Fin)	Tiempo de Desarrollo(min)
Elaboración de las grabaciones de prueba	**03/05/09**	**4**

Realizar los test en la herramienta	**03/05/09**	**5**
Ejecución de las pruebas	**03/05/09**	**10**
Análisis de los resultados	**03/05/09**	**20**
Total	**03/05/09**	**39**

Tabla # 12 Cronograma del Proceso de Pruebas Automatizadas de TIP.

Para cada uno de los proyectos a los cuales se les aplicaron las pruebas, el tiempo de la elaboración de las grabaciones de prueba fueron de 4 minutos. A la realización de los test se le dedicaron 5 minutos, en la ejecución fueron 10 minutos y 20 para el análisis de los resultados dando esto un total de 39 en la ejecución de las pruebas.

Seguimiento y Reporte de defectos

Los resultados que se obtuvieron durante las pruebas fueron analizados por los desarrolladores del software y el ingeniero de pruebas. De estos resultados se destacan los aspectos más importantes que pueden interferir en el rendimiento del sistema.

3.3.3 Realización de los Test

Para el uso de la herramienta PustToTest TestMaker se conformó un único test compuesto por la grabación realizada sobre el sitio web que recogen de manera general 32 peticiones que conforman la navegación más significativa del sistema, donde se simuló la interacción de 2, 4, 8 ,16, 20, 30, 34, 40 y 46 usuarios de forma simultánea.

3.3.4 Ejecución de las Pruebas

Las pruebas fueron realizadas a los portales Web que presentan las siguientes características de software:

Portal Web Cubacel

- ✓ Servidor web: Apache Tomcat 6.0.13
- ✓ Lenguaje utilizado: Java
- ✓ Con el siguiente hardware de servidor:

- ✓ Disco Duro: ST380817AS (160 GB, 7200 RPM, SATA)
- ✓ Memoria del sistema: 512 Mb (PC3200 DDR SDRAM)
- ✓ Procesador: Intel Pentium 4, 3000 MHz

Portal Web TIP

- ✓ Servidor web: Apache Tomcat 6.0.13
- ✓ Lenguaje utilizado: Java
- ✓ Servidor de Base de Datos: Postgres sql
- ✓ Con el siguiente hardware de servidor:
- ✓ Disco Duro: ST380817AS (160 GB, 7200 RPM, SATA)
- ✓ Memoria del sistema: 512 Mb (PC3200 DDR SDRAM)
- ✓ Procesador: Intel Pentium 4, 3000 MHz

3.3.5 Resultados de las Pruebas

Solo se mostrarán los resultados obtenidos de aplicar las pruebas al proyecto TIP y las correspondientes al proyecto Cubacel serán añadidas en los Anexos # 3, 6 y 7.

En general las peticiones realizadas a la aplicación tienen un tiempo promedio de respuesta de 110 milisegundos lo cual demuestra que ha sido implementado siguiendo los requerimientos establecidos en los que se plantean buenas prácticas de programación. El tiempo total promedio de navegar por los 33 pasos de la grabación es de 3640.6667 milisegundos, siendo 5719 y 3094 milisegundos el tiempo máximo y mínimo respectivamente que se necesita para recorrerlos.

Tiempo Promedio Total (ms)	Tiempo Promedio por paso (ms)	Tiempo Máximo(ms)	Tiempo Mínimo (ms)	Cantidad de Errores	TPS
3640.6667	**110.32323**	**5719**	**3094**	**0**	**0.33333334**

Tabla # 13 Resultados de las pruebas funcionales.

La fig. # 37 muestra el resultado de la realización de pruebas de carga, estrés y volumen. Esta está compuesta en el eje de las X por las transacciones por segundo (TPS) que es capaz de realizar el servidor de la aplicación web, para dar respuesta a los pedidos de forma simultánea a que es sometida la aplicación (eje Y).

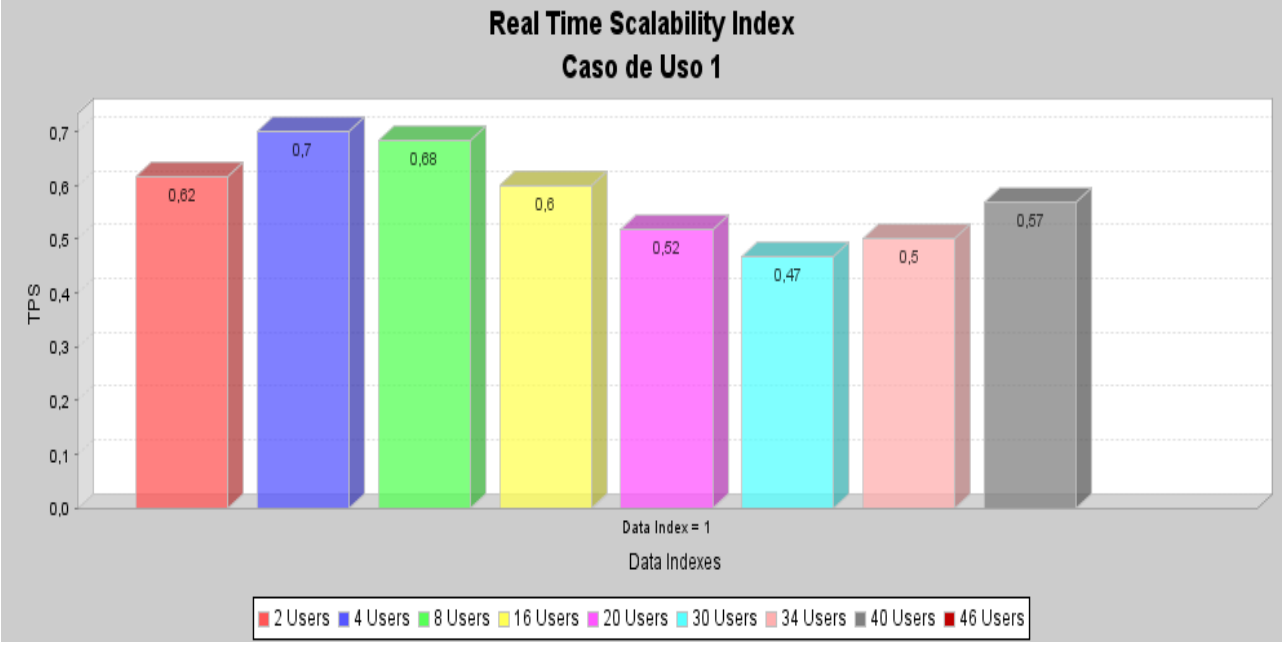


Fig. # 37 Resultados de las prueba Carga y Estrés.

Los resultados de las prueba arrojan que el servidor no es capaz de soportar la carga de 46 usuarios de forma concurrente. Siendo 40 el número máximo de conexiones soportada por la aplicación al unisonó. A continuación se muestran otros de los gráficos obtenidos de la realización de los test:

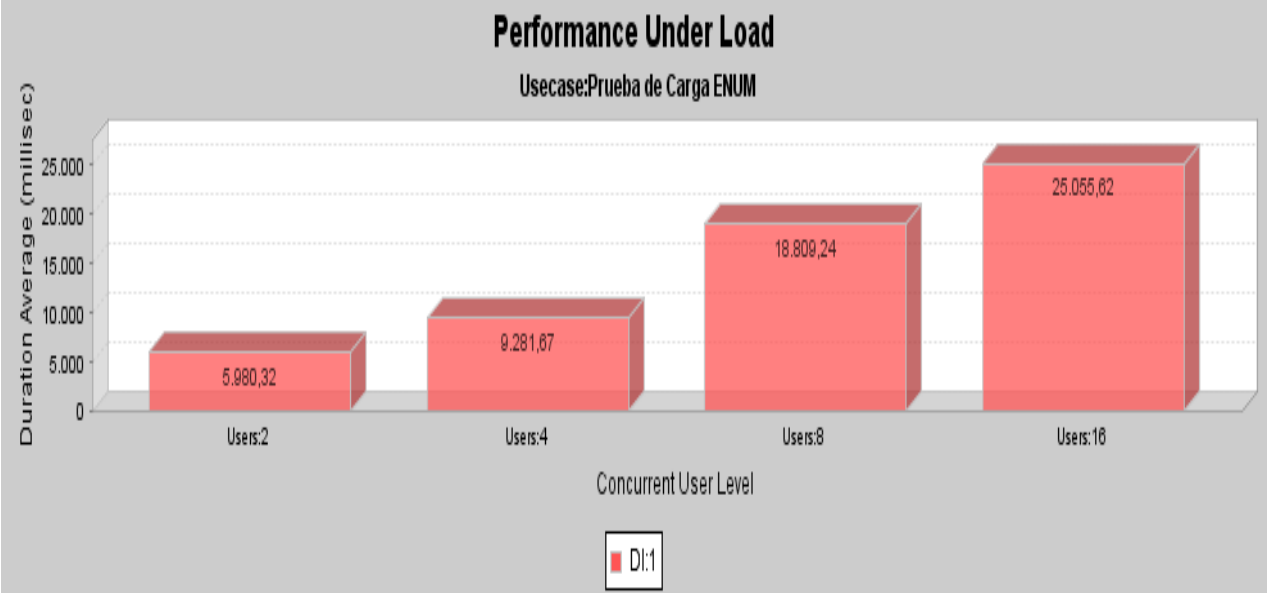


Fig. # 38 Desenvolvimiento bajo carga.

La Fig. # 38 muestra el tiempo total en milisegundos que se necesita para dar respuesta a cada nivel de usuarios concurrentes simulado, en este caso para 2, 4, 8, 16 usuarios. El resultado no es positivo, ya que el sistema tiene que ser capaz de responder en el mismo tiempo todas las peticiones concurrentes simuladas. O sea, el servidor debe mantener constante el tiempo de respuesta, independientemente de la cantidad de usuarios que se encuentren interactuando con el software.

Para cada nivel de usuarios simulado en los casos de pruebas, TestMaker crea un diagrama de Distribución de Transacciones en el Tiempo (Ver Fig. # 39). En eje vertical indica la duración de las transacciones en milisegundos medidas durante la ejecución de la prueba (eje horizontal). Esta gráfica es muy importarte a la hora de identificar largas o cortas tendencias en las tasas de respuesta de la aplicación.

En la Fig. # 39 muestra la ejecución del caso de prueba: Prueba de Carga TIP para una cantidad de 8 usuarios.

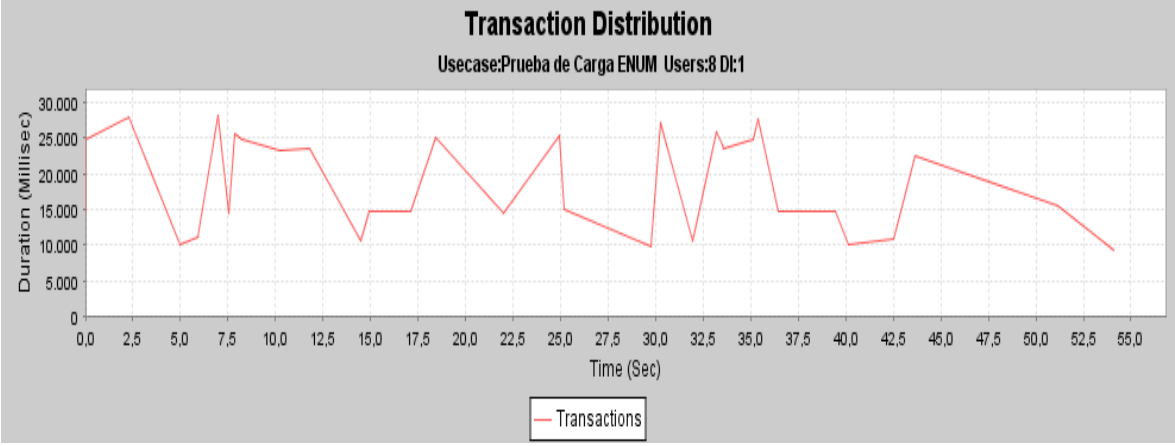


Fig. # 39 Distribución de las Transacciones en el Tiempo.

Observando la figura anterior se obtiene como resultado que la aplicación consume demasiado tiempo para dar respuesta a cada una de las peticiones de forma general, hasta el punto de acercarse a los 30000 ms (30s).

Realizando un resumen de los datos brindados por las pruebas, los resultados no son satisfactorios debido a que el servidor donde se encuentra la aplicación no es el óptimo. El mismo se encuentra con deficientes recursos lo cual limita la cantidad de conexiones, y aumenta el tiempo de respuesta de cada una de las peticiones.

3.4 Validación de la propuesta de solución

Luego de proponer las herramientas para dar solución a los problemas detectados durante la realización de pruebas a los productos en los proyectos de telecomunicaciones de la Facultad 2, fue necesario conocer el grado de aceptación de estas por expertos en temas de calidad de software y realización de pruebas.

La validación se realizó utilizando el método Delphi basado en el criterio de expertos. A continuación se muestra el procedimiento a emplear para la aplicación de este método:

- ✓ **Formulación del Objetivo de la Evaluación**

Valorar la Propuesta de Herramientas para Pruebas de Software Automáticas, tanto en la calidad que presenta su concepción teórica, como la efectividad que se espera alcanzar con su aplicación en la práctica en los proyectos productivos del Polo de Telecomunicaciones de la Faculta 2.

- ✓ **Selección de los expertos**

Se conformó un panel integrado por expertos de la Dirección de Calidad de Software de la Universidad del Departamento de Pruebas de Liberación.

Los expertos se seleccionaron según las características:

- Conocimientos acerca de los contenidos que sustentan la propuesta de implementación.
  - Calidad de Software.
  - Pruebas de Software.
  - Pruebas Automáticas de Software.
- Prestigio en el colectivo de trabajo.
- Capacidad de análisis y pensamiento lógico.
- Integración a las actividades productivas.

Elegir los expertos atendiendo a las características mencionadas propició obtener resultados con calidad, junto a otras cualidades propias de estos como fueron: la seriedad, la honestidad, la sinceridad, la responsabilidad y otras en este sentido, que hicieron que las opiniones brindadas fueran confiables y válidas para el objetivo propuesto.

Una vez confeccionado el panel se invitó cada experto de manera personal para que participaran en el proceso de validación y aceptación de la propuesta.

- ✓ **Elaboración del cuestionario**

En el cuestionario elaborado se hizo énfasis en las preguntas de tipo cerradas, pero en la totalidad de ellas se incluyó la posibilidad de que el experto emitiera su criterio sobre el tema tratado, algo característico de las preguntas abiertas. Esta forma de relacionar ambos tipos (abiertas y cerradas) permitió hacer en estos casos análisis cuantitativos y cualitativos sobre la misma pregunta.

- ✓ **Elección de la metodología a seguir**

El método de expertos que se utilizó fue el de la Metodología de la Preferencia que es la más empleada, por su exactitud, objetividad y rapidez.

Esta metodología permitió superar las limitaciones, relacionadas con la complejidad de su aplicación y del procesamiento de los datos y alcanzó una imagen integral y más amplia de la posible evolución del resultado científico sometido a valoración, reflejando las valoraciones individuales de los expertos, las cuales estuvieron fundamentadas, tanto en un análisis estrictamente lógico como en su experiencia intuitiva y a la vez facilitó el correspondiente análisis estadístico.

- ✓ **Ejecución de la metodología**

Los expertos que conformaron el panel recibieron un resumen de la propuesta de solución como documentación primaria para responder los temas encuestados, además del cuestionario con un total de 3 preguntas con las características explicadas anteriormente.

- ✓ **Procesamiento de la información**

A continuación se detallan las preguntas realizadas, así como, una valoración general de los resultados de ellas.

1. ¿Considera usted que las herramientas propuestas para realización de pruebas automáticas pueden aportar mejoras a la calidad de las aplicaciones desarrolladas en los proyectos productivos de la Facultad 2?

   Todos los expertos coinciden en que la propuesta presentada puede aportar mejoras en la calidad de las aplicaciones desarrolladas en la Facultad 2.

2. ¿Considera que estas dos herramientas, el PustToTest TestMaker y el TestNG pueden ser utilizadas en los proyectos de la Facultad 2 para ser empleadas en sus procesos de pruebas?

   Todos los expertos coinciden en si pueden ser utilizadas e inclusive pueden ser utilizadas por varios proyectos productivos de la Universidad los cuales por sus características las emplearían íntegramente en sus procesos de pruebas.

3. ¿Qué evaluación le otorgaría a la solución propuesta?

   El total de los expertos entrevistados consideran que con la utilización del presente trabajo se incrementaría la calidad final de los productos desarrollados en los

proyectos productivos de la Facultad 2, por tal razón le otorgan la evaluación de “Bien”.

Además el 100% de los entrevistados, le atribuyen gran importancia al trabajo, pues en la Universidad no se tiene un grupo de herramientas que contemple todo el proceso de pruebas y estas pueden contribuir a crear un grupo de herramientas, con las cuales los especialistas en el laboratorio de calidad, puedan probar todas las aplicaciones desarrolladas por la Universidad.

3.5 Conclusiones del Capítulo

La realización de las pruebas a los proyectos sirvió para demostrar la calidad de los productos desarrollados por los mismos y en qué medida estos cumplen con las expectativas del cliente. Se demostró que con la utilización de las herramientas para la realización de las pruebas, se ahorra tiempo y recursos al proyecto. Las pruebas en general se realizaron de manera eficiente con una buena organización y todos los procedimientos quedaron reflejados en el presente trabajo, lo que apunta favorablemente a la definición del proceso como una buena práctica de concepción, elaboración y ejecución de las pruebas de unidad, carga, estrés, volumen y regresión de manera automatizada.

Con este capítulo además se demuestra la validez de la propuesta a través del estudio y la valoración del presente trabajo por un grupo de expertos a los cuales se les aplicó el método de validación “Delphi” a través de entrevistas a los mismos.

Conclusiones Generales

Este trabajo ha tratado de demostrar la importancia que tienen las pruebas de unidad, carga, volumen, regresión y estrés de manera automatizada en las aplicaciones desarrolladas por los proyectos productivos de la Facultad 2, demostrando que la aplicación de herramientas en este proceso, la definición de tareas de manera planificada y organizada, que se necesitan para realizar estas pruebas favorece a los equipos de producción ya que la inversión total de tiempo es muy pequeña en comparación a la inversión que se necesitaría si se utilizara personal para simular cualquier cantidad de usuarios necesarios para probar las aplicaciones.

En el trabajo se hace una descripción del proceso de prueba de unidad, carga, volumen, regresión y estrés, así como se presenta una descripción en forma de manual en el capítulo 2 acerca del uso de las herramientas PustToTest TestMaker y TestNG, este manual está orientado al entendimiento de la herramienta y la importancia de cada uno de los elementos que ella brinda en la elaboración de las pruebas.

Con esta investigación y el manual, fácilmente cualquier especialista implicado en el rol de las pruebas puede aplicar este proceso a su entorno de trabajo y obtener mejores resultados en menos tiempo y con menos recursos, ya que los conocimientos que aquí se reflejan se han expresado de manera organizada ejemplificando cada una de ellos y mostrándolos de manera amena y sencilla.

Con respecto a los resultados de las pruebas aplicadas en los proyectos productivos se considera que han sido de gran importancia ya que han confirmado la calidad de los productos desarrollados, no se pudieron obtener datos totalmente exactos con respecto a algunas pruebas ya que las aplicaciones serán puestas en funcionamiento en otras condiciones de hardware, pero se obtuvo una referencia del nivel de calidad del trabajo realizado, estos resultados fueron dados y sometidos a evaluación por el equipo de evaluación de cada uno de los proyectos.

En esta investigación, para dar el cumplimiento a los objetivos planteados, se realizó:

- ✓ Un estudio bien detallado de las Herramientas de Pruebas de Software Automáticas existentes en la actualidad y asiendo especial énfasis en aquellas que

pudieran ser vinculas, según las tecnologías que manejan, con los proyectos del Polo de Telecomunicaciones.

- ✓ Tomando como punto de partida el estudio realizado, una propuesta de dos herramientas, para el uso dentro de los proyectos productivos, que agilizaran el proceso de prueba dentro de los mismos.

Recomendaciones

- ✓ Exhortar a los equipos de desarrollo de la Facultad 2 a la utilización de estas herramientas dentro del proceso de Prueba.

- ✓ Publicar el manual confeccionado para que pueda ser utilizado en otros proyectos productivos de la Universidad.

- ✓ Crear un repositorio en la Facultad con estas herramientas y otras más, para que los líderes de proyectos puedan contar con ellas para el proceso de prueba en los proyectos productivos

- ✓ Designar un miembro de cada uno de los proyectos productivos (preferiblemente con el rol de probador) para que se especialice en la utilización del uso de herramientas automáticas, para que estas sean empleadas correctamente en los proyectos productivos.

Referencias Bibliográficas

1. Pressman, R.S.,"Ingeniería del software. Un enfoque práctico". 2ª ed. 1992: McGraw-Hill

2. Pressman, R.S., "Ingeniería del Software. Un enfoque práctico". Quinta ed. 2002: s.l Mc Graw Hill.

3. D'Onofrio, A.-P.D.L., "Probando software y números de versión". 2002

4. Craig, Rick D.; Jaskiel, Stefan P. (2002). "*Systematic Software Testing"*. Artech House Publishers. ISBN 1-58053-508-9.

5. Piattini, Mario G.; *et al.* (2004). "*Análisis y diseño de aplicaciones informáticas de gestión, una perspectiva de ingeniería del software".* Alfaomega,

6. Kaner, C.; Falk, J.; Nguyen, H.Q (1993). "*Testing Computer Software"*, segunda edición. Van Nostrand Reinhold.

7. Myers, Glenford (*2004*). "*The Art of Software Testing"*, segunda edición. John Wiley & Sons. ISBN 0-471-04328-1.

8. Gutiérrez, Javier Jesús. (2005). "Generación de pruebas de sistema a partir de la especificación funcional". [En línea] http://www.lsi.us.es/~javierj/investigacion_ficheros/NDEAv25.pdf [Consulta: enero de 2009].

9. Jacobson, I.; Booch, G.; Rumbaugh, J. (1999). "*El proceso Unificado del Software".* Addison-Wesley Longman, Amsterdam.

10. José María Serrano Chica, Salvador García López. 2009. Universidad de Jáen. Asignatura de Computación y Estadística. [En línea] 22 de Febrero de 2009. [Citado el: 02 de Mayo de 2009.] http://wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf.

Bibliografía

1. **Beust. C; Suleiman.H**."Next Generation Java™ Testing TestNG and Advanced Concepts".

2. **Beust, Cédric**. Next-Generation Testing with TestNG. **.** [En línea] 2009. [Citado el: 21 de Febrero de 2009.] www.artima.com/lejava/articles/testng.html

3. **Equipo de Desarrollo**. JUnit. JUnit. [En línea] 2003. [Citado el: 02 de 03 de 2009.] http://www.junit.org/taxonomy/term/12.

4. **Equipo de Desarrollo**. TestNG. Documentation. [En línea] [Citado el: 24 de febrero de 2009.] http://testng.org/doc/documentation-main.html.

5. **Equipo de Desarrollo.** Push To Test. Push To Test. [En línea] 2009 [Citado el: 16 de Enero de 2009.] http://www.pushtotest.com.

6. **Equipo de Desarrollo.** Jakarta. JMeter. [En línea] 2009 [Citado el: 17 de febrero de 2009.] http://jakarta.apache.org/jmeter/.

7. **Equipo de Desarrollo.** Parasoft. Parasoft. [En línea] 2009 [Citado el: 03 de marzo de 2009.] http://www.parasoft.com/jsp/products/home.jsp?product=Jtest.

8. **Equipo de Desarrollo.** Compuware. QALoad. [En línea] 2009. [Citado el: 05 de Marzo de 2009.] http://www.compuware.com/products/qacenter/qaload.htm.

9. **Equipo de Desarrollo.** Opensourcetesting. Java Testing Tools. [En línea] Mayo de 2008 [Citado el: 05 de Marzo de 2009.].

10. **Equipo de Desarrollo.** Opensourcetesting. Funtional Testing Tools. [En línea] [Citado el: 05 de Marzo de 2009.] http://www.opensourcetesting.org/functional.php.

11. **Equipo de Desarrollo.** Open Source**.** Performance Testing Tools. [En línea]. [Citado el: 20 de Marzo de 2009.] http://www.opensourcetesting.org/performance.php

12. **Equipo de Desarrollo.** Selenium. Selenium. [En línea] 2008. [Citado el: 22 de Enero de 2009.] http://seleniumhq.org/.

13. **Equipo de Desarrollo.** Developer.spikesource. TestGen4Web. [En línea] 21 de Mayo de 2008. [Citado el: 04 de Febrero de 2009.] developer.spikesource.

14. **Javier J. Gutiérrez, Darío Villadiego, María J. Escalona, Manuel Mejías (2004).** Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. "Aplicación de La programación orientada a aspectos en el diseño e implementación de pruebas funcionales".

15. **José María Serrano Chica, Salvador García López. 2009.** Universidad de Jáen. Asignatura de Computación y Estadística. [En línea] 22 de Febrero de 2009. [Citado el: 02 de Mayo de 2009.] http://wwwdi.ujaen.es/asignaturas/computacionestadistica/pdfs/tema6.pdf.

16. **Massol. V; Husted .T**."JUnit in Action".

17. **Pressman, R.S.,** "Ingeniería del software. Un enfoque práctico". 2ª ed. 1992: McGraw-Hill

18. **Pressman, R.S.,** "Ingeniería del Software. Un enfoque práctico". Quinta ed. 2002: s.l Mc Graw Hill.

19. **Rainsberger. J.; Stirling. S.,**"JUnit Recipes Practical Methods for Programmer Testing"

Glosario de Términos

- ✓ **FTP**: Protocolo de Transferencia de Archivos) es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor.

- ✓ **Jakarta**: no es un software en sí mismo, sino que más bien se puede ver como un proyecto de proyectos, un repositorio. Está compuesto por varios subproyectos que dan soluciones a problemas en particular.

- ✓ **JDBC**: Es un API de Java, la cual permite conectar los programas escritos en Java con la base de datos.

- ✓ **RUP**: Proceso Unificado Racional es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

- ✓ **SOAP**: es un protocolo elaborado para facilitar la llamada remota de funciones a través de Internet.

- ✓ **Software**: es el conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema.

- ✓ **XML**: Es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de una gran variedad de datos

- ✓ **JAR**: Archivo de Java, del inglés Java Archive.

- ✓ **HTTP**: Protocolo de transferencia de hipertexto, del inglés Hypertext Transfer Protocol. Es el protocolo usado en cada transacción de la Web.

- ✓ **HTML**: Lenguaje de marcación de hipertexto. Es el lenguaje para la publicación de hipertexto en la World Wide Web (www)

- ✓ **Java**: lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90

- ✓ **JNDI**: Interfaz de Nombrado y Directorio Java es una Interfaz de Programación de Aplicaciones (API) para servicios de directorio.

- ✓ **LDAP**: Protocolo Ligero de Acceso a Directorios es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.

- ✓ **XML-RPC** es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes

- ✓ **Jython**: Python en Java es un lenguaje de programación de alto nivel, dinámico y orientado a objetos basado en Python e implementado en Java (100%),

- ✓ **PHP**: es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor

- ✓ **Ruby**: es un lenguaje de programación interpretado, reflexivo y orientado a objetos. Combina una sintaxis inspirada en Python, Perl con características de programación orientada a objetos.

- ✓ **Web 2.0**: se refiere a una segunda generación en la historia de la Web.

- ✓ **HTTPS**: Protocolo seguro de transferencia de hipertexto es un protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto, es decir, es la versión segura de HTTP.

- ✓ **SMTP**: Protocolo Simple de Transferencia de Correo, es un protocolo de la capa de aplicación. Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos

- ✓ **POP3**: se utiliza en clientes locales de correo para obtener los mensajes de correo electrónico almacenados en un servidor remoto. La mayoría de los suscriptores de los proveedores de Internet acceden a sus correos a través de POP3.

- ✓ **IMAP**: es un protocolo de red de acceso a mensajes electrónicos almacenados en un servidor. Mediante IMAP se puede tener acceso al correo electrónico desde cualquier equipo que tenga una conexión a Internet.

- ✓ **Groovy**: es un lenguaje de programación orientado a objetos implementado sobre la plataforma Java. Tiene características similares a Python, Ruby, Perl y Smalltalk.

- ✓ **JRE**: En tiempo de ejecución Java y se corresponde con un conjunto de utilidades que permite la ejecución de programas java sobre todas las plataformas soportadas.

- ✓ **Mozilla Firefox**: es un navegador de Internet libre y de código abierto descendiente de Mozilla Application Suite, desarrollado por la Corporación Mozilla, la Fundación Mozilla y un gran número de voluntarios externos.

- ✓ **SOAP**: es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de XML-RPC. SOAP fue creado por Microsoft, IBM y otros y está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.

Anexos

Anexo # 1. Entrevistas realizadas a los líderes de proyecto del Polo de Telecomunicaciones

Entrevista para obtener información sobre la aplicación de Pruebas a los proyectos productivos del Polo de Telecomunicaciones

Elaborada por: Karel Alejandro Charro Pérez

Andrés Luis Licea Castellanos

Universidad de las Ciencias Informáticas

La presente entrevista tiene como objetivo la obtención de información para la realización de una propuesta de Herramienta de Pruebas de Software Automáticas a los software desarrollados en el Polo de Telecomunicaciones.

Esta propuesta será utilizada con el fin de disminuir la cantidad de personal y tiempo empleado en la realización de pruebas a productos, así como a mejorar la calidad de estos.

1. ¿Cuál es la metodología de desarrollo que se aplica en el proyecto?
2. ¿Qué tipos de pruebas se aplican en el proyecto?
3. ¿Desde cual flujo de trabajo comienzan a realizar las pruebas?
4. ¿Utilizan Herramientas Automáticas para la aplicación de estas pruebas, en caso positivo mencionarlas?
5. ¿Qué tipo de aplicaciones desarrollan? Web___, Desktop___ , ambas___
6. ¿Cuál o cuáles son los lenguajes de programación utilizados en el proyecto?

Anexo # 2. Autorizo para Realizar Pruebas a los Proyectos Productivos

FACULTAD 2 TELECOMUNICACIONES Y SEGURIDAD INFORMATICA.

CURSO 2008-2009

Ciudad Habana, 15 de Marzo de 2009.

"Año 51 de la Revolución"

Autorizo para Pruebas de Software

Yo_____ Ingeniero en Ciencias Informáticas y líder del proyecto _____, autorizo a los estudiantes Karel Alejandro Charro Pérez y Andrés Luis Licea Castellanos que optan por el título de Ingenieros en Ciencias Informáticas a realizarles pruebas mediante la utilización de herramientas automáticas a la aplicación Web desarrollada por el proyecto productivo _____ a utilizar el resultado de estas pruebas en la investigación de su trabajo de diploma.

_____ _____

Nombre del que solicita Nombre del que autoriza

Anexo #3. Resultados de las Pruebas Funcionales a TIP.

Pasos de la grabación	Tiempo promedio	Tiempo máximo	Tiempo mínimo	%Error
1	984.1667	2953	375	0
2	0	0	0	0
3	213.66667	313	156	0
4	0	0	0	0
5	182.16667	219	156	0
6	0	0	0	0
7	182.5	219	156	0
8	0	0	0	0
9	275.66666	406	218	0
10	0	0	0	0
11	177.33333	204	156	0
12	0	0	0	0
13	177.0	218	157	0
14	0	0	0	0
15	10.333333	16	0	0
16	2.6666667	16	0	0
17	164.0	187	141	0
18	2.6666667	16	0	0
19	190.33333	235	156	0
20	2.5	15	0	0
21	169.33333	203	141	0
22	0	0	0	0
23	200.5	359	156	0
24	0	0	0	0
25	166.33333	204	156	0
26	0	0	0	0
27	166.66667	218	141	0
28	0	0	0	0
29	166.33333	203	156	0
30	0	0	0	0
31	2.66666667	16	0	0
32	174.5	219	141	0
33	0	0	0	0

Anexo #4. Casos de Prueba de Unidad en Cubacel.

Caso de prueba: Mostrar cantidad de suscripciones.

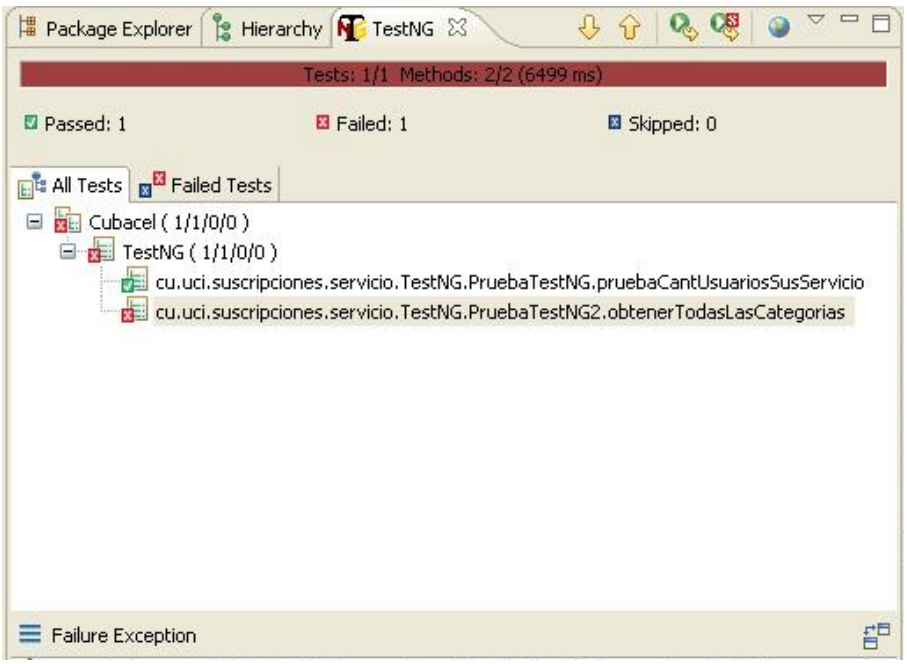
```
@Test (groups = { "suscripciones" })
public void pruebaCantUsuariosSusServicio()
{
	//System.out.println(3);
	IMostrarReporteSuscripciones test =
	(IMostrarReporteSuscripciones)context.getBean("MostrarReporteSus");
	Assert.assertNotNull(test);
	int a = test.obtenerCantidadUsuariosSuscritosAUnServicio("80");
```

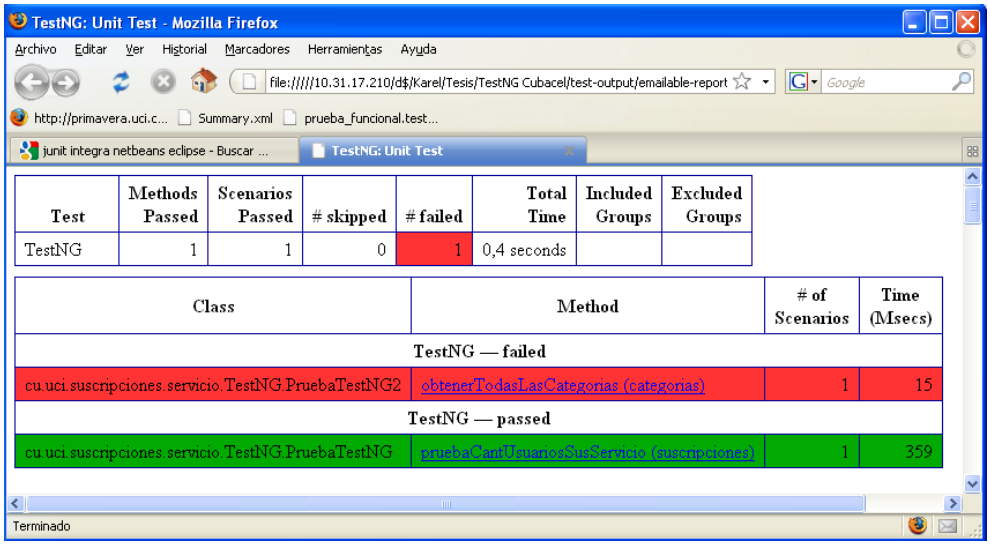
```
        System.out.println(a);
}
```

Caso de Prueba: Obtener categorías de servicios

```
@Test(groups = { "categorias" })
public void obtenerTodasLasCategorias()
{
        IMostrarInformacion inf=
        (IMostrarInformacion)context.getBean("MostrarInfo");
        Assert.assertNotNull(inf);
        List<Categoria> serv= inf.obtenerTodasLasCategorias();
        System.out.println(serv.get(0).getNombre());
        System.out.println(serv.get(1).getNombre());
        Assert.assertNotNull(serv);
}
```

Anexo #5. Resultados de las Pruebas de Unidad en Cubacel.





Test	Methods Passed	Scenarios Passed	# skipped	# failed	Total Time	Included Groups	Excluded Groups
TestNG	1	1	0	1	0,4 seconds		

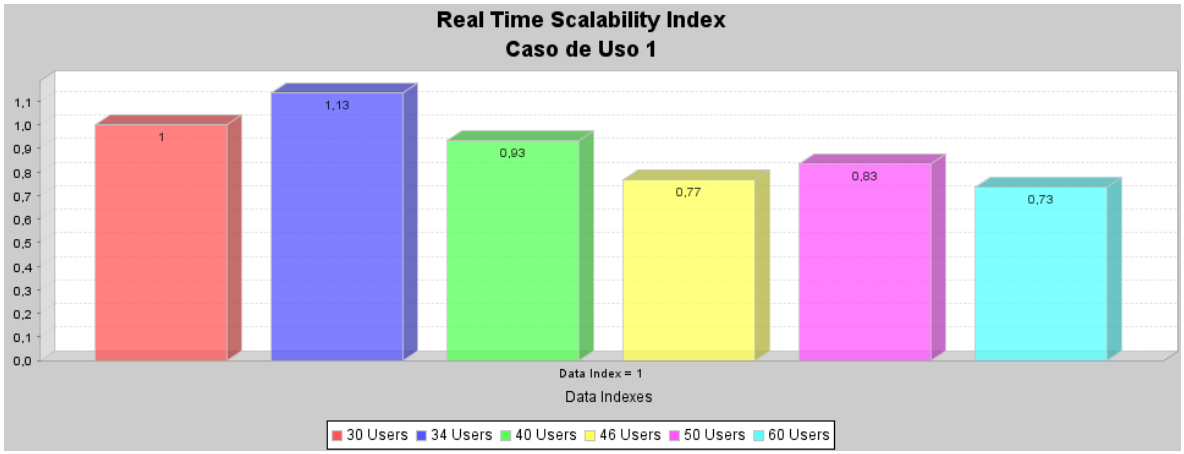
Class	Method	# of Scenarios	Time (Msecs)
TestNG — failed			
cu.uci.suscripciones.servicio.TestNG.PruebaTestNG2	obtenerTodasLasCategorias (categorias)	1	15
TestNG — passed			
cu.uci.suscripciones.servicio.TestNG.PruebaTestNG	pruebaCantUsuariosSusServicio (suscripciones)	1	359

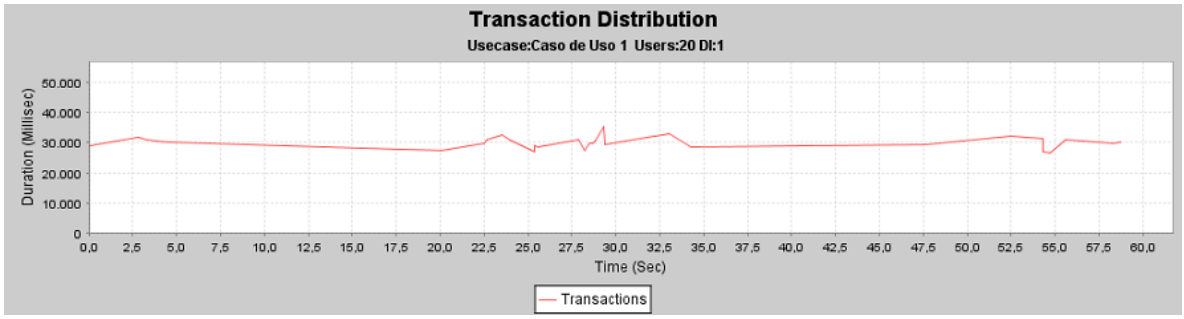
Anexo #6. Resultados de las Pruebas Funcionales a Cubacel.

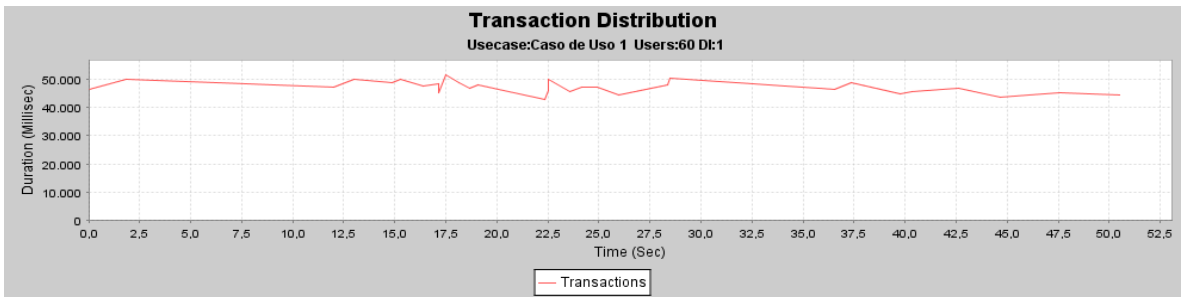
Tiempo Promedio Total (ms)	Tiempo Promedio por paso (ms)	Tiempo Máximo (ms)	Tiempo Mínimo (ms)	Cantidad de Errores	TPS
2036.5	63.64	2734	1469	0	0.54545456

Pasos de la grabación	Tiempo promedio	Tiempo máximo	Tiempo mínimo	%Error
1	111.666664	171	93	0
2	0.0	0	0	0
3	112.0	172	78	0
4	0.0	0	0	0
5	171.83333	281	109	0
6	0.0	0	0	0
7	122.5	172	94	0
8	0.0	0	0	0
9	104.166664	172	62	0
10	0.0	0	0	0
11	197.83333	344	109	0
12	0.0	0	0	0
13	119.833336	219	94	0
14	0.0	0	0	0
15	143.16667	219	94	0
16	0.0	0	0	0
17	127.5	234	93	0
18	0.0	0	0	0
19	127.666664	391	62	0
20	0.0	0	0	0
21	132.83333	204	94	0
22	0.0	0	0	0
23	93.666664	141	78	0
24	0.0	0	0	0
25	119.833336	172	63	0
26	0.0	0	0	0
27	143.33333	188	79	0
28	0.0	0	0	0
29	106.666664	172	62	0
30	0.0	0	0	0
31	99.333336	157	63	0
32	0.0	0	0	0

Anexo #7. Resultados de las Pruebas de Carga, Estrés y Volumen a Cubacel.







Anexo #8. Entrevista utilizada para la validación de la propuesta

1- ¿Considera usted que las herramientas propuestas para realización de pruebas automáticas pueden aportar mejoras a la calidad de las aplicaciones desarrolladas en los proyectos productivos de la Facultad 2?

Si____ No____

2- ¿Considera que estas dos herramientas, el PushTotest Testmaker y el TestNG pueden ser utilizadas en los proyectos de la Facultad 2 para ser empleadas en sus procesos de pruebas?

Si____ No____

Nota: En caso de ser no la respuesta Argumentar por qué.

_____

_____

_____

3- ¿Qué evaluación le otorgaría a la solución propuesta?

Bien____ Regular____ Mal____