Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

"Pruebas Automatizadas para el Proceso Unificado"

T E S I S

Que para obtener el grado de
Maestra en Ingeniería (computación)

P R E S E N T A

Carolina González Nava

Directora: M. en C. Ma. Guadalupe Elena Ibargüengoitia González

MEXICO, D.F MARZO 2012

Agradecimientos

Ésta tesis es el resultado de un trabajo realizado por mucho tiempo, pero no es solo mi triunfo, es el triunfo de todas esas personas que estuvieron conmigo, que de algún modo su presencia fue la fortaleza para terminar este sueño que es mío y de todos.

En primer lugar le agradezco a mi tutora y profesora la M. en C. Guadalupe Elena Ibargüengoitia González por haberme dado la oportunidad de trabajar con ella además de brindarme su apoyo en momentos muy difíciles para mí; además de toda su paciencia y generosidad.

A mis padres, por su apoyo incondicional, los únicos que creen en mí en todo momento no importando las circunstancias, deseo decirles: este logro es un beneficio para mí pero es una victoria para ustedes. Ustedes lograron darlo todo para que no me faltara nada y pudiera dedicarme de lleno a mis estudios, lucharon contra todo para procurar mi buena educación, me enseñaron que aparte de los estudios hay más en la vida por ver, me han educado con valores, principios y las dos caras de la moneda de todas las eventualidades de la vida. Les agradezco por haberme concebido, educarme, brindarme de una infancia feliz y por seguir a mi lado.

A mi querida hermana Cris, le doy las gracias por ser la razón de mi conciencia, por ayudarme a ver más allá de lo que mis ojos ven y por brindarme su opinión que respeto y aprecio.

A mis amigos siempre presentes y dispuestos a darme su ayuda. Mayela, siempre animándome, la fuerza que reflejas es la misma que me ha contagiado para ser más fuerte como persona. David, alegre y carismático, eres mi ejemplo para decir siempre se puede, cuándo se quiere. A ti, Cin, por ser quién eres, de criterio objetivo, pensamiento neutral, observadora, con opinión propia, muy inteligente y consiente de la salud; me enseñaste que ante todo, lo primero es la salud física y mental. Héctor, eres un amigo leal y lo que me has enseñado es la infinita tolerancia al prójimo, si todos fuéramos más tolerantes muchas cosas se resolverían. Javi, gracias por siempre estar ahí, por prestarme tu hombro cuándo lo necesité, por darme porras en mis peores momentos, por decirme en voz alta lo que ya sé pero no lo digo. César, por tener la paciencia de presenciar mis peores momentos de debilidad y aun así tener la fuerza para levantarme, gracias. Rodrigo, por ayudarme a ver que el fin del este pequeño camino estaba más cerca de lo que yo creía.

Quiero agradecerle a toda la generación 2010-I del IIMAS, ya que realmente considero que nuestra unión y convivencia fue clave para que todos lográramos salir adelante y llegar hasta donde estamos ahora.

RESUMEN

Hoy en día existe una alta demanda para la creación de software que este especializado en el negocio de interés (venta de artículos, automatizar la administración y gerencia de la empresa o negocio, etc.) pero está demanda también va acompaña por la liberación pronta del software lo que implica que el periodo de pruebas debe ser más rápido.

Este hecho nos lleva a pensar en formas para realizar las pruebas de una manera rápida pero que sea eficaz garantizando la calidad del software. Con esta idea es por la que nacen las herramientas de pruebas automatizadas. Herramientas de software que prueban programas que son parte de un sistema en desarrollo permitiendo así tener un parámetro de medición y mejorar, según el caso lo permita, los costos en tiempo y esfuerzo de la elaboración y ejecución de pruebas.

En esta propuesta se plantea seguir el Proceso Unificado (PU) agregando como deberían de implementarse las pruebas automatizadas, desde asignación de responsabilidades en los roles ya definidos por el mismo PU, agregar nuevos apartados en los artefactos y finalmente se explica cómo se realiza la implementación con dos herramientas de automatización (JUnit y Selenium), está parte de la implementación varía con respecto de la herramienta por lo que resulta difícil generalizar los pasos para la implementación sin embargo se abarca en gran medida cómo podría realizarse una implementación.

Los pasos para la automatización, aquí descritos, se encuentran inmersos en el flujo de trabajo de pruebas definido por el PU, funcionan no importando que tipo de sistema se esté desarrollando ni está ligado a una plataforma específica ya que no son pasos dependientes de un software sino que forman parte del mismo PU.

ÍNDICE GENERAL

INTRODUCCIÓN

PROBLEMÁTICA

En todo sistema de software es necesario aplicar pruebas para garantizar la calidad del software. Por lo que la fase de pruebas de un sistema es muy importante pero por diversas cuestiones, el tiempo que se le dedica a las pruebas siempre se ve disminuido o hay que aplicar las pruebas lo más rápido posible para poder entregar el sistema en tiempo.

En esta fase de pruebas, se deben tener resultados del sistema lo más pronto posible ya que de existir un defecto es necesario corregirlo lo más pronto posible.

Finalmente, la elaboración de pruebas de manera tradicional consume muchos recursos (tiempo, personal y capital) y lo deseable es la optimización de éstos con el mejor provecho tanto para desarrollador, sistema y cliente.

OBJETIVOS

Establecer cómo automatizar las pruebas siguiendo los pasos definidos por el Proceso Unificado y conseguir la automatización con herramientas de código abierto y/o software libre.

Definir una guía para el manejo de algunas herramientas dependiendo el enfoque de la prueba (se maneja únicamente pruebas unitarias).

ESTRUCTURA DE LA TESIS

En el capítulo 1 se explican brevemente los modelos de proceso de software enfatizando las fases o etapas de las que estos consisten. Esto con el fin para ver sus características principales y al final seleccionar una metodología a utilizar en la tesis, describiendo el porqué de su utilización. Después se da una explicación más detallada del Proceso Unificado, estableciendo los antecedentes de qué es y en qué consiste el Proceso Unificado. Se describen todos los flujos de trabajo para posteriormente enfocarnos en el flujo de trabajo de pruebas. En el flujo de trabajo de pruebas se describen las diferentes actividades que hay que realizar, los artefactos que se generan y los roles que desempeñan cada una de las actividades. En seguida, se describen los tipos más importantes de pruebas existentes. Posteriormente se explica, qué son las pruebas automatizadas, que dentro de todas las pruebas que se le pueden aplicar a un sistema cuáles son las que son automatizables y finalmente se habla sobre las ventajas y desventajas que presenta la automatización de pruebas. Se finaliza con una explicación sobre qué son las herramientas de automatización, se mencionan algunas herramientas clasificadas según su enfoque en la prueba. Después se da una breve reseña del software que se utilizó para el desarrollo de esta tesis.

En el capítulo 2 se aborda la automatización en cada parte del flujo de pruebas para después fijar los pasos a seguir para poder llevar a cabo la automatización de las pruebas. Se describen cinco pasos a seguir los cuales abarcan la captura de requisitos y como llevar su registro, la realización de la estrategia del plan de pruebas, mientras que a la vez de dividen las pruebas en pruebas automatizables y no automatizables, se plantean los requerimientos que debe cumplir la plataforma de pruebas, se establece el control de las pruebas desarrolladas, ejecutadas y el avance de su elaboración, también se lleva el registro de los casos prueba y de los defectos y su estado, finalmente se describen las responsabilidades que adquieren los roles mencionados en el capítulo 2.

En el capítulo 3 se tiene un caso de estudio donde se aplica todo lo planteado en los capítulos anteriores. Se describe paso por paso como se realizó la implementación de las pruebas automatizadas y cuáles son los resultados obtenidos.

CAPÍTULO 1: CONCEPTOS GENERALES

RESUMEN

Se da un breve marco histórico de los modelos de proceso de software y cuál es su principal objetivo. Posteriormente se detallan algunos de los modelos para el proceso de pruebas, como son el modelo en cascada, en espiral, en prototipos, Proceso Unificado; y además de incluir algunas metodologías ágiles cómo: Scrum, cristal y extreme programming. Se presenta el criterio que se consideró para seleccionar una de las metodologías y utilizarla en el desarrollo de esta tesis. Se explica detalladamente en que consiste el Proceso Unificado, describiendo sus fases y sus flujos de trabajo para después centrarse en el flujo de trabajo de pruebas; detallando los artefactos, actividades y roles involucrados. En seguida se menciona una clasificación de pruebas que será la manejada a lo largo de este trabajo. Proseguimos con introducir una definición de pruebas automatizadas, porqué es bueno automatizar, qué se debe automatizar, y las ventajas y desventajas que presentan. Finalmente se mencionan algunas de las herramientas existentes para la automatización de pruebas, clasificadas en funcionales y de carga, enfatizando en las utilizadas en esta tesis.

1.1 MODELOS DE PROCESO DE SOFTWARE

Los modelos de proceso de software son introducidos en 1970, para poder un orden y un mejor entendimiento al proceso de software. En un modelo de proceso de software, generalmente se describen tareas y artefactos del desarrollo de software. Se propone una clasificación para los modelos en: abstracto y concreto. Los modelos de proceso abstractos definen la forma de cómo organizar el desarrollo de software (modelo de prototipos, en espiral). En los modelos de proceso concreto se establecen todos los aspectos que son necesarios para crear una instancia del modelo de proceso de un proyecto (proceso unificado). Esto quiere decir que se definen todas las actividades, roles y artefactos que se necesitan en un proyecto (1).

Modelo en cascada.

El primer modelo de procesos en identificar las actividades principales en el desarrollo de software. En la Figura 1.1 se puede observas las etapas del modelo en cascada.

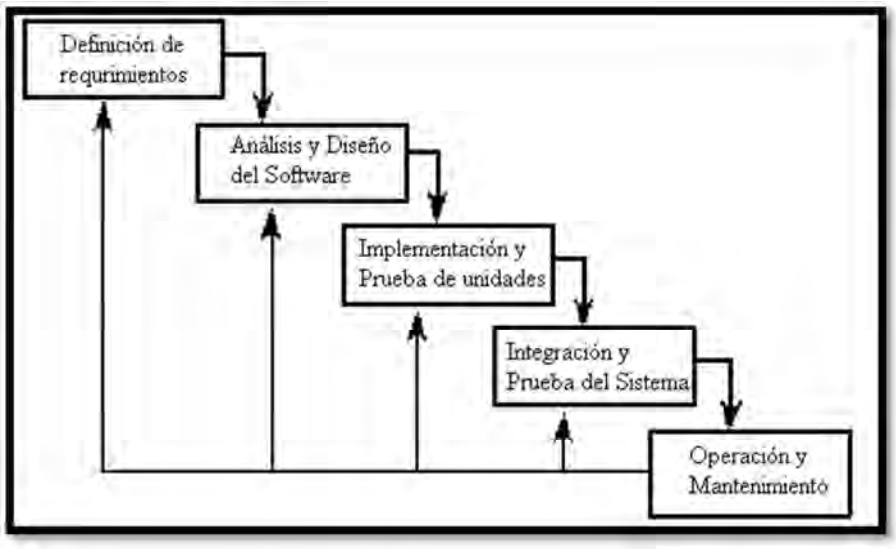


FIGURA 1. 1 MODELO EN CASCADA
Imagen tomada de (2) pág. 62, Figura 4.1 El ciclo de vida de software.

Etapas del modelo en cascada (2)

1. **Definición de requerimientos**. Se definen las características que se desea que tenga el sistema como el objetivo, la meta y las restricciones; todo consultando al cliente.
2. **Análisis y diseño del software**. En el análisis Se dividen los requerimientos en hardware y software, y se define la arquitectura completa del sistema. En el diseño se identifican y describen las abstracciones esenciales del sistema y las relaciones que existen entre ellas.
3. **Implementación y prueba de unidades**. Se implementan los diferentes módulos del sistema los cuáles se prueban verificando que cumplan con la especificación solicitada.
4. **Integración y prueba del sistema**. Los módulos del sistema se integran y se prueba la funcionalidad de un sistema cómo un todo. Después de haber realizado las pruebas, y corregido los defectos del sistema, es entregado al cliente.
5. **Operación y mantenimiento**. Se instala el sistema y entra en funcionamiento práctico. El mantenimiento es la corrección de defectos no encontrados en las etapas anteriores y mejorar el sistema con la llegada de nuevos requerimientos.

En el modelo en cascada no se avanza a la siguiente etapa hasta que no se haya terminado por completo la que se encuentra en elaboración. Una de sus ventajas principales es que la documentación se elabora en cada una de las etapas y el modelo en cascada solo se recomienda cuándo se han entendido perfectamente los requerimientos y estos no tendrán un cambio fuerte durante el desarrollo.

Modelo de prototipo.

El objetivo del modelo de prototipo es obtener una idea general del funcionamiento completo del sistema. La principal ventaja de este modelo es que el diseño y la

implementación tienen retroalimentación de los usuarios cercanos al proyecto. En la Figura 1.2 se muestra el diagrama para el modelo de prototipo.

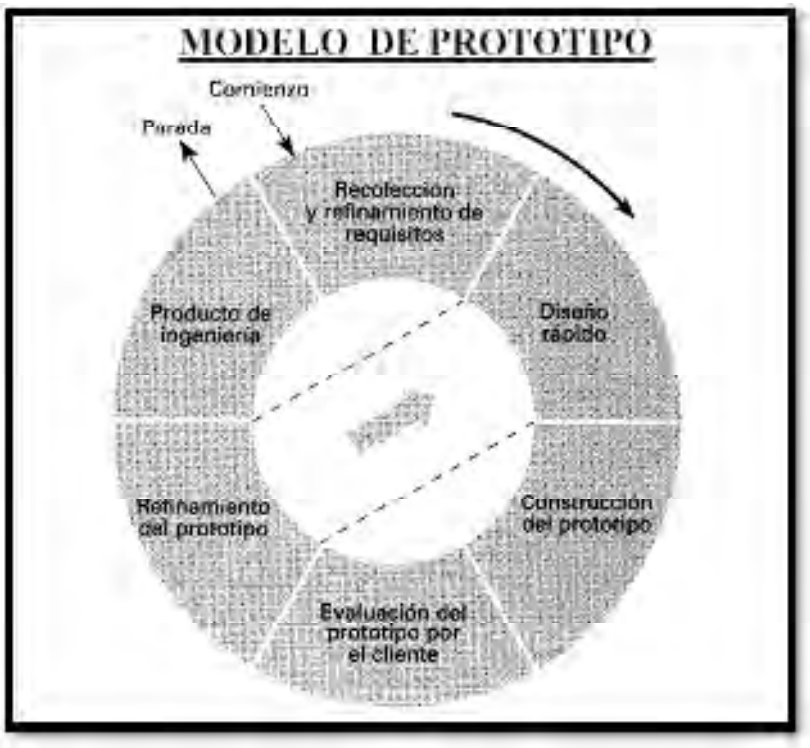


FIGURA 1. 2 MODELO DE PROTOTIPO
Imagen tomada de (3) Modelo de prototipo.

Etapas del modelo de prototipo (4)

1. **Recolección y refinamiento de requisitos**. Se determinan cuáles son los requisitos de entrada y salida, ignorando los demás requisitos.
2. **Diseño rápido**. Al tener las entradas y salidas se establecen como deberían de estar relacionadas.
3. **Construcción del prototipo**. El prototipo consiste únicamente en las interfaces del usuario.
4. **Evaluación del prototipo por el cliente**. El cliente da su "visto bueno" o las correcciones que deben hacerse al prototipo presentado.
5. **Refinamiento del prototipo**. De existir modificaciones al prototipo, se vuelve a realizar el diseño con los requisitos nuevos o modificados del cliente.
6. **Producto de ingeniería**. Es la entrega del sistema completo y funcional.

Modelo en espiral.

El modelo en espiral en lugar de definir el proceso de desarrollo de software como una secuencia de pasos a seguir, lo hace como una espiral. Donde cada ciclo de la espiral representa una fase del proceso y a la vez cada ciclo está dividido en cuatro sectores. La

característica principal del modelo es que está centrado en los riesgos del proyecto. En la Figura 1.3 se muestra la representación gráfica del modelo en espiral (2).

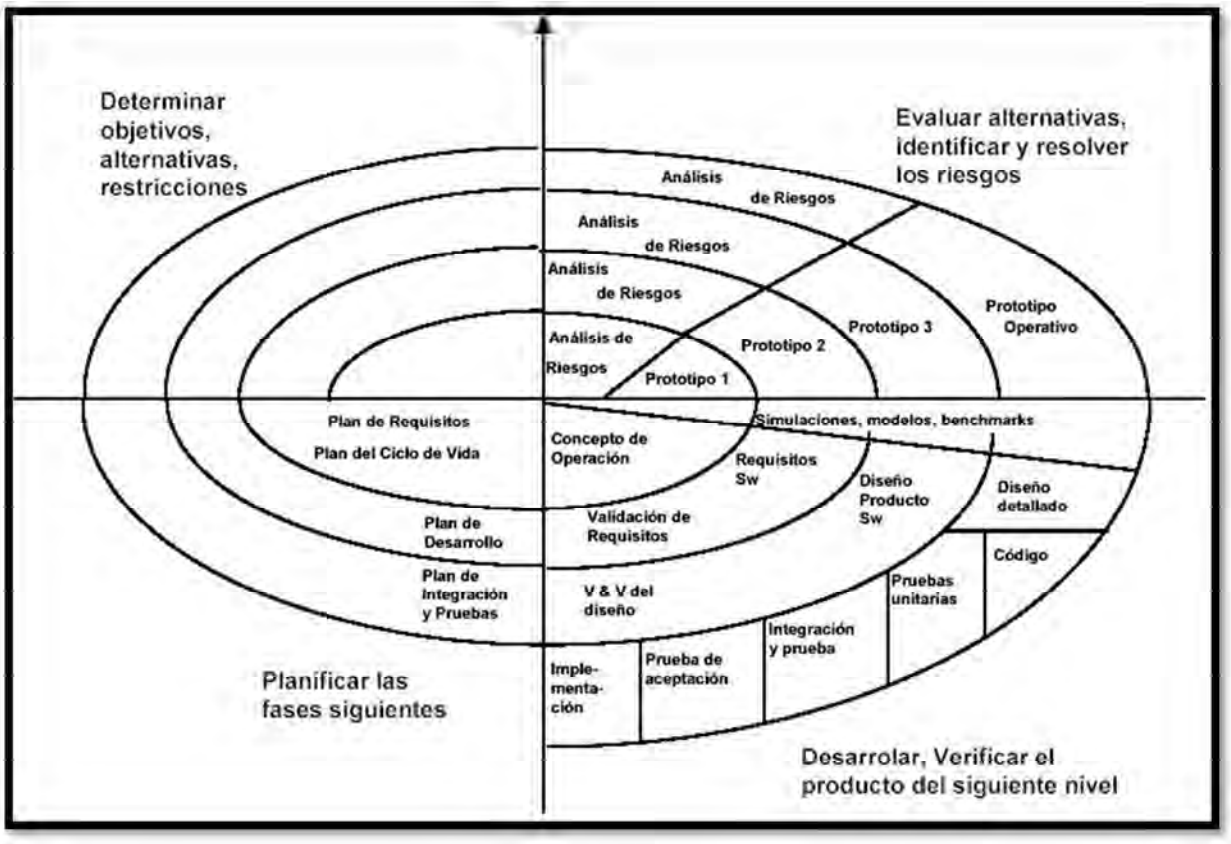


FIGURA 1. 3 MODELO EN ESPIRAL
Imagen tomada de (5).

Sectores del modelo en espiral (2)

1. **Definición de objetivos**. Se definen los objetivos específicos del proyecto además de las restricciones del proceso y del producto. Se identifican riesgos, dando prioridad a los mismos, y se crea un plan de contingencia.
2. **Evaluación y reducción de riesgos**. Se realiza un análisis detallado de los riesgos e identifican los pasos para amortiguar dichos riesgos.
3. **Desarrollo y validación**. Después de evaluar los riesgos, se selecciona un modelo que se adecúe más para aminorar los riesgos principales.
4. **Planificación**. Se revisa el proyecto para determinar si puede continuar o es necesario regresar a algún paso posterior en la espiral.

Proceso Unificado[1].

[1] Se hablará más detalladamente del Proceso Unificado en la sección 1.2 de esta tesis.

Es un proceso iterativo e incremental. Consta de cuatro etapas las cuáles son: inicio, elaboración, construcción y transacción. En el proceso unificado es necesario detallar los artefactos y los roles que existirán a lo largo del desarrollo.

Metodologías ágiles.

El método ágil espera que durante la implementación se ajusten los requerimientos. El objetivo de esto es poder ofrecer lo más pronto posible una funcionalidad del sistema con los módulos más importantes para el usuario (1).

Características comunes de los modelos ágiles (1):

- Desarrollo iterativo en ciclos cortos (no mayores de 3 meses).
- El equipo de proyecto es pequeño (entre 6 y 10 personas).
- El cliente se integra al equipo el mayor tiempo posible.
- El trabajo es organizado y asignado por el propio equipo
- La documentación es reducida al mínimo.

Las metodologías ágiles más representativas son (6):

Scrum. Recomendada para cambios constantes de requisitos del proyecto. Las características principales de Scrum es la realización de iteraciones llamadas *sprints* (duración aproximada de 30 días), al final de cada sprint se obtiene un incremento que en forma de ejecutable que se muestra al cliente, y las reuniones contantes y diarias a lo largo del proyecto para coordinar e integrar.

Metodologías Crystal. Enfocadas a las personas que integran el equipo de desarrollo y a la reducción al mínimo de los artefactos. Siendo centrado en el equipo se invierte en la mejora de habilidades de los integrantes. Se establecen políticas de trabajo, las cuáles dependerán del tamaño del equipo, y se implanta un código de colores como Crystal Clear (de 3 a 8 miembros) y Crystal Orange (de 25 a 50 miembros).

Extreme Programming XP. Es una metodología ágil en la que su clave de éxito es la de fomentar las relaciones personales dentro del equipo de desarrollo de software, promoviendo el trabajo en equipo y procurando el aprendizaje de los integrantes del equipo. Se requiere de constantes y fluida comunicación con el cliente. Recomendada para cuando se tienen requisitos imprecisos y volubles.

De los modelos mencionados anteriormente se decidió trabajar en esta tesis con el Proceso Unificado. Se escogió este modelo porque engloba las fases principales que se manejan en todos los modelos que son la captura de requisitos, el diseño, elaboración del sistema y la parte de pruebas para después entregar el sistema al cliente. Aunque las otras metodologías también presentan su grado de iteración solo el Procesos Unificado realiza la iteración en cada una de sus fases, por lo que resulta más flexible para el desarrollo de software. No se prefirió una metodología ágil ya que en estos se omite la documentación o solo realizan la

mínima necesaria y la propuesta presentada es para que forme parte de la documentación. En seguida se detallan las características del Proceso Unificado.

1.2 EL PROCESO UNIFICADO

El Proceso Unificado (7)[2][3], es un proceso para el desarrollo de software y consiste en la transformación de los requerimientos del cliente a un sistema de software. El Proceso Unificado se basa en tres ideas fundamentales: los *casos de uso*, *centrado en la arquitectura*, y que es *iterativo e incremental*. Y para modelar los productos, el Proceso Unificado hace uso del Lenguaje de Modelado Unificado, UML.

Los *casos de uso* son fragmentos de funcionalidad que deben proporcionar al usuario un valor o servicio y son utilizados primordialmente para la captura de requisitos donde se pretende encontrar los requisitos que, al ser implementados resulten beneficiosos para el usuario. Con los casos de uso se pretende dar una representación adecuada de los requisitos para los usuarios, clientes y desarrolladores. Para el sistema un *actor* es una persona o sistema que interactúa con el sistema que está en desarrollo.

El Proceso Unificado está centrado en la *arquitectura* porque es la que define la estructura del software para el manejo e implementación de los casos de uso. La arquitectura describe los aspectos físicos y dinámicos del sistema. Sin embargo, la arquitectura y los casos de uso no funcionan ni se desarrollan de manera independiente sino que trabajan en conjunto donde los casos de uso definen la función y la arquitectura la forma. La importancia de tener una arquitectura es para la organización del software.

Dado que la arquitectura y los casos de uso evolucionan juntos, la forma de mantener el equilibrio entre el crecimiento de ambos es por medio del tiempo. Para lograr este control se pretende que las diferentes actividades que surgen a lo largo del Proceso Unificado se realicen dividiendo el proyecto según el avance del sistema y solo se avanza si se está satisfecho con el paso que se ejecutó. Es por esto que se dice que el Proceso Unificado *es iterativo e incremental*, la división del proyecto en pequeños proyectos los cuales resultan ser una iteración y el resultado al finalizar la iteración es un incremento del software.

La vida de un sistema de software, como lo define el Proceso Unificado, se encuentra dividido en ciclos y cada ciclo a su vez se divide en cuatro fases: inicio, elaboración, construcción y transacción. Y finalmente cada fase se subdivide en iteraciones.

Fase de inicio. Se comienza por una definición del proyecto para poder obtener los casos de uso primordiales que ayudarán a establecer una arquitectura candidata que finalmente sirve para poder dar una estimación aproximada del alcance del proyecto y se identifican los

[2] La definición del Proceso Unificado, sus fases, los flujos de trabajo, el detalle del flujo de trabajo de pruebas, actividades, artefactos y roles descritas en el flujo de trabajo de pruebas son un resumen de la referencia (7) de los capítulos 1 y 11.

[3] Ya que no se pretende profundizar en el tema comparando puntos de vista sobre el Proceso Unificado solo se tomó una referencia para dar el marco teórico general.

riesgos para poder mitigarlos. El objetivo de esta fase es entender lo qué se va a construir y este entendimiento debe ser además un acuerdo mutuo entre el cliente y los desarrolladores de lo que debe y no debe hacer el sistema.

Fase de elaboración. Se detallan los casos de uso y se diseña la arquitectura para obtener como resultado una línea base de la misma. El resultado esperado de esta fase es tener estabilidad en los casos de uso principales, la arquitectura y el plan del proyecto; además de tener control de los riesgos identificados. En esta fase se toma la decisión de si el proyecto es viable o no.

Fase de construcción. Se crea el producto usando la arquitectura definida en la fase de elaboración, se van añadiendo las funcionalidades solicitadas por el cliente hasta tener un producto funcional en una versión beta. Es posible hacer mejoras y sugerencias a la arquitectura.

Fase de transacción. Se prueba la versión beta del sistema detectando defectos y deficiencias, se corrigen los errores encontrados, si es posible se agregan nuevas funcionalidades que no impacten en gran medida al sistema, se entrena el usuario y finalmente se libera el sistema.

El Proceso Unificado consta de cinco flujos de trabajo fundamentales que se realizan a lo largo de todas las fases en mayor o menor medida dependiendo el desarrollo del sistema software.

Flujo de trabajo de requisitos. Es la captura de los requisitos asegurando que los desarrolladores construyan un sistema correcto para el cliente. Para lograr esto se debe llegar a un acuerdo entre desarrolladores, cliente y usuarios de lo que debe hacer el sistema. La descripción de los requerimientos se realiza en lenguaje del cliente y ayuda a planificar las iteraciones.

Flujo de trabajo de análisis. En este flujo se refinan y estructuran los requisitos para que el equipo examine y revise los requisitos, los comprenda y así pueda desarrollarlos correctamente.

Flujo de trabajo de diseño. El objetivo de este flujo es definir la arquitectura o estructura del software para dejarla en términos comprensibles para los programadores, con esto se pretende fijar un punto de partida para iniciar la implementación. Además de que se define al sistema como una composición de partes que lo vuelven manejable.

Flujo de trabajo de implementación. Es llevar todo lo que se tiene documentado a términos de código, es decir, se construye el sistema de software. También se planea la integración de cada uno de los componentes ejecutables y se realizan las pruebas unitarias de cada uno.

Flujo de trabajo de pruebas. En esta parte se definen las pruebas, los procedimientos y las métricas de evaluación que se van a emplear para la corroboración de que tan correcto es el sistema de software y la calidad del mismo en términos de los requisitos.

1.2.1 FLUJO DE TRABAJO DE PRUEBAS

El flujo de trabajo de pruebas tiene como objetivos principales la planificación de las pruebas de integración y del sistema en cada iteración, el diseño e implementación de las pruebas (abarcando desde los casos de prueba, procedimientos de prueba y los componentes de prueba) y finalmente la realización de las diferentes pruebas.

Durante el flujo de trabajo de pruebas se deben obtener documentos que representen la información necesaria para planear, elaborar, diseñar, construir, y ejecutar las pruebas, los cuales son conocidos como *artefactos*. En el equipo de pruebas deben de asignarse responsabilidades entre los integrantes para poder organizarse entre ellos y esta asignación se realiza mediante los *roles*. Y finalmente, a lo largo del flujo de trabajo se deben de llevar a cabo las *actividades* necesarias para poder realizar las pruebas.

1) Artefactos

En el flujo de pruebas hay varios artefactos que se deben obtener al finalizar alguna actividad como el *modelo de pruebas*, definir los *casos de prueba*, los *procedimientos de prueba*, los *componentes de prueba*, el *plan de prueba*, *defectos* y *evaluación de la prueba*.

- **Modelo de pruebas.** En el modelo de pruebas se describe cómo se van a probar los componentes ejecutables y cómo deben ser probados aspectos específicos del sistema. El modelo de pruebas debe ser una colección de casos de prueba, procedimientos de prueba y componentes de prueba.

- **Caso de prueba.** En el caso de prueba se debe especificar la forma de probar el sistema considerando la entrada al sistema y las condiciones a las que está sujeto. Lo que se va a probar es un requisito o una colección de éstos. Los casos de prueba pueden especificar cómo probar un caso de uso y se verifica que el resultado de la interacción entre actores y el sistema sea la deseada. Debe satisfacer las precondiciones y postcondiciones de los casos de uso y además de seguir la secuencia de acciones definidos en el caso de uso; o bien puede especificar como probar la realización de un caso de uso y debe verificar la interacción entre los componentes que implementa dicho caso de uso. En cualquiera de los dos casos se deben verificar los diferentes escenarios que puede presentar un mismo caso de uso.

- **Procedimiento de prueba.** El procedimiento de prueba es una lista los pasos a seguir para que una persona pueda ejecutar la prueba ya sea de manera manual o cómo interaccionar con una herramienta para poder ejecutar la prueba.

- **Componente de prueba.** El componente de prueba automatiza uno o más procedimientos de prueba. Los componentes de prueba pueden ser desarrollados en un lenguaje de guiones o lenguaje de programación o bien ser implementados en una herramienta de automatización. Un componente de prueba proporciona entradas de prueba, controla y monitorea la ejecución del mismo e informa de los resultados de la prueba. Generalmente, los componentes de prueba son llamados en inglés test drivers o test scripts.

- **Plan de prueba.** Para poder llevar a cabo la correcta elaboración y ejecución de las pruebas se debe tener un plan de prueba en el que se describen estrategias, recursos y la planificación. La estrategia para este plan es la definición del tipo de prueba a realizar para cada iteración y los objetivos de la prueba, se establece el nivel de cobertura de la prueba, el nivel de cobertura del código y un porcentaje de pruebas para determinar que la prueba fue exitosa o no.

- **Defecto.** A lo largo del proceso de prueba se detectan defectos, donde el defecto es una anomalía en el sistema y se debe controlar y resolver.

- **Evaluación de prueba.** Al final de la ejecución de la prueba se debe tener una evaluación de prueba donde se analizan los resultados de la prueba como la cobertura de la prueba y de código, además del estado de los defectos.

2) Roles

En el proceso de prueba hay varios roles:

- El **ingeniero de prueba** es el responsable de mantener la integridad del modelo de pruebas y es el encargado de planear las pruebas, también debe seleccionar y describir los casos de prueba y los procedimientos de prueba y finalmente realiza la evaluación de las pruebas de integración y del sistema.

- El **Ingeniero de componentes** es el responsable de seleccionar y realizar los componentes de prueba, que es la automatización de los procedimientos de prueba.

- El **Ingeniero de pruebas de integración** es el responsable de realizar las pruebas de integración y de documentar los defectos encontrados mediante las pruebas.

- El **Ingeniero de pruebas de sistema** es el responsable de realizar las pruebas del sistema necesarias sobre la construcción que muestra el resultado de una interacción completa. También es el encargado de verificar las interacciones entre actores y el sistema. Y de igual manera tiene que documentar los defectos encontrados durante estas pruebas.

3) Actividades

- **Planificar la prueba.** Se describe una estrategia, la cual debe contemplar el tipo de prueba a realizar, cómo y cuándo debe ejecutarse la prueba y el cómo se sabe que la prueba resultó exitosa. Se deben estimar los requisitos para el esfuerzo de la prueba y se planifica el esfuerzo de la prueba.

Generalmente se prueban casos principales y requisitos porque no es posible probar un sistema al 100% porque revisar todos los aspectos representa inversión alta en tiempo y costo.

- **Diseñar la prueba.** El diseño de la prueba es la identificación de los casos de prueba y de los procedimientos de prueba. El diseño de los casos de prueba implica realizar la verificación de la interacción entre componentes y determinar los casos de prueba con solapamiento mínimo.

El diseño de los casos de prueba del sistema debe revisar el funcionamiento del sistema como un todo, realizar combinaciones entre los casos de uso instanciados bajo diferentes condiciones donde se vigila el funcionamiento en paralelo, influencia mutua y el uso de los recursos. Dados los casos de prueba se debe estructurar el procedimiento de prueba para cada caso de prueba o si son semejantes los casos de prueba se puede reutilizar el procedimiento de prueba.

- **Implementar prueba.** La implementación de la prueba consiste en llevar a cabo los procedimientos de prueba y cuando se hace uso de una herramienta se especifican las acciones a seguir para la ejecución de la prueba.

- **Realizar pruebas.** Para la realización de la prueba hay que seguir una serie de pasos:

    1. Realizar las pruebas.

    2. Comparar los resultados obtenidos con el esperado o determinar el significado de los resultados de las pruebas que no coinciden con el valor esperado o no se pudo determinar un resultado esperado.

    3. Informar sobre defectos en los componentes al Ingeniero de componentes.

    4. Informar de defectos sobre el sistema a los diseñadores de prueba (ingenieros de prueba).

- **Evaluación de la prueba.** Se evalúan los esfuerzos de la prueba y se analizan los resultados obtenidos contra los estimados. Los ingenieros de prueba preparan métricas que permiten evaluar la calidad del software y qué cantidad de pruebas es necesario hacer. Generalmente, se fijan dos métricas principales:

    - La completitud de la prueba. Está métrica indica el porcentaje de los casos de prueba probados y el porcentaje de código probado.

    - La fiabilidad. Se obtiene por medio del análisis de las predisposiciones de los defectos encontrados y las tendencias de las pruebas que se ejecutan y dan como resultado el valor esperado.

1.2.2 TIPOS DE PRUEBAS

Las pruebas se pueden clasificar de varias maneras. Una de ellas las agrupa en dos grandes categorías y, a veces, una tercera como combinación de ellas.

Pruebas de **caja negra**. En este tipo de pruebas no se requiere saber qué es lo que pasa internamente en el programa sino tan solo suministrar datos y estudiar la salida del programa. La estrategia de la prueba está basada, generalmente, en los requerimientos y las especificaciones del cliente (8). Los defectos que se intentan encontrar por medio de este tipo de prueba son (9):

1. Detectar funciones incorrectas o faltantes.
2. Defectos en las interfaces.
3. Defectos en el acceso a la base de datos.
4. Problemas de desempeño.
5. Problemas al inicializar o terminar el sistema.

Es necesario tener un control de versiones y llevar un registro de los defectos encontrado, esto debido a que existen causas para no haber detectado algunos defectos en un principio como (9):

- Defectos colaterales. Son los que aparecen después de haber resuelto los defectos en una versión previa.
- Defectos secundarios. Son aquellos que aparecen solo después de corregir una sección, ya que no se tenía paso a ellos.

Pruebas de **caja blanca**. Se trata de probar la mayor cantidad de código posible, tratando de generar casos de prueba que al ser ejecutados abarquen el mayor número de líneas de código y que al menos cada línea sea ejecutada una vez (8). Para realizar pruebas de tipo caja blanca se considera la estructura del código, los casos de prueba al ser generados a partir de la estructura del código no es posible hacerlos antes de que sistema haya sido terminado. Esto también trae una gran desventaja ya que si cambia la estructura de los datos o los algoritmos, habrá que realizar los casos de prueba nuevamente (10).

Pruebas de **caja gris**. Dado que las pruebas de tipo de caja negra pueden dejar escapar algunos errores, como los llamados falso positivo, donde se genera un error interno en la ejecución del programa (como almacenar una variable en un registro equivocado en la base de datos) y sin embargo la prueba no detecta el error y da un resulta positivo. La prueba de caja gris permite acceso al código y a los algoritmos para la generación del caso de prueba pero solo se enfoca en algunas áreas del sistema que sean más susceptibles de fallar (8).

1.3 PRUEBAS AUTOMATIZADAS

¿Qué son las pruebas automatizadas?

Las pruebas automatizadas de software se definirán como la utilización de herramientas para la captura, grabación y/o reproducción de la prueba (11). Una definición más formal descrita en (11) sería:

"*Aplicación e implementación de tecnología de software a lo largo de todo el ciclo de vida de pruebas de software con el objetivo de mejorar la eficiencia y efectividad del ciclo de vida de las pruebas de software*"

Para decir que una prueba automatizada de software ha sido implementada correctamente debe (11):

- Correr en múltiples computadoras.
- Correr en diferentes sistemas operativos.
- Soportar integración de múltiples herramientas de pruebas.
- Soportar nuevas aplicaciones bajo desarrollo y aplicaciones existentes.

Ahora bien, tenemos que contestar a la pregunta ¿Por qué automatizar? En estos tiempos el cliente quiere software funcional que sea liberado más rápido, y al tener que librar el software más rápido se acorta el tiempo disponible para prueba y hay mayor frecuencia de necesitar probar el software y no hay personal que cubra ese trabajo por lo que al automatizar se espera que haya una reducción de costo y tiempo.

El primer factor que se mejora con la automatización de pruebas es el *tiempo*, aunque es necesario invertir un poco más de tiempo en la construcción de la prueba, una vez que ésta está hecha se puede ejecutar tantas veces como sea necesario. La reducción del tiempo se ve reflejada en el momento de la ejecución de la prueba porque una prueba automatizada puede ser corrida sin la necesidad de que haya un intermediario que la esté ejecutando, por lo que la prueba se ejecuta en tiempo constante y sin contratiempos ocasionados por el intermediario (como: comer, dormir o realizar otra tarea). Al contrario de una prueba que se ejecuta de manera manual ya que suele invertírseles mucho tiempo y mano de obra para ejecutarla, además del hecho de que si es necesario repetir la prueba se duplican los costos.

Al estar las pruebas desarrolladas con una herramienta de software se disminuye el contacto humano y por lo tanto se evita el error que las personas pueden llegar a generar (11).

Teniendo las pruebas almacenadas en una herramienta, se podría decir, que se posee un banco de pruebas el cual puede ser reutilizado todas las veces que sea requerido y si algún otro proyecto llega a tener semejanza con alguna prueba almacenada en el banco de pruebas es posible tomar la prueba, realizar la modificaciones pertinentes y ejecutar la prueba; de esta forma se reutiliza el código empleado y se disminuyen tiempos de construcción de prueba para futuros proyectos (12).

Por otro lado, en la elaboración de un sistema de software se genera más de una versión, por mantenimiento o mejora del mismo, y al tener que realizar las pruebas nuevamente consumen más tiempo del que se está llevando en el desarrollo, ya que hay que considerar que a pesar de que solo se modifique una característica, las demás características que no tienen relación aparente deben ser probadas porque es posible introducir errores al modificar el código y por la interacción entre diferentes fragmentos del mismo. Con las pruebas automatizadas se puede volver a repetir el conjunto de pruebas que garanticen la calidad del software.

Sin embargo hay factores que considerar para decidir si es realmente necesario automatizar la prueba.

La construcción de pruebas automatizadas resulta ser una tarea muy costosa en tiempo y en mano de obra. Debido a que es necesario realizar la definición de los casos de prueba y la selección de estos casos prueba es laboriosa por el hecho de tener que elegir solo aquellos que den una aportación valiosa a la prueba. Otro factor que incrementa el tiempo y el esfuerzo empleado es según la cobertura a obtener con la prueba, entre más esfuerzo se emplea resulta ser más costosa y entre menos cobertura es menos costosa pero se sacrifica la calidad del software (11).

Para determinar la calidad del software es necesario definir las métricas con las cuales será evaluado y se definen a partir de los requerimientos y especificaciones del sistema.

¿Qué se puede automatizar?

Lo que es posible automatizar incluye la simulación de la actividad del actor donde se pueden probar dos cosas, que la secuencia de eventos sea la correcta y el sistema no haga cosas que no debe hacer y también la carga de trabajo que es posible soportar con más de un actor interactuando con el sistema.

Es posible automatizar módulos del sistema probando que cada módulo esté funcionando bien. Generalmente las pruebas se realizan sobre una clase y se verifican los métodos que hay en ella pero el tamaño del módulo puede variar.

En pocas palabras la automatización en la que me enfocaré será a la ejecución de la prueba, ya que el diseño e implementación de la prueba son necesariamente elaborados por un Ingeniero de pruebas. Ahora bien, no en todos los casos es posible automatizar toda la prueba por lo que se busca automatizar lo más que se pueda.

Ventajas y desventajas para la automatización

La automatización ofrece muchas ventajas porque es posible ampliar la cobertura de las pruebas incrementando el número de escenarios a analizar. También tenemos un incremento en el tiempo efectivo de la prueba porque se puede dejar ejecutando la prueba

en horarios no laborales y además brinda la oportunidad a la persona que está realizando la prueba de realizar otras tareas mientras se está ejecutando la prueba.

Y finalmente las pruebas automatizadas permiten la verificación rápida y eficiente de los errores que hayan sido arreglados, esto se debe a que la prueba se repite exactamente como se hizo la primera vez (13).

Sin embargo, también hay desventajas para las pruebas automatizadas a veces es difícil encontrar una herramienta que resulte adecuada para lo que se desea probar y por ende habrá que realizar un herramienta propia que quizás solo sirva para ese proyecto. Por otro lado si llega a existir la herramienta que funcione para lo que se desea está tendrá algún costo.

1.4 HERRAMIENTAS PARA LA AUTOMATIZACIÓN

Las herramientas para la automatización de pruebas son software especializado para realizar tareas usualmente hechas por el hombre para evaluar el correcto funcionamiento de un sistema en desarrollo y determinar la calidad que este poseerá cuando sea terminado. Existen diversas herramientas para poder realizar estas tareas, tantas como tipos de pruebas existentes algunas como las más importantes son: herramientas para pruebas funcionales y de carga pero también hay herramientas para probar la interfaz de usuario, etc. Finalmente se puede encontrar una cantidad enorme de herramientas disponibles para su uso ya sea de forma gratuita o con un costo. Cabe mencionar que estás herramientas, al ser un software especializado, esté debe instalarse preferentemente en máquinas dedicadas a pruebas y deben de instalarse con ellas los requerimientos mínimos que necesiten para su correcto funcionamiento sin embargos hay algunas que además de tener su parte independiente generaron otra (plug-in) que se acopla a plataformas de desarrollo (IDE).

1.4.1 DIVERSAS HERRAMIENTAS PARA LA AUTOMATIZACIÓN DE PRUEBAS DE SOFTWARE

A continuación se mencionan algunas de las herramientas para la automatización de pruebas existentes en el mercado, esto con el fin de mostrar la gran variedad existente.

Pruebas funcionales

HP Functional Testing software permite crear conjuntos de pruebas funcionales y de regresión. Captura, verifica y realiza repeticiones de las interacciones del usuario automáticamente y ayuda a identificar e informar sobre los defectos de aplicación. Con la aplicación de nuevas construcciones, es necesario actualizar una sola referencia en el repositorio compartido, y la actualización se propaga a todas las pruebas de referencia. Diseñado para las plataformas Oracle, SAP, Seibel, Windows, Delphi, PowerBuilder, ASP .Net y J2EE (14).

SilkTest es utilizado ampliamente en las industrias para realizar las pruebas de regresión. Los creadores de SilkTest aseguran que se crean pruebas robustas al usar está herramienta además de una ejecución rápida. SilkTest automatiza las pruebas de regresión, las pruebas de cross-plataform y las pruebas de localización. Disponible para Windows de forma gratuita por 30 días (15).

Prueba zeta (Zeta Test) 1.0.2.62 se usa para documentar y llevar un registro de los procesos de los flujos de trabajo. Zeta Test tiene como objetivo cubrir los aspectos de la aplicación como el proceso de instalación, la documentación, la usabilidad de la interfaz de usuario y el comportamiento de una aplicación en diferentes sistemas operativos. Disponible para Windows de forma gratuita (16).

TestV (vTest) es una herramienta para las pruebas de regresión y funcionales de aplicaciones web. VTest genera scripts para que sean ejecutados de forma automática en horarios establecidos sin intervención humana. Utiliza JavaScript para generar los casos prueba. Los casos prueba son fácilmente modificables por lo que aumenta la eficiencia a lo largo del desarrollo del sistema. Disponible para Windows en forma de forma gratuita pero limitada, cuenta con versiones profesionales y empresariales con dos tipos de licencias: Dedicated y Floating. Estas versiones cuentan con un costo que va desde los $995 hasta los $2995 dólares (17).

JUnit es un conjunto de bibliotecas utilizadas para hacer pruebas unitarias de aplicaciones Java. JUnit permite realizar la ejecución de clases Java de manera controlada con el fin de comprobar que los métodos se comportan de la forma esperada (18). Es un framework que se puede utilizar como plug-in en Eclipse y NetBeans de manera libre. NetBeans está disponible para Windows, Solaris, Linux y Mac OS (19) y Eclipse está disponible para Windows, Linux y Mac OS (20).

Ruby on rails no es una herramienta para la automatización de pruebas pero utiliza la biblioteca Test::Unit como estándar para crear pruebas de unidad. Cuando se usan los generadores de Rails para crear cualquier módulo rails automáticamente crea una prueba de unidad para dicho modulo. Rails hace uso del modelo MVC que hace más fácil escribir pruebas de unidad. Se encuentra disponible de forma gratuita para Windows y Linux (21).

Pruebas de carga

WAPT es una herramienta para las pruebas de estrés y carga en páginas web. Es capaz de crear usuarios virtuales y sesiones simultáneas para un mismo sitio web. WAPT está diseñado para Windows y cuenta con una versión de prueba de 30 días. Una ventaja que permite es que si ya se ha comprado una versión anterior de WAPT permite la actualización gratuita (22).

Webserver Stress Tool realiza la simulación de peticiones HTTP generando hasta 10,000 usuarios simultáneos. Webserver Stress Tool puede realizar pruebas de rendimiento, de carga, de estrés, de rampa (aumento de usuario en un determinado tiempo para determinar el número de usuarios máximo antes de producir mensajes de error). Cuenta con una versión de prueba gratuita de 30 días y las versiones comerciales que van desde $249.95 a los $2,895 dólares (23).

vPerformer se utiliza para evaluar el rendimiento y la escalabilidad de las aplicaciones web. vPerfomer permite controlar el estado de componentes externos como servidores web, servidores de bases de datos, dispositivos de red, etc., con esto permite identificar los cuellos de botella. Entre algunas de sus ventajas: tiene una interfaz amigable para reducir la curva de aprendizaje, utiliza JavaScript, y se puede realizar el registro en grupo o individual de las peticione HTTP. Esta disponible para Windows en una versión de prueba gratuita. Las versiones comerciales tienen un precio entre los $1,995 a los $5,995 dólares, el precio varía según el número de usuarios virtuales y el tipo de licencia (24).

JMeter es una aplicación de escritorio diseñada para las pruebas de carga de un software cliente/servidor. JMeter puede ayudar en las pruebas de regresión ya que permita la creación de scripts.. Puede ser utilizado para simular una carga pesada para poner a prueba su resistencia o para analizar el rendimiento general en diferentes tipos de carga. Bajo la licencia de Apache Software Foundation, JMeter es de distribución libre. JMeter puede ser usado en las siguientes plataformas Unix (Solaris, Linux, etc.), Windows (98, NT, XP, etc.) y OpenVMS Alpha 7.3+ (25).

WebLOAD es una herramienta de la empresa Radview, Inc para la generación de carga inicialmente diseñada para probar aplicaciones Web. Mediante la simulación precisa del comportamiento de los usuarios, y la predicción de los requerimientos de capacidad, WebLOAD detalla los “cuellos de botella”, restricciones, fallos y puntos débiles de la Aplicación, antes de que lleguen a suponer pérdida de tiempo, de ventas o, lo que es más importante, pérdida de clientes (26). WebLOAD funciona para diferentes versiones de Windows (7, Vista, XP, Windows server 2008) y para Linux. Para ambos sistemas cuenta con una versión libre posible descargar desde la página (27) y una versión profesional la cual varía su costo dependiendo de las características que se deseen (28).

La selección de las herramientas utilizadas en este trabajo fue en base al proyecto que se utiliza para ejemplificar las pruebas. El proyecto fue desarrollado sobre NetBeans por lo que se decidió mantener el entorno de desarrollo para evitar los retrasos por cambio de ambiente e incompatibilidad. Bajo este mismo criterio se seleccionó el uso de JUnit además de que la programación sigue siendo en Java. Finalmente se seleccionó Selenium IDE porque ofrece una interfaz amigable para realizar pruebas sobre la aplicación web y lo hace sobre el navegador de Mozilla Firefox sin ningún problema.

1.4.2 NETBEANS IDE, UNA HERRAMIENTA PARA EL DESARROLLADOR.

NetBeans IDE es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java - pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos (plug-in) para extender el NetBeans IDE.

Para el desarrollo de esta tesis se instaló NetBeans IDE 6.9.1, con fecha de descarga el 25/03/2011 y obtenida en la página http://netbeans.org/downloads/6.9.1/. También se requirió instalar Java SE 6 update 24 (que es el jdk que necesita NetBeans IDE) se descargó el 25/03/2011 en http://www.oracle.com/technetwork/java/javase/downloads/java-se-6u24-download-338091.html. Para instalar NetBeans IDE revisar la página http://netbeans.org/community/releases/69/install_es.html.

Se incluye una guía para la creación de proyectos en Netbeans 6.9.1 disponible en el apéndice 1.

1.4.3 PRUEBAS UNITARIAS EN NETBEANS IDE CON JUNIT

JUnit es un framework que permite realizar pruebas unitarias y controlar las pruebas de regresión para aplicaciones java. NetBeans IDE cuenta con un plug-in que permite la generación de las plantillas para pruebas JUnit, por lo que el programador se concentra en la esencia de la prueba y no en clases intermedias para lograr la ejecución de la prueba.

Las clases intermedias a las que nos referimos son los drivers y los stubs. Un driver para la prueba es un módulo de software que se utiliza para invocar un módulo (el del sistema en desarrollo) y proporcionar entradas de prueba, control y seguimiento de la ejecución, e informar los resultados de pruebas. Y un stub es un programa “dummy” que se utiliza como un sustituto para simular el trabajo de un módulo que aún no se encuentra en desarrollo o no está listo (29).

Para la ejecución de las pruebas se utilizó JUnit 3.8.2 que viene integrado en el proyecto generado por NetBeans IDE junto con JUnit 4.5. Se recomienda desinstalar una de las librerías porque al momento de ejecutar las pruebas si se dejan ambas hay conflictos entre ellas, para este caso se decidió desinstalar la librería JUnit 4.5.

Para el manejo de pruebas unitarias con JUnit integrado como plug-in en NetBeans ver el apéndice 2.

1.4.4 PRUEBAS PARA APLICACIONES WEB CON SELENIUM IDE

Selenium IDE es una herramienta que permite grabar, depurar y editar las pruebas. Selenium IDE puede ejecutar la prueba en el entorno real que se ejecutará el sistema. Además de que es una herramienta que tiene la facilidad de grabar las acciones que se desean probar y si no se desea optar por esta opción permite la creación manual de las acciones y además genera

el código en diversos lenguajes como C#, PHP, JUnit3, JUnit4, Groovy, Perl, Python y Ruby (30). Las plataformas soportadas por Selenium IDE se ilustran en las siguientes tablas:

Navegadores

Navegador	Selenium IDE	Selenium Remote Control	Selenium Core
Firefox 3	Grabación y reproducción de las pruebas	Inicio de navegador y ejecución de las pruebas	Ejecución de las pruebas
IE 8	No soportado	Inicio de navegador y ejecución de las pruebas	Ejecución de las pruebas
Safari 3	No soportado	Inicio de navegador y ejecución de las pruebas	Ejecución de las pruebas
Opera 9	No soportado	Inicio de navegador y ejecución de las pruebas	Ejecución de las pruebas

Sistemas Operativos

OS	Selenium IDE	Selenium Remote Control	Selenium Core
Windows	Trabaja en Firefox 2+	Inicio de navegador y ejecución de las pruebas	Ejecución de las pruebas
Linux	Trabaja en Firefox 2+	Inicio de navegador y ejecución de las pruebas	Ejecución de las pruebas

Datos tomados de http://seleniumhq.org/about/platforms.html. Para mayor información consultar la página de Selenium IDE (30).

Se utilizó la versión 1.1.0 de Selenium IDE descargado el 30 de abril de 2011 de la dirección web http://seleniumhq.org/download/.

Para ver cómo se crea un caso de prueba en Selenium con Mozilla Firefox como navegador ir al apéndice 3.

CAPÍTULO 2: EMPEZANDO A AUTOMATIZAR

RESUMEN

Se da una explicación de algunos elementos que podrían o son candidatos a ser automatizables. Se divide la automatización en cinco pasos, los cuales abarcan la captura de requisitos y como llevar su registro, la realización de la estrategia del plan de pruebas, mientras que a la vez se dividen las pruebas en pruebas automatizables y no automatizables, se plantean los requerimientos que debe cumplir la plataforma de pruebas, se establece el control de las pruebas desarrolladas, ejecutadas y el avance de su elaboración, también se lleva el registro de los casos prueba y de los defectos y su estado, finalmente se describen las responsabilidades que adquieren los roles mencionados en el capítulo 2 y también se muestran los nuevos artefactos generados.

INTRODUCCIÓN

Para poder llevar a cabo la automatización de las pruebas es necesario fijar que es lo que sí se puede automatizar y qué no. Los mejores candidatos son las acciones repetitivas.

Las actividades que se pueden automatizar son:

- Plan de pruebas
- Generación de datos para la prueba
- Ejecución y análisis de los resultados
- Estatus del defecto y monitoreo
- Creación del reporte

Automatización del Plan de pruebas

Lo que se debe fijar en la automatización del plan de pruebas es la compatibilidad con la herramienta, es decir, sí lo que se desea probar es posible probarlo con una herramienta existente y el costo de ésta. Sin embargo si no existe una herramienta habrá que construirla y entonces habrá que valorar costos tanto monetarios como de tiempo y personal.

Automatización de la generación de datos

La automatización de la generación de datos vale la pena porque permite explorar varios escenarios por lo que se tiene una mayor cobertura en amplitud (categorías) y profundidad (cantidad de datos).

Automatización de la ejecución de la prueba y análisis de los resultados

Para la automatización de la ejecución de la prueba debe tenerse en cuenta que la construcción debe estar documentada e indicar qué es lo nuevo y qué ha cambiado. Los defectos deben ser actualizados la siguiente ocasión que se realice la prueba y se debe tener un control de versiones. En la parte del análisis se deben de poder identificar los defectos que surjan, estas acciones se conocen como pruebas de regresión.

Estatus de defecto y monitoreo

El volver a ejecutar una prueba es un claro ejemplo del apoyo de la automatización y para la ingeniería de la prueba se necesita documentar los detalles de un defecto para poder recrearlo en la siguiente ejecución de la prueba y así comprobar que el defecto fue corregido o ha sufrido algún cambio.

Creación del reporte

- Análisis
  La prueba puede dar falsos positivos y falsos negativos. Donde el falso positivo es aquel que la prueba marca que no hay ningún defecto con el sistema pero el defecto si existe, solo que la prueba no es capaz de detectarlo. Y un falso negativo es cuando la prueba ha marcado que existe un defecto cuando en realidad no lo hay sino que es error de la prueba. Para poder hacer la distinción el ingeniero de pruebas deberá determinar qué tan correcta es la salida.
- Defecto de la entrada
  Una vez detectado un defecto (no falsos positivos ni negativos) se debe documentar y llevar un seguimiento si es posible con una herramienta que sea capaz de hacerlo.
- Recurrencia
  La recurrencia es cuando un defecto que haya sido identificado antes, aparentemente fue arreglado pero reaparece. Debe identificarse correctamente si es un defecto nuevo o uno que ya había sido detectado antes; es muy importante hacer la distinción y llevar un seguimiento de cada defecto por separado.
- Cierre
  Una vez que el defecto haya sido arreglado se cierra, es decir, en la documentación se coloca que el defecto ha sido arreglado. Solo el ingeniero de prueba puede cerrar los defectos.

Para la realización de pruebas se debe definir métricas para cuantificar la calidad del software. Las métricas algunas son (2):

- **Métricas estáticas**
  - *Fan-in/Fan-out*. Fan-in es el número de funciones que llaman a otra función y el Fan/out es el número de funciones que son llamadas. Tener un Fan-in alto

implica que la función (o funciones) que son altamente solicitadas, al ser modificadas causaran un gran impacto en el proyecto porque sus cambios afectaran a todas aquellas funciones que la llaman. Y tener un Fan-out alto, quiere decir que el sistema es altamente complejo por el control que se necesita.

- *Longitud de código*. Es una medida del tamaño del programa, donde mientras más grande sea más susceptible a errores es.
- *Longitud de los indicadores*. Es una medida promedio de la longitud de los identificadores, en donde más grande sea, supondrá que significa algo y por lo tanto el programa será más comprensible.
- *Profundidad del anidamiento*. Esta medida es para las instrucciones "if" anidadas, por su profundidad son difíciles de entender y por lo tanto tienden al error.

- **Métricas dinámicas**
  - *Profundidad del árbol de herencia*. Es el número de niveles de herencia entre clases. Cuanto más profundo sea más difícil de entender las clases ubicadas en las hojas.
  - *Métodos pesados por clase*. Es el número de métodos en cada clase. A cada método se le asigna un peso dependiendo su complejidad, si es un método simple se la da una un valor de peso bajo, si la complejidad es alta se le da un número mayor.
  - *Número de operaciones sobrescritas*. Son el número de métodos reescritos por una subclase. Esto quiere decir que la clase de la cuál heredan no es una buena clase padre.

2.1 PASOS PARA LA AUTOMATIZACIÓN

Para la automatización se recomienda seguir los siguientes seis pasos:

1. Conocer los requerimientos.
2. Desarrollo de la estrategia de pruebas.
3. Definir la plataforma para las pruebas automatizadas de software.
4. Seguimiento constante de los avances y ajuste.
5. Elección del personal adecuado para el proyecto.

2.2 PASO 1: CONOCER LOS REQUERIMIENTOS

Se realiza una recolección de los requerimientos del cliente los cuales deben separarse en requerimientos que se pueden probar y los que no se pueden probar.

La prueba se realiza sobre el hardware y software que va a necesitar el sistema al final del desarrollo por lo que hay que considerar los requerimientos de entorno.

Para comenzar la implementación se plantea llevar el registro de los requisitos como se muestra en la Tabla 2.1.

Requisito	Prioridad	¿Se puede probar? S/N

TABLA 2. 1 CAPTURA DE REQUISITOS

- **Requisito**. Se coloca una descripción en lenguaje natural de la interpretación del requisito, acordada por el equipo de desarrollo junto con el cliente.
- **Prioridad**. Nivel de importancia determinado por el cliente.
- **¿Se puede probar? S/N**. Determinado por el equipo de pruebas se dividen todos los requisitos en probables o no probables, para con los probables realizar el plan de pruebas.

El *requisito* se describe en lenguaje natural para que el cliente pueda entender y dar su aprobación a lo que el equipo de desarrollo entendió de la solicitud del cliente. Se maneja la *prioridad* del cliente para que el equipo de desarrollo y el equipo de pruebas fijen el objetivo del sistema y puedan planear cuáles son las actividades previas a realizar para alcanzar el objetivo del sistema. Finalmente se clasifican los requisitos en probables y no probables para delimitar el plan de las pruebas.

El formato completo para el registro de las captura de requisitos se encuentra en el apéndice A.

2.3 PASO 2: DESARROLLO DE LA ESTRATEGIA DE PRUEBAS

Para el desarrollo correcto de la automatización de pruebas habrá que preguntarse qué pruebas son posibles automatizar o realmente es conveniente hacerlo. Para esto se sugiere realizarse las siguientes preguntas, si se contesta más de una "SÍ" es recomendable automatizar, en caso contrario es más recomendado hacer la prueba de forma manual.

Criterios para la automatización	SÍ	NO
¿La prueba es ejecutada más de una vez?		
¿La prueba cubre la mayoría de las rutas críticas?		
¿Es la prueba eminentemente costosa si se lleva a cabo de forma manual?		
¿Hay situaciones críticas en la que los componentes deben ser automatizados?		
¿La prueba cubre el área más compleja del sistema?		
¿La prueba requiere de muchas combinaciones de datos usando los mismos pasos de la prueba?		

Los resultado esperados son:		
Constantes		
Variables (dentro de un porcentaje de tolerancia)		
¿La prueba consume mucho tiempo para dar un análisis de cientos de salidas?		

Nota: Tabla tomada de (11) página 133, Figure 6-2.

Estos criterios no son absolutos, por lo que quizás se cumplan varios que sean indicativos para la automatización pero no obligan a que realmente se automatice ya que hay muchos factores a considerar. Los criterios se enfocan únicamente al estado del sistema en desarrollo pero no menciona nada sobre el equipo de desarrollo (quizás no cuentan con las habilidades necesarias para automatizar, o tal vez sea más fácil para el equipo desarrollar las pruebas de forma manual por mera experiencia propia) ni el estado general del sistema (tiempo, aumento de costo por usar herramientas de automatización).

Es necesario estar conscientes que no se debe tratar de automatizar todo, debido a factores como el tiempo, costo o habilidad del equipo de desarrollo. Al final las pruebas automatizadas planeadas podrían no se llevarse a cabo en su totalidad.

Se debe realizar un análisis de los casos de uso y establecer de cuáles se pueden automatizar sus pruebas.

Se debe analizar si lo que se va a automatizar es reutilizable, en caso contrario habrá que considerar si es realmente necesario hacerlo porque representa un costo y si este costo está dentro de lo aceptable en términos de tiempo, personal o monetario.

En el momento de la automatización hay que considerar que es necesario priorizar el orden en cómo se van hacer las pruebas. Los criterios son varios, incluyendo (11):

- De mayor al menor riesgo.
- De mayor a menor complejidad.
- Necesidades del cliente.
- Restricciones de presupuesto.
- Restricciones de tiempo.
- Restricciones de personal.

En mi criterio, dar prioridad a las pruebas con mayor riesgo es para un plan que está enfocado en amortiguar los riesgos (por ejemplo el modelo de proceso en espiral, explicado en el capítulo 1), sin embargo esto no es del todo bueno ya que podría estarse dejando de lado aspectos importantes como la complejidad de los módulos. Por otro lado, comenzar por los módulos con mayor complejidad en un principio puede ser laborioso y difícil para el equipo de pruebas pero una vez pasado esta parte los demás módulos serán más sencillos, sin embargo esto también podría retrasar al equipo de pruebas ya que le dedicarían mucho

tiempo al módulo complejo y quizás les quite tiempo para las demás pruebas del resto de los módulos. Ahora bien, cuando se presentan versiones beta del sistema al cliente, mostrando lo que más le interesa al cliente, es necesario hacer las pruebas de esas partes justamente pero esto no es muy bueno porque el cliente generalmente le interesa ver una versión casi final del sistema y esto lleva un trasfondo más grande para lograr el funcionamiento solicitado. En ocasiones hay que mediar las pruebas por las restricciones de presupuesto que se tenga, aunque se quiera probar todo si no se tienen los recursos, simplemente no es posible. La restricción de tiempo existe en todos los proyectos, personalmente, utilizaría este método para tiempos muy cortos, para poder organizar y realizar el mayor número de pruebas posibles. Finalmente, tratándose de equipos pequeños de prueba habrá que analizar la cantidad de pruebas que hay que hacer por el número de personas que realizarán las pruebas, porque de contar el personal necesario para hacer las pruebas quizás se necesite capacitar más gente y esto es un costo agregado, o simplemente decidir realizar las más importantes basado en alguno de los otros criterios mencionados anteriormente.

Para elaborar el plan de pruebas es necesario fijar las métricas con las cuales se va a medir la calidad del software. Las métricas son necesarias para poder evaluar la productividad del equipo de desarrollo y para indicar la calidad del sistema.

Algunas categorías de métricas son:

- **Cobertura**. Parámetros para medir el alcance y de la prueba.
- **Progreso**. Parámetros que contraponen el progreso del sistema contra el criterio de éxito.
- **Técnicas.** Se enfocan en las características del sistema, como: complejidad lógica, el grado de modularidad, etc.
- **Calidad.** Es un indicador del cómo se ajusta el sistema a los requisitos implícitos y explícitos del cliente.
- **Tamaño.** Es para determinar el tiempo y personal necesarios para la terminar el sistema.

La estrategia de la prueba se describe en el plan de pruebas (artefacto definido en el capítulo 2) y para armar la estrategia de la prueba se propone llevar un registro para indicar cuáles de las pruebas se van a automatizar y cuáles no.

En la Tabla 2.2 se muestra el formato para el plan de pruebas.

No. Caso de uso	Caso de uso	Objetivo de la prueba	Éxito de la prueba	Implementación de la prueba	Asignación	Prioridad

TABLA 2. 2 PLAN DE PRUEBAS

- **No. Caso de uso**. Se asigna un identificador al requisito el cuál debe ser único.
- **Caso de uso**. Nombre del caso de uso.
- **Objetivo de la prueba**. Se fijan los objetivos que tiene la prueba.
- **Éxito de la prueba**. Se establece una condición para considerar que la prueba fue exitosa.
- **Implementación de la prueba**. Se define la forma en la que se implementará la prueba ya sea de forma manual o con alguna herramienta de automatización. Si se usa una herramienta está deberá ser indicada.
- **Asignación**. Es el nombre del Ingeniero de componentes que estará encargado de elaborar la prueba.
- **Prioridad**. Nivel de importancia, que dependiendo del criterio que se esté manejado.

El *número del caso de uso* y el *nombre del caso de uso* es el mismo que se le asigna en el diagrama de casos de uso archivado en la especificación de requerimientos. Los *objetivos de la prueba* deben ser concisos, claros y precisos, con el fin de que sean entendibles por todos los miembros del equipo de pruebas. Para establecer el *éxito de la prueba* se habrá tenido que definir con anterioridad la métrica o métricas a usar. La *implementación de la prueba* es solo de modo indicativo ya que la elaboración explícita de la prueba va en el diseño de la prueba que es parte de las actividades definidas en el Proceso Unificado (ver capítulo 2). Este formato es complementario al plan de pruebas como artefacto del Proceso Unificado por lo que para fines de archivado se maneja la *asignación* del personal como una futura referencia. La *prioridad* es definida por el equipo de desarrollo, la cual debe estar balanceada con la prioridad del cliente, establecida en la Tabla 2.1.

El formato completo para el plan de pruebas se encuentra en el apéndice B.

2.4 PASO 3: DEFINIR LA PLATAFORMA PARA LAS PRUEBAS AUTOMATIZADAS DE SOFTWARE

La plataforma elegida debe ser iterativa e incremental según las necesidades de la prueba y preferentemente que esté compuesta por diversas herramientas de prueba para los diferentes tipos de pruebas.

Los aspectos a revisar son los casos de prueba, la lógica que éstos llevan, los datos a utilizar y el código de la prueba.

En la revisión de los casos de prueba se debe verificar que los pasos escritos en la prueba sean legibles, legítimos y precisos. La lógica de la prueba es que los pasos lleven una secuencia lógica y además que está sea eficiente. Los datos deben ser escogidos de tal manera que aporten algo a la prueba y deben de cubrir tanto amplitud como profundidad de la prueba para garantizar que se están cubriendo todos los aspectos. El código generado para

la prueba debe implementar los casos prueba y la lógica de estos. Se revisa que la prueba cubra todos los requerimientos y que no haya duplicidad de código.

2.5 PASO 4: SEGUIMIENTO CONSTANTE DE LOS AVANCES Y AJUSTE

Para el seguimiento de las pruebas se calcularan diversos porcentajes que darán el estado del proyecto.

Porcentaje de automatización (PA). Es el porcentaje de los casos prueba que se van a automatizar y se expresa por la ecuación:

$$PA(\%) = \frac{CPA}{CT} = \frac{Número\ de\ casos\ prueba\ automatizables}{Número\ de\ casos\ prueba\ \ totales}$$

Este porcentaje solo nos sirve como referencia para ver que tanto se va a automatizar pero no es estrictamente exacto, ya que aunque la prueba sea automatizable no necesariamente significa que está deba ser automatizada.

Progreso de automatización (AP). Es la medida del avance de implementación sobre las pruebas que ya han sido automatizadas y las que todavía no lo han sido.

$$AP(\%) = \frac{CA}{CPA} = \frac{Número\ de\ casos\ prueba\ automatizados}{Número\ de\ casos\ prueba\ automatizables}$$

Progreso de la prueba (PP). Para el momento en que se ejecutan las pruebas y de llevar un control sobre el avance de la ejecución de las mismas se obtiene un índice sobre los casos prueba ya ejecutados y el número total de casos prueba.

$$PP = \frac{CE}{CT} = \frac{Número\ de\ casos\ prueba\ ejecutados}{Número\ de\ casos\ prueba\ totales}$$

En el diseño de la prueba al definir los casos de prueba se presentaran los casos en donde se aplique una prueba de caja blanca o una de caja negra y seguramente se necesiten de herramientas distintas pero eso no afecta el cómo se documente la prueba. Al momento de ejecutar la prueba se debe registrar cuál fue el valor obtenido, si la prueba ha cumplido con el criterio de éxito y si hay algún comentario, para poder llevar este control tanto a la hora de realizar el diseño como al ejecutar la prueba se sugiere hacer uso de la Tabla 2.3.

Identificador	Caso prueba	Valor esperado	Valor obtenido	Resultado de la prueba	Comentarios

TABLA 2. 3 EJECUCIÓN DE LA PRUEBA

- **Identificador**. Es el identificador que se asignó al caso de uso o bien se asigna un nuevo identificador el cuál debe de seguir siendo único.
- **Caso prueba**. Son los pasos para realizar la prueba.
- **Valor esperado**. Es la respuesta del sistema que se espera obtener.
- **Valor obtenido**. Es la reacción real del sistema.
- **Resultado de la prueba**. Dependiendo del criterio de aceptación de la prueba definido en el plan de pruebas se determina si la prueba fue aprobada o no lo fue.
- **Comentarios**. Son notas que deja el Ingeniero de pruebas de sistema para el Ingeniero de pruebas. Principalmente si el defecto de la prueba fue por causa del componente y no del sistema.

El *identificador* es el mismo asignado en la Tabla 2.2 con el fin de no llevar siempre el nombre del caso de uso pero si referenciarlo para saber de quién se está hablando. El *caso de prueba* es definido por el ingeniero de pruebas en la actividad de planificación de la prueba y es donde se describen las acciones y los datos a introducir en la prueba. El ingeniero de pruebas también es el encargado de establecer cuál es el *valor esperado* de la prueba. El *valor obtenido*, *resultado de la prueba* y los *comentarios* son llenados por el encargado de ejecutar la prueba. El encargado de ejecutar la prueba es capaz de establecer si la prueba fue exitosa o no, por el criterio de éxito de la prueba definido en la Tabla 2.2. Y finalmente en los *comentarios* describe alguna anomalía observada durante la ejecución de la prueba.

El formato completo se encuentra en el Apéndice C.

Durante la ejecución de la prueba es necesario llevar un control sobre los defectos que se presenten durante la ejecución de la prueba. Y para eso hacemos uso de la Tabla 2.4.

Identificador	Descripción del defecto			Prioridad	Estado
	Acciones	Lo que hace actualmente	Lo que debería hacer realmente		

TABLA 2. 4 DETECCIÓN DE DEFECTOS

- **Identificador**. Es el identificador del caso de uso en donde la prueba no fue aceptada debido a un defecto.
- **Descripción del defecto.**
    - **Acciones.** Es la secuencia de pasos que se dieron para que ocurriese el defecto.
    - **Lo que hace actualmente.** Descripción de la respuesta del sistema después de ejecutar las acciones indicadas.
    - **Lo que debería hacer.** La respuesta del sistema que debería ocurrir después de realizar las mismas acciones que generaron el defecto.

- **Prioridad**. Nivel de afectación a la funcionalidad del sistema.
- **Estado**. Referente a si el defecto ha sido corregido, amortiguado o no ha sido resuelto.

Como se ha mencionado anteriormente el *identificador* es la forma para poder relacionar con el nombre del caso de uso, con las tablas propuestas y tener un seguimiento constante del mismo. La *descripción del defecto* es necesario incluso aunque la prueba sea automatizada y para este caso se menciona cual fue la clase o el método en donde ocurrió el defecto. La *prioridad* es para clasificar la urgencia del problema, esto principalmente a que se está trabajando en iteraciones y es necesario saber si es posible pasar a la siguiente iteración o el equipo se debe concentrar en solucionar el problema. El *estado* es para poder visualizar el avance del defecto y poder identificar aquellos que aún no hayan sido resueltos.

Ahora bien, para los defectos que han sido corregidos y en el caso de que se presenten más adelante se lleva un registro de la corrección de defectos mostrado en la Tabla 2.5.

Identificador	Causa	Solución

TABLA 2. 5 CORRECCIÓN DE DEFECTOS

- **Identificador**. Identificador dado al requisito o al caso de uso en el plan de pruebas.
- **Causa**. El motivo por el cual se daba el defecto.
- **Solución**. Acción o decisiones que se tomaron para corregir el defecto o amortiguar su efecto.

Se describe la *causa* del defecto para determinar las condiciones que dieron lugar al mismo y en dado caso de que se repita el mismo defecto (el mismo defecto definido por las mismas condiciones) contar ya con la solución al problema y no tener que volver a "encontrar" la misma solución. Y si en apariencia pareciera que se tratase del mismo defecto pero las causas son distintas es necesario documentarlo porque en realidad se trata de defectos distintos.

Ambos formatos se pueden ver en el Apéndice D.

Para realizar la automatización son necesarios los campos *caso de prueba* y *valor esperado* de la Tabla 2.3, ya que en el primero se describe los pasos que deben ser implementados y en el segundo se establece cual debe ser la respuesta del sistema.

2.6 PASO 5: ELECCIÓN DEL PERSONAL ADECUADO PARA EL PROYECTO

La elección del personal es muy importante debido a que son la razón del éxito de la implementación de los pasos anteriores. En el momento de la elección de personal se deben considerar las siguientes habilidades básicas:

- Adaptabilidad. Esto porque la pruebas automatizadas se desarrollan con distintas técnicas, diferentes ambientes, diferentes tecnologías, etc.
- Aprendizaje rápido. Para no perder mucho tiempo aprendiendo las nuevas herramientas que se vayan a utilizar durante el desarrollo de las pruebas.
- Capacidad para conceptualizar actividades complejas.
- Entender requerimientos complejos y poder diseñar planes de prueba para estos.
- Ser creativo, tener la capacidad de poder visualizar diferentes caminos o circunstancias que se pueden presentar y así identificar cuando el sistema podría fallar.
- Con capacidad verbal y escrita para poder comunicar a las partes interesadas cuestiones e ideas.

Pero el Proceso Unificado ya define los roles que deben de existir en un equipo de pruebas que son:

- Ingeniero de pruebas
- Ingeniero de pruebas de integración
- Ingeniero de pruebas de sistema
- Ingeniero de componentes

Los cuales fueron mencionados en el capítulo 2.

Sin embargo a estos roles se les agregaron algunas responsabilidades, además de las definidas por el Proceso Unificado. La Tabla 2.6 muestra las responsabilidades de los roles mencionados.

Rol	Responsabilidad
Ingeniero de pruebas	• Analiza el diagrama de casos de uso y los requisitos para la automatización para poder desarrollar el plan de pruebas. • Establece los casos prueba. • Supervisa el avance del desarrollo de las pruebas. • Verifica que la prueba automatizada, en términos de código, implemente de forma correcta los casos de uso y la lógica que estos tienen. • Verifica si el defecto de la prueba fue error de la prueba o del sistema.
Ingeniero de pruebas de integración	• Ejecuta la prueba (automatizada o no) y llena los formatos de la ejecución de la prueba y detección de defectos.
Ingeniero de pruebas de	• Ejecuta la prueba (automatizada o no) y llena los formatos de la

sistema	ejecución de la prueba y detección de defectos.
Ingeniero de componentes	• Implementa las pruebas.

TABLA 2. 6 RESPONSABILIDADES DE LOS ROLES DEL PU

A continuación en la figura 2.1 se muestra un diagrama del flujo de trabajo, con los roles involucrados en las pruebas y los respectivos productos que se están proponiendo.

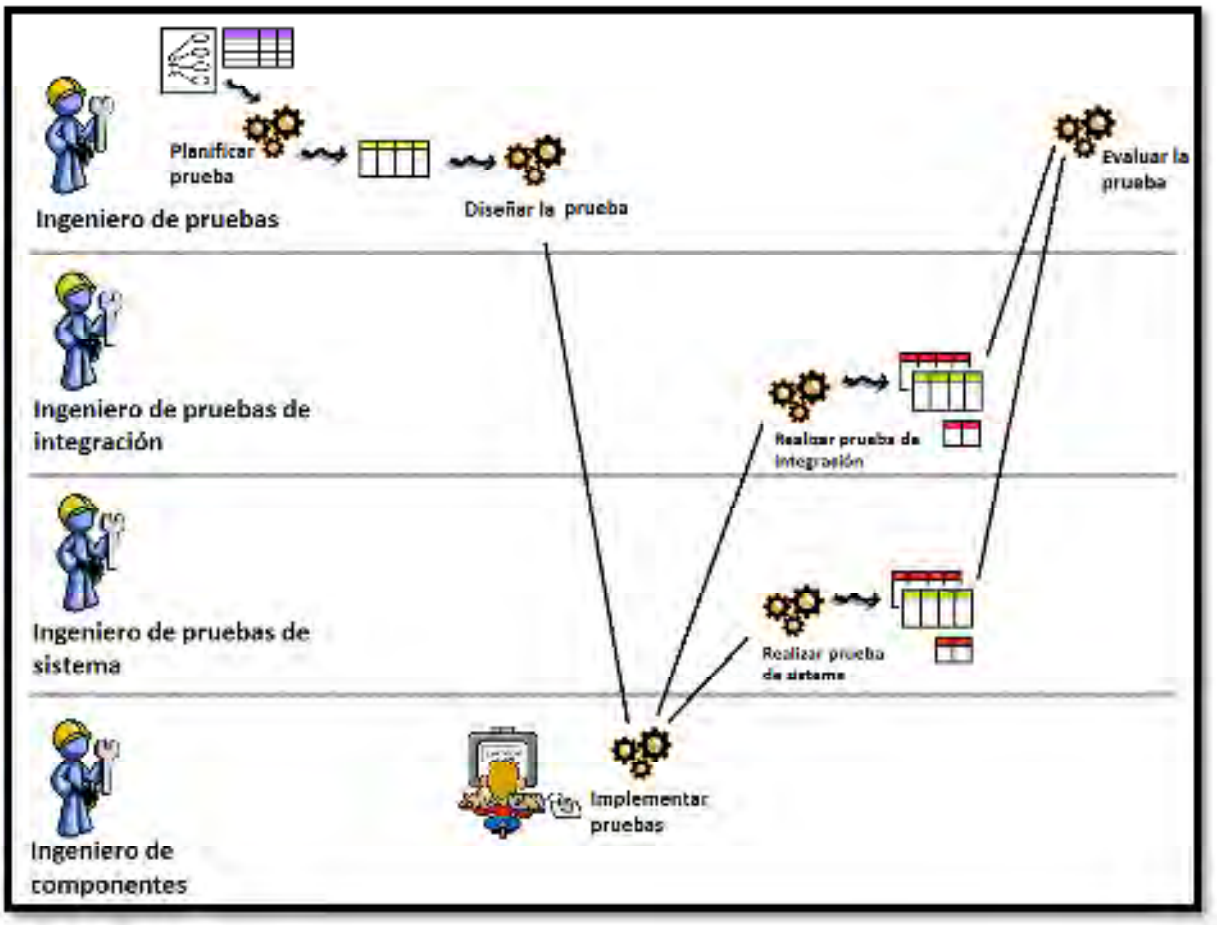


FIGURA 2. 1 DIAGRAMA DEL FLUJO DE TRBAJO

Nomenclatura de la Figura 2.1

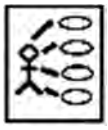
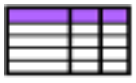


Diagrama de casos de uso



Criterios para la automatización

Tabla 5.2 Plan de prueba

Tabla 5.3 Ejecución de la prueba

Tabla 5.4 Detección de defectos

Tabla 5.5 Corrección de defectos

Para poder iniciar el proceso de las pruebas el ingeniero de pruebas necesita contar con el diagrama de casos de uso, el formato para la captura de requisitos (apéndice A) y los criterios para la automatización de pruebas. Con esto elabora el plan de pruebas que es donde se describen estrategias, recursos y la planificación de las pruebas.

Para la planificación de la prueba se tiene el formato de plan de prueba (apéndice B). Con el plan de pruebas se diseña la prueba donde se describen los pasos de cómo construir la prueba, ese proceso de construcción pasa al ingeniero de componentes que es el encargado de implementar las pruebas.

Las pruebas ya elaboradas pasan al ingeniero de pruebas de integración y al ingeniero de pruebas de sistema que son los encargados de ejecutar la prueba y llenan el formato de la ejecución de la prueba (apéndice C) sólo en los apartados de valor obtenido, resultado de la prueba y si tienen algún comentario u observación durante la ejecución de la prueba (los primeros campos fueron llenados anteriormente por el ingeniero de pruebas). En el caso de haber presentado un defecto en la ejecución de la prueba se llena el formato de detección de defectos (apéndice D) y esos formatos son enviados al ingeniero de pruebas.

Con esos formatos el ingeniero de pruebas evalúa la prueba y determina si el defecto que ocurrió fue un error de la prueba o bien si fue error del sistema. De ser error de la prueba se tiene que corregir, ya sea, en el plan de prueba o en la implementación de la prueba para luego volverla a ejecutar y volver a revisar. En caso de que el error sea del sistema, es un defecto encontrado y se notifica al equipo de desarrollo (11).

CAPÍTULO 3: EJEMPLO DE LA IMPLEMENTACIÓN PARA LAS PRUEBAS AUTOMATIZADAS

RESUMEN

A un proyecto realizado por el equipo Adhoc Systems en un periodo aproximado de cinco meses, con el nombre de B@UNAM se la aplicó el método propuesto detallando cómo queda el llenado de las tablas y cómo pasa la información documentada a código.

INTRODUCCIÓN

Para la ejemplificación de las pruebas automatizadas se tomó el sistema B@UNAM desarrollado por el equipo Adhoc Systems en el periodo agosto – diciembre del año 2010. El sistema consistió en un control escolar, encargado del registro y modificación de los datos de alumnos, profesores y tutores, y la creación de grupos.

El criterio para la selección de los casos para ejemplificar fue la de tratar de dar un panorama general del uso para la propuesta planteada en los capítulos anteriores.

3.1 PASO 1: CONOCER LOS REQUERIMIENTOS

Como se mencionó en el Capítulo 2, es necesario tener un registro de los requerimientos que el cliente solicité y en seguida se muestra un fragmento del archivo de la captura de requerimientos generales del sistema B@UNAM correspondiente a la Tabla 2.1.

CAPTURA DE REQUISITOS

Analista: Javier, Arlen, Hector, Dario, Daniel y Carolina **Iteración:** 1

Sistema: B@UNAM

Requisito	Prioridad	¿se puede probar? S/N
El usuario podrá registrar un alumno en el sistema		
Contiene información de registro correspondiente a los datos generales del alumno (datos de identificación y lugar de nacimiento)	Media	SI
Contiene información de registro correspondiente al domicilio del alumno	Media	SI
Contiene información de registro correspondiente a los estudios anteriores del alumno (secundaria, bachillerato y licenciatura)	Media	SI

FIGURA 3. 1 FRAGMENTO DEL DOCUMENTO CAPTURA DE REQUISITOS PARA EL SISTEMA B@UNAM

Es posible realizar tantos archivos como sean necesarios, en este caso se tiene un fragmento para los requisitos del módulo de grupos del sistema B@UNAM. Esto se hace con el fin de poder detallar cada módulo.

CAPTURA DE REQUISITOS

Analista:	Darío y Carolina	**Iteración:**	1
Sistema:	B@UNAM	**Módulo:**	Grupos

Requisito	Prioridad	¿se puede probar? S/N
GENERALES		
Se pide que exista la generación de uno o más grupos de forma simultánea.	Media	SI
Para que un objeto sea considerado como grupo debe tener profesor, clave del grupo y asignatura.	Media	NO
En la vista se desea poder ver la clave y el nombre del plan, plantel e ISI.	Media	NO
El sistema debe permitir que se ingrese el nombre o la clave y en automático mostrar su complemento correspondiente.	Media	SI
El ciclo escolar debe estar arriba del plan para poder ver que ciclos tiene el plan seleccionado.	Media	NO

FIGURA 3. 2 CAPTURA DE REQUISITOS DEL MÓDULO GRUPOS DEL SISTEMA B@UNAM

En este caso los requisitos que no se pudieron se debió a que el caso del primero es característica propia del objeto llamado grupo y los otros dos son referentes a la visualización de la interfaz; por lo que la única forma de probarlos es mostrando la vista al cliente y que dé su visto bueno,, no requiere más prueba.

3.2 PASO 2: DESARROLLO DE LA ESTRATEGIA DE PRUEBAS

Después de haber realizado el análisis a los requerimientos y tener los diagramas de casos de uso necesarios para el desarrollo del sistema se elabora el plan de pruebas y se determina cuáles de todas las pruebas serán automatizadas y cuáles no. Cada prueba se analiza con los criterios para la automatización.

Criterios para la automatización	SÍ	NO
¿La prueba es ejecutada más de una vez?	X	
¿La prueba cubre la mayoría de las rutas críticas?	X	
¿Es la prueba eminentemente costosa si se lleva a cabo de forma manual?	X	

¿Hay situaciones críticas en la que los componentes deben ser automatizados?		X
¿La prueba cubre el área más compleja del sistema?	X	
¿La prueba requiere de muchas combinaciones de datos usando los mismos pasos de la prueba?	X	
¿Son los resultados esperados constantes, no cambian o varían con cada prueba? Si los resultados varían, ¿están dentro del porcentaje de tolerancia como valor esperado?	X	
¿La prueba consume mucho tiempo para dar un análisis de cientos de salidas?		X

Ahora una breve explicación del porqué de las respuestas dadas anteriormente.

- Los módulos del sistema tienen las mismas acciones: insertar, modificar, dar de alta o baja; por lo que las pruebas serán semejantes en los tres módulos.
- Dado que el objetivo del equipo de desarrollo es entregar el sistema con las funciones antes mencionadas se estaría cubriendo sus funciones principales.
- La prueba es muy costosa en tiempo porque los formularios son muy largos y hay muchas restricciones.
- No se presentan situaciones en extremo críticas para la automatización.
- Se están cubriendo las tres funciones básicas del sistema y por ende las más complejas.
- Sí, para verificar que los campos sean llenados correctamente y no con basura.
- Son constantes.
- No está dentro del alcance de sistema interactuar con muchos usuarios.

Finalmente el plan de pruebas general, correspondiente a la Tabla 5.2, queda de la siguiente manera:

PLAN DE PRUEBAS

Ingeniero de prueba:	Carolina		
Sistema:	B@UNAM	**Módulo:**	General

No. Caso de uso	Caso de uso	Objetivo de la prueba	Éxito de la prueba	Implementación de la prueba	Asignación	Prioridad
3	Administrar Asesor					
3.1	Registrar asesor	Verificar que los datos del asesor queden	El registro quedó correctamente insertado	Se usará JUnit para realizar la prueba	Carolina	Alta
3.2	Modificar asesor	Verificar que los datos modificados queden almacenados en la	Los cambios se efectuaron con éxito	Se usará JUnit para realizar la prueba	Carolina	Alta
3.3	Cambiar estado asesor	Verificar que el estado del asesor haya sido cambiado y se mantenga así	El estado del asesor se cambio con éxito	Se usará JUnit para realizar la prueba	Carolina	Alta

FIGURA 3. 3 PLAN DE PRUEBAS GENERAL DEL SISTEMA B@UNAM

Después hay que realizar el diseño de la prueba correspondiente a la Tabla 2.3, donde se plantean los casos prueba, los datos a utilizar y cuando se programe hay que verificar que el programa lleve la secuencia y lógica de los casos prueba.

3.3 PASO 3: DEFINIR LA PLATAFORMA PARA LAS PRUEBAS AUTOMATIZADAS DE SOFTWARE

El proyecto fue desarrollado en NetBeans IDE y se decidió que el desarrollo de la prueba se hiciera en la misma plataforma por la facilidad que presta NetBeans IDE con su pantalla de resultados y se evita mudar el proyecto a otra plataforma. Las pruebas de caja blanca se realizaron con JUnit y las pruebas de caja negra se realizaron con Selenium. JUnit al estar desarrollado en Java se adapta a nuestro código que también está en el mismo lenguaje y NetBeans cuenta con un plug-in del mismo por lo que no es necesario mudar el proyecto a una nueva plataforma solo para probar el sistema. Y Selenium se ejecuta directamente sobre el navegador (solo funciona en Mozilla Firefox) por lo que no representa conflicto para el mismo proyecto.

3.4 PASO 4: SEGUIMIENTO CONSTANTE DE LOS AVANCES Y AJUSTE

En este caso no se realizó el calculó de los porcentajes ya que solo sirven para cuando se está desarrollando las pruebas del sistema por completo pero este no es el caso, solo se han tomaron algunos casos representativos.

Para iniciar el diseño de las pruebas se tiene que el sistema se encuentra en un proyecto de NetBeans IDE 6.9.1, el cual usa JUnit para realizar las pruebas de caja blanca y que ya fue detallado en el capítulo 1. Después de haber creado la clase a probar se deben modificar los métodos que se crean de manera automática, hay que estructurar el caso prueba con los datos prueba y establecer cuál es el valor esperado y cuál es el que obtiene la prueba para que está los comparé.

Para ejemplificar lo que se acaba de mencionar en la Figura 3.4 se muestra el caso prueba para el método ValidarVacios:

Identificador	Caso prueba	Valor esperado
control.AdministracionAlumnos/CtrlAdministrarAlumnoAOTest/TestValidarVacios	cad= "Ab cd cd vf vf"	false
	cad= ""	true

FIGURA 3. 4 CASO PRUEBA DE TESTVALIDARVACIOS

Hay que notar que el identificador utilizado está compuesto por *nombre del paquete/nombre de la clase/nombre del método prueba* recomendado para pruebas de caja blanca, ya que nos da la ruta del elemento que está probando.

El siguiente fragmento de código muestra el contenido del método que se desea probar:

```
public boolean validarVacios(String cad) {
    return cad.trim().isEmpty();
  }
```

El método *validarVacios* recibe una cadena y regresa un valor booleano, *true* si la cadena está vacía y *false* en caso contrario.

El código que prueba este método se muestra en seguida:

```
public void testValidarVacios() {
    System.out.println("validarVacios");
    CtrlAdministrarAlumnoAO instance = new CtrlAdministrarAlumnoAO();
    //Dato prueba
    String cad1 = "Ab cd cd vf vf";
    String cad2 = "";
    //Valor esperado
    boolean expResult1 = false;
    boolean expResult2 = true;
    //Se ejecuta el método y se comparan
    boolean result1 = instance.validarVacios(cad1);
    boolean result2 = instance.validarVacios(cad2);
    assertEquals(expResult1, result1);
    assertEquals(expResult2, result2);
  }
```

Se imprime el nombre de la prueba solo por motivos de visualización, se crea una instancia de la clase a la que pertenece el método que se está probando. Se establece el dato prueba y dado que no es una cadena vacía se espera que el método regrese como valor “false”. Con la instancia creada se ejecuta el método introduciendo el valor prueba y finalmente el resultado se compara con el valor esperado. Si no hay problema el test será exitoso mostrándose en una ventana similar a la de la Figura 3.5.

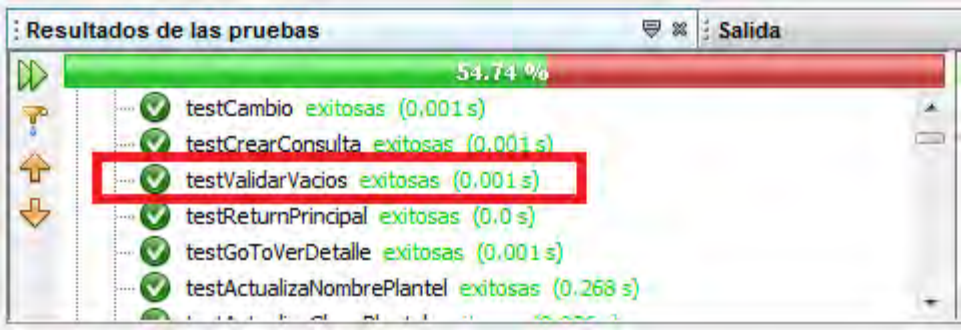


FIGURA 3. 5 SALIDA DE NETBEANS DESPUÉS DE EJECUTAR LA PRUEBA

Para las pruebas de caja negra se usó Selenium. Como ejemplo se muestra el caso prueba del caso de uso 5.4 Consultar grupos.

Identificador	Caso prueba	Valor esperado
5.4	Plantel=1006	Mostrar 3 registros
	Plantel=1006 Plan=1 Asignatura=Álgebra y principios de física	Mostrar 2 registros

FIGURA 3. 6 CASO PRUEBA CU 5.4 CONSULTAR GRUPOS

La prueba implementada luce de esta forma:

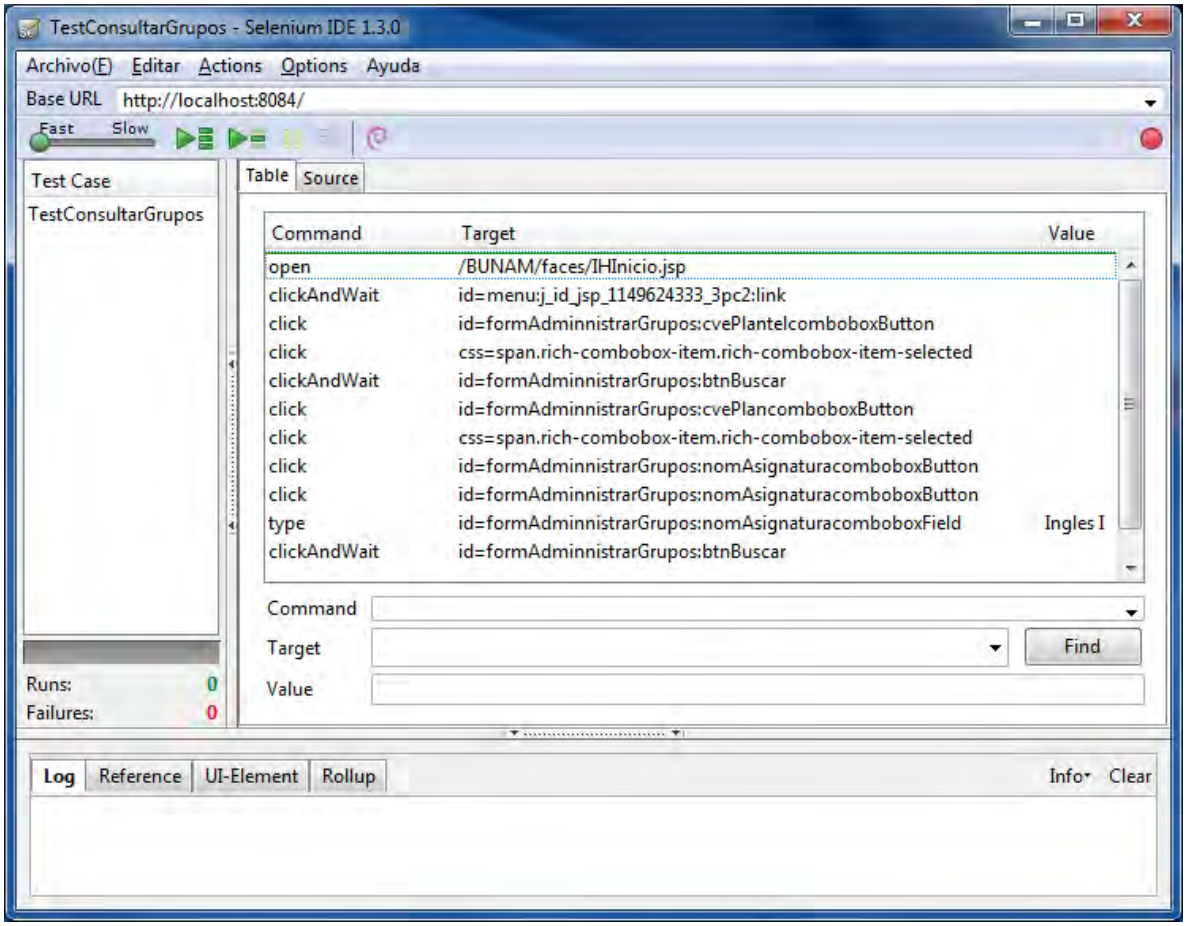


FIGURA 3. 7 IMPLEMENTACIÓN DEL CASO PRUEBA 5.4 CONSULTAR GRUPOS

Y después de ejecutar la prueba la salida se visualiza como se muestra en la Figura 3.8.

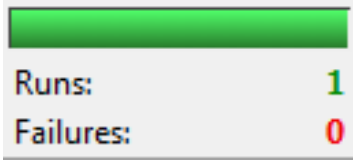


FIGURA 6. 8 SALIDA DEL RESULTADO DE LA PRUEBA

En las figuras 3.9 A y B se observa la ejecución de la pruebas para el método *TestCrearConsulta* donde todos los campos han sido ya llenados (el de comentarios es opcional y sólo para observaciones)

Identificador	Caso prueba	Valor esperado
control.AdministracionAlumnos /CtrlAdministrarAlumnoAOTest /TestCrearconsulta	idPalntel="1006" idISI="1006" idAlumno=""	"AND plantel_alumnos.idplantel=1006 AND plantel_alumnos.idisi=1006"
	idPalntel="" idISI="" idAlumno=""	""

FIGURA 3. 9 A. DOCUMENTO DE LA EJECUCIÓN DE LA PRUEBA

Valor obtenido	Resultado de la prueba	Comentarios
"AND plantel_alumnos.idplantel=1006 AND plantel_alumnos.idisi=1006"	Exitosa	
""		

FIGURA 3. 9 B. DOCUMENTO DE LA EJECUCIÓN DE LA PRUEBA

En la figura 3.10, observa el resultado de haber ejecutado la prueba y en la figura 3.11 se ve la salida del resultado para visualizar que realmente está es la que se esperaba.

FIGURA 3. 10 RESULTADO DE LA PRUEBA PARA TESTCREARCONSULTA

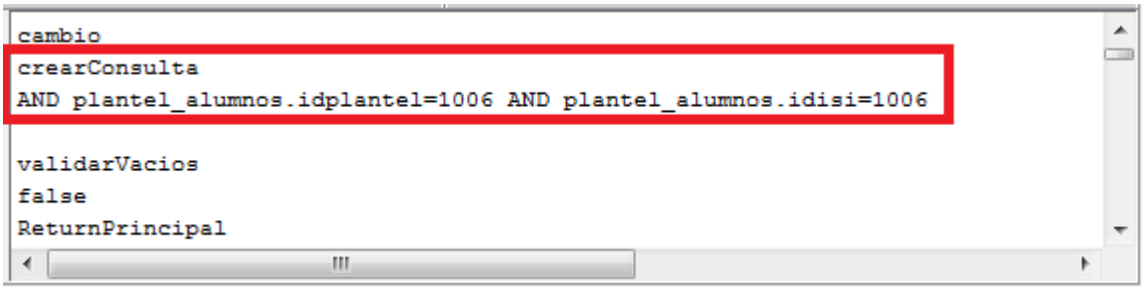


FIGURA 3. 11 IMPRESIÓN DEL RESULTADO

Ahora que se han empezado a ejecutar las pruebas también empiezan a aparecer los errores y defectos por lo que es necesario identificar cuáles son del sistema y cuáles son de las pruebas.

Ahora bien, en el siguiente código

```
public void testValidarVacios() {
    System.out.println("validarVacios");
    CtrlAdministrarAlumnoAO instance = new CtrlAdministrarAlumnoAO();
    //Dato prueba
    String cad = null;
    //Valor esperado
    boolean expResult = false;
    //Se ejecuta el método y se comparan
    boolean result = instance.validarVacios(cad);
    System.out.println(result);
    assertEquals(expResult, result);
}
```

Se introdujo valor nulo a la cadena lo que resulto en un error de excepción mostrado en la figura 3.12

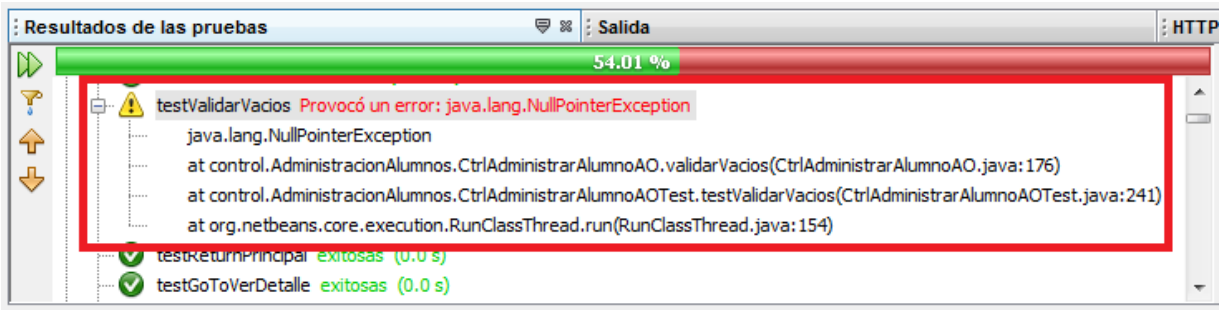


FIGURA 3. 12 ERROR DE EXCEPCIÓN

Para este caso se tiene que registrar el defecto en el documento para la detección de defectos como se muestra en las figuras 3.13 A y B.

Identificador	Descripción del defecto		
	Acciones	Lo que hace actualmente	Lo que debería hacer
control.AdministracionAlumnos/CtrlAdministrarAlumnoAOTest/TestValidarVacios	Se introdujo null al valor de cad	Manda una excepción: NullPointerException	No permitir que el campo quede con valor nulo y suplirlo por defecto con cadena vacía

FIGURA 3. 13 A. REGISTRO DEL DEFECTO

Prioridad	Estado
Media	Pendiente

FIGURA 3. 14 B. REGISTRO DEL DEFECTO

3.5 PASO 5: ELECCIÓN DEL PERSONAL ADECUADO PARA EL PROYECTO

De manera involuntaria en la realización de las pruebas de este sistema todos los miembros del equipo tomaron en algún momento todos y cada uno de los roles para la elaboración de pruebas, cada grupo trabajo sus propias pruebas correspondientes a su módulo asignado. Por lo que solamente es necesario mencionar que el hecho de la existencia de los roles no implica que únicamente una persona es la encargada de esas actividades sino que muchas pueden tomar ese mismo rol para un mismo módulo o distintos.

RESUMEN DE LOS APORTES DEL TRABAJO

En todas partes se habla de automatizar tareas sin embargo siempre se expresan pasos generales de cómo llevar a cabo esta tarea. La finalidad de este trabajo es presentar la forma de automatizar pruebas de software, es decir apoyarse en herramientas automáticas de prueba para que lleven a cabo tareas repetitivas en la prueba de software.

En este trabajo se enfocó en el Proceso Unificado por ser un proceso estándar y muy común entre los desarrolladores de software, que ejemplifica las actividades comunes de los procesos de software en general.

Como en todo proceso la documentación definida es muy importante por eso las tablas propuestas para documentar lo necesario para las pruebas son aportes importantes de este trabajo. Al realizar las tablas siempre se consideró que estas fueran "ligeras" en representación pero que fueran capaces de albergar la información más importante para la elaboración de pruebas del software.

Otra aportación de este trabajo fue la integración de manuales para el manejo de las herramientas automáticas para prueba de software. A pesar de que en estos tiempos existe una gran variedad de manuales para la gran mayoría de las herramientas, puedo decir que por experiencia propia, se encuentran incompletos. Por esta razón se decidió realizar los manuales tratando de que estos fueran lo más detallados posible y explicando aspectos de la herramienta que no son tan intuitivos.

El llevar está propuesta a otros procesos o metodologías no implica complicaciones porque los pasos para realizar la automatización son genéricos y la forma de condensar la información para automatizar se plateó que fuera lo más práctica posible. Por ejemplo, si se necesitará aplicar en una metodología ágil, se podrían tomar las tablas presentadas sin necesidad de anexar más información porque se está representando lo indispensable para las mismas; en el caso de una metodología que siga el modelo en espiral que se centra en riesgos, las pruebas quedarían igualmente expresadas en las tablas descritas bajo el criterio de priorizar los riesgos. Por lo que la propuesta no está atada a una forma en específico de realizar las pruebas.

CONCLUSIONES GENERALES

El objetivo de este trabajo era establecer los pasos para la automatización siguiendo el marco del Proceso Unificado utilizando herramientas de código abierto y/o software libre. Además de proveer de un manual de las herramientas utilizadas. Los objetivos se cumplieron por que se presenta una propuesta para realizar pruebas automatizadas dentro de los pasos definidos por el Proceso Unificado además de agregar la información a los artefactos que se generan en el flujo de trabajo de pruebas para el registro de las pruebas automatizadas. Por otro lado en los apéndices 1, 2 y 3 se encuentra el manual de cómo crear y ejecutar pruebas unitarias.

Además, durante la elaboración de la tesis comprendí mucho de la etapa de pruebas que todo sistema de software en desarrollo debe tener y aparte que este proceso de pruebas puede realizarse de manera más fácil con ayuda de herramientas que nos permitan hacer las pruebas. Me di cuenta de la gran cantidad de herramientas que existen por lo que es probable encontrar una que se adecué, tal vez no al 100% pero sí en una gran medida, a nuestras necesidades. Aprendí a usar JUnit y Selenium como herramientas de pruebas, ver cómo funcionaban y ver que al tener un plan bien estructurado de las pruebas es posible tener resultados rápidos y efectivos sobre el estado del sistema.

Los pasos para la automatización no alteran el flujo de trabajo de las pruebas definido por el Proceso Unificado, que era una de las finalidades. Se añadieron algunas responsabilidades a los roles ya existentes por el Proceso Unificado y no se alteraron los artefactos solo se les añadió una forma de cómo sintetizar la información más importante para el programador y para el equipo de pruebas.

La aportación principal son los pasos para la automatización y aunque no son absolutos dan un panorama general de los puntos que se deben tocar para automatizar pruebas. Además de que también se está proveyendo de una forma de llevar un registro y control del análisis, elaboración y ejecución de las pruebas, lo cual siempre es importante para futuras referencias, en especial si hay que dar mantenimiento al sistema. Finalmente, también se realizó un pequeño manual para el manejo de las herramientas.

El manual de las herramientas para la automatización de pruebas a pesar de ser pequeño presenta funcionalidad general para llevar a cabo una prueba de principio a fin. Con el ejemplo de la implementación se logra apreciar la estructura de un programa que prueba otro programa y lo sencillo que puede llegar a ser utilizar herramientas debido a la reutilización de código.

TRABAJO A FUTURO

Implementar la metodología planteada en diversos sistemas para lograr enfatizar la funcionalidad que tiene y poder realizar mejoras a la metodología, además de aumentar el plan propuesto para pruebas de sistema y de integración.

Seguir investigando las herramientas que se encuentran disponibles y aquellas de las cuales se tenga acceso poder realizar un manual ya que eso ayudaría a futuras generaciones a usarlas y no descartarlas por el tiempo de aprendizaje que estás puedan llegar a tener.

REFERENCIAS BIBLIOGRÁFICAS

1. **LICHTER, Horst**. *Software Processes in an Agile Worl***.** Malaysia : Workshop Proceedings, 2011. 978-967-0194-14-1.

2. **SOMMERVILLE, Ian.** *Ingeniería del software.* Madrid : Addison-Weslwy, 2005. 84-7829-074-5.

3. **cflores334.** Introducción a la Ingeniería de Software. *Modelo de Prototipos.* [En línea] 20 de Octubre de 2007. [Citado el: 28 de Enero de 2012.] http://cflores334.blogspot.es/1192848180/.

4. **Ramanathan, Murugappan.** *A new software process model designed from the basics of evolutionary biology and software evolution.* Oklahoma : s.n., 2007. 0549362339, 9780549362333.

5. **Chamberlain Noboa , Ricardo.** Ricardo Chamberlain Noboa Blog de Administración de Proyectos. *Modelo de espiral.* [En línea] Abril de 2011. [Citado el: 2012 de Enero de 28.] http://www.ricardochamberlain.com/2011/04/el-modelo-de-espiral/.

6. **Letelier, Patricio y Penadés, Carmen .** Ciencia y Técnica Administrativa. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* [En línea] 15 de Enero de 2006. [Citado el: 28 de Enero de 2011.] http://www.cyta.com.ar/ta0502/v5n2a1.htm. 1666-1680.

7. **JACOBSON, Ivar, BOOCH, Grady y RUMBAUGH, James.** *El proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educación, 2000.

8. **Répási, Tibor.** *Software Testing – State of the Art and Current Research Challenges***.** Timisoara : s.n., 2009, SACIIEEE, págs. 47-50. 978-1-4244-4477-9.

9. **Guardati, Silvia y Ponce, Alain.** MoProSoft Modelo de Procesos para la Industria de Software. *GUÍA DE PRUEBAS DE SOFTWARE (GPS) PARA MOPROSOFT.* [En línea] 2010. [Citado el: 29 de Enero de 2012.] http://comunidadmoprosoft.org.mx/Aportaciones/GPS_para_MoProSoftv1.0.pdf.

10. **Cristiá, Maximiliano.** Facultad de Ciencias Exactas, Ingeniería y Agrimensura. *Introducción al Testing de Software.* [En línea] Noviembre de 2009. [Citado el: 29 de Enero de 2012.] http://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf.

11. **DUSTIN, Elfried, GARRETT, Thom y GAUF, Bernie.** *Implementing Automated Software Testing: introduction, management, and performance.* Estados unidos : Addison-Wesley, 1999. 0-201-43287-0.

12. **Volokh, Eugene y VESOFT.** Adager. The Adapter/ Manager of IMAGE/ SQL Databases. *AUTOMATED TESTING -- WHY AND HOW.* [En línea] Diciembre de 1990. http://www.adager.com/vesoft/automatedtesting.html.

13. **BINDER, Robert.** *Testing Object-Oriented Systems: Models, Patterns, and Tools.* s.l. : Addison-Wesley Professional, 2000. 0-201-8093-9.

14. **HP.** HP Unified Functional Testing software. *HP Functional Testing software.* [En línea] Agosto de 2010. http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA2-3779ENW.pdf.

15. **Borland Software Corporation.** Borlan A Micro focus Company. *SilkTest.* [En línea] 1994. http://www.borland.com/us/products/silk/silktest/index.aspx.

16. **Zeta Software GmbH.** Zeta Test. *What is Zeta Test?* [En línea] 2012. [Citado el: 29 de Enero de 2012.] http://www.zeta-test.com/what-is-zeta-test.html.

17. **Verisium.** VERSIUM. *vTest Functional and Regression Test Automation.* [En línea] 2011. [Citado el: 29 de Enero de 2012.] http://www.verisium.com/products/vTest/index.html.

18. **JUnit.org.** JUnit.org. *Resources for Test Driven Development.* [En línea] http://www.junit.org/.

19. **Oracle Corporation.** Netbeans. *Descarga Netbenas IDE 6.9.1.* [En línea] http://netbeans.org/downloads/6.9.1/.

20. **The Eclipse Foundation.** Eclipse. *Eclipse Downloads.* [En línea] 2012. http://www.eclipse.org/downloads/.

21. **Martinez, Davis.** Hackerdude. *Software Development Blog.* [En línea] http://www.hackerdude.com/courses/rails/Teoria_PruebasDeUnidad.html.

22. **SoftLogica.** WAPT Web Application Testing. *WAPT 7.5.* [En línea] 2003. [Citado el: 29 de Enero de 2012.] http://www.loadtestingtool.com/.

23. **Paessler AG.** PAESSLER the network monitoring company. *Webserver Stress Tool.* [En línea] 1998. [Citado el: 29 de Enero de 2012.] http://www.paessler.com/webstress.

24. **Verisium.** VERSIUM. *VPerformer Load Testing and Performance Testing.* [En línea] 2011. [Citado el: 29 de Enero de 2012.] http://www.verisium.com/products/vPerformer/index.html.

25. **Apache Software Foundation.** Apache Software Foundation. *Apache JMeter.* [En línea] 1999. http://jmeter.apache.org/.

26. **OPTIMA Technology.** Optima Technology. *WebLOAD.* [En línea] 1991. http://www.optima.com.mx/webload.htm.

27. **RadView Software Ltd.** WebLOAD Test with Confidence. *WebLOAD Professional vs WebLOAD Open Source Load Generation Engine.* [En línea] 2011. http://www.webload.org/.

28. **RadView Software Ltd**. RADVIEW Test with Cofidence. *Pricing: End user.* [En línea] 2011. http://www.radview.com/pricing/end-user-pricing.aspx.

29. **Amit.** Software Testing. *Drivers and Stubs.* [En línea] 5 de Ocotubre de 2007. http://amit-badola.blogspot.com/2007/10/drivers-and-stubs.html.

30. **SeleniumHQ.** SeleniumHQ Web application testing system. *What is Selenium?* [En línea] http://seleniumhq.org/.

CAPTURA DE REQUISITOS

Analista: nombre del analista que capturó el requisito **Iteración:** número de iteración en la que se encuentra el avance del proyecto

Sistema: nombre del sistema

Requisito	Prioridad	¿se puede probar? s/N
Se coloca una descripción en lenguaje natural de la interpretación del requisito.	Nivel de importancia determinado por el cliente.	Determinado por el equipo de pruebas se dividen todos los requisitos en probables o no probables, para con ellos realizar el plan de pruebas.

PLAN DE PRUEBAS

Ingeniero de prueba: ___________

Sistema: ___________ Módulo: ___________

No. Caso de uso	Caso de uso	Objetivo de la prueba	Éxito de la prueba	Implementación de la prueba	Asignación	Prioridad
Se asigna un identificador al requisito el cual debe ser único.	Se coloca el nombre del caso de uso	Se fijan los objetivos que tiene la prueba.	Se establece una condición para considerar que a prueba fue exitosa.	Se define la forma en la que se implementará la prueba ya sea de forma manual o con alguna herramienta de automatización. Si se usa una herramienta está deberá ser indicada.	Es el nombre del Ingeniero de componentes que estará encargado de elaborar la prueba.	Nivel de importancia, que dependiendo del criterio que se esté manejando, asignado por el equipo de pruebas.

EJECUCIÓN DE LA PRUEBA

Ingeniero de pruebas de sistema: ____________ Sistema: ____________

Ingeniero de componente: ____________

Ingeniero de prueba: ____________ Módulo: ____________

Identificador	Caso prueba	Valor esperado	Valor obtenido	Resultado de la prueba	Comentarios
Es el identificador que se asignó al requisito o al caso de uso.	Son los pasos para realizar la prueba.	Es la respuesta del sistema que se espera obtener.	Es la reacción real del sistema.	Dependiendo del criterio de aceptación de la prueba definido en el plan de pruebas se determina si la prueba fue aprobada o no lo fue.	Son notas que deja el Ingeniero de pruebas de sistema para el Ingeniero de pruebas. Principalmente si el error de la prueba fue por causa del componente y no del sistema.

DETECCIÓN DE DEFECTOS

Ingeniero de pruebas de sistema: _____

Sistema: _____ Módulo: _____

Identificador	Descripción del defecto			Prioridad	Estado
	Acciones	Lo que hace actualmente	Lo que debería hacer		
Es el identificador del caso de uso o requisito (anteriormente asignado) en donde la prueba no fue aceptada debido a un defecto.	Es la secuencia de pasos que se dieron para que ocurriese el defecto.	Descripción de la respuesta del sistema después de ejecutar las acciones indicadas.	La respuesta del sistema que debería ocurrir después de realizar las mismas acciones que generaron el defecto.	Nivel de afectación a la funcionalidad del sistema.	Referente a si el defecto ha sido corregido, amortiguado o no ha sido resuelto.

CORRECIÓN DE DEFECTOS

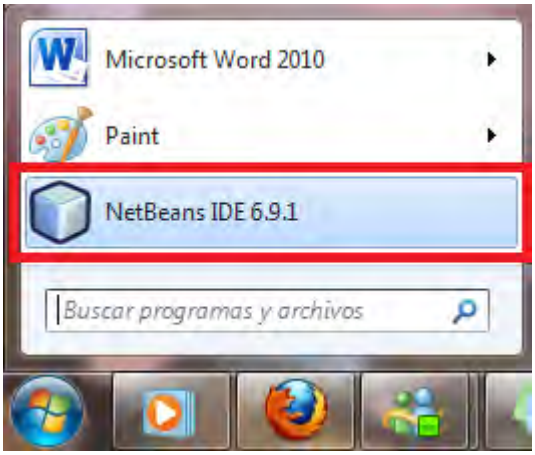
Ingeniero de pruebas de sistema: _____

Sistema: _____ Módulo: _____

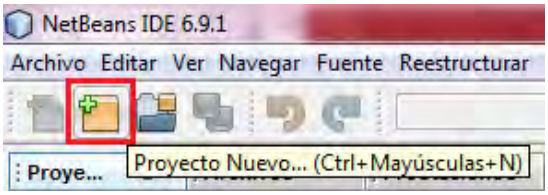
Identificador	Causa	Solución
Es el identificador del caso de uso o del requisito donde se aplico una prueba y se hayó el defecto.	EL motivo por el cual surgia el defecto.	La acción que se tomó para corregir el defecto o amortiguar su efecto

Crear un proyecto en NetBeans IDE

Paso 1: Se inicia el IDE, dando clic en el ícono de NetBeans.



Paso 2: Después de que se haya iniciado el IDE se da clic en el ícono para crear un nuevo proyecto.

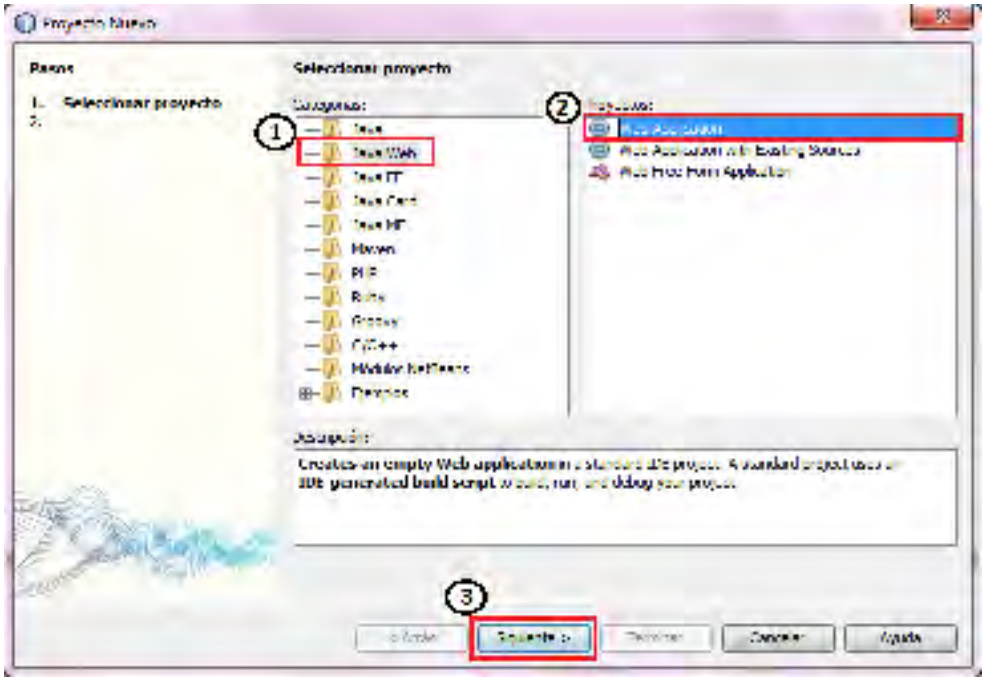


Paso 3: Seleccionar el tipo del proyecto.

Para aplicaciones Web se selecciona “Java Web”.

Después se selecciona “Web Application”.

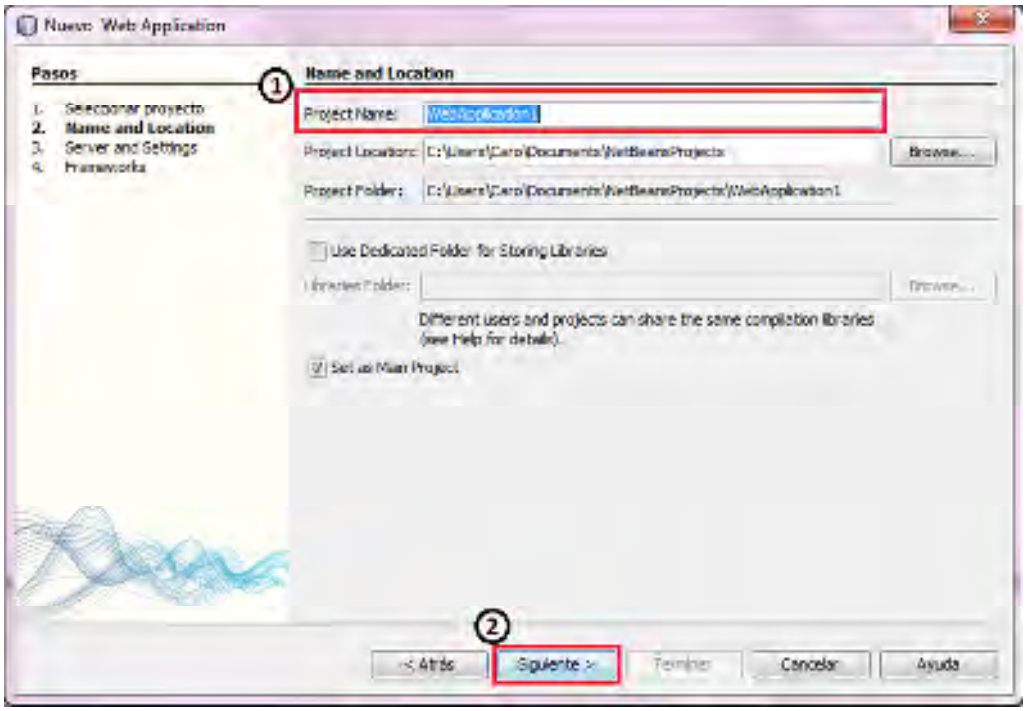
Dar clic en el botón “Siguiente”.



Paso 4: Escribir el nombre del proyecto.

Se le da un nombre al proyecto.

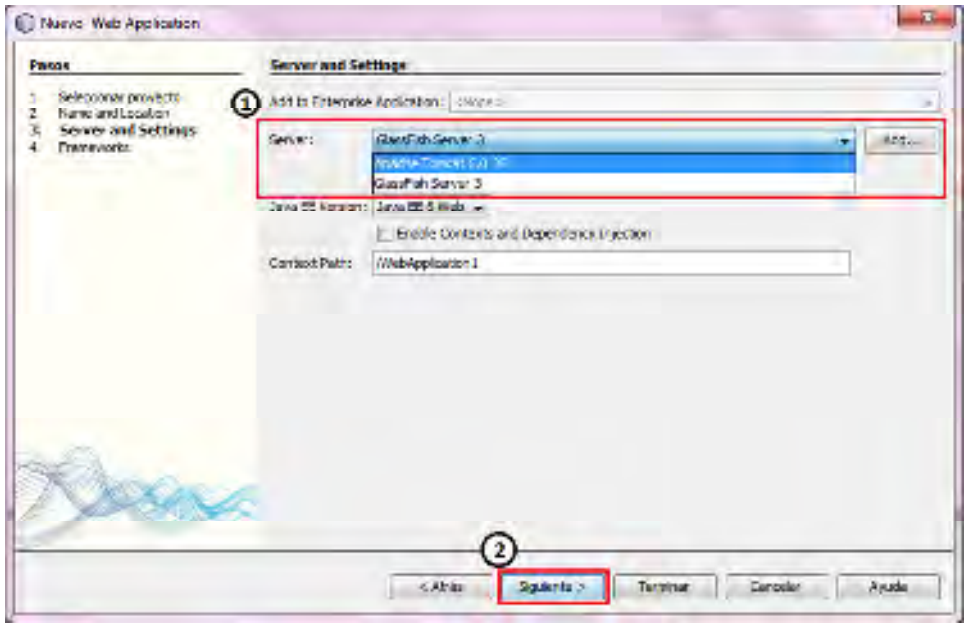
Se da clic en “Siguiente”.



Paso 5: Seleccionar servidor y configurar el proyecto.

Para este caso se seleccionó el servidor de Apache Tomcat 6.0.26.

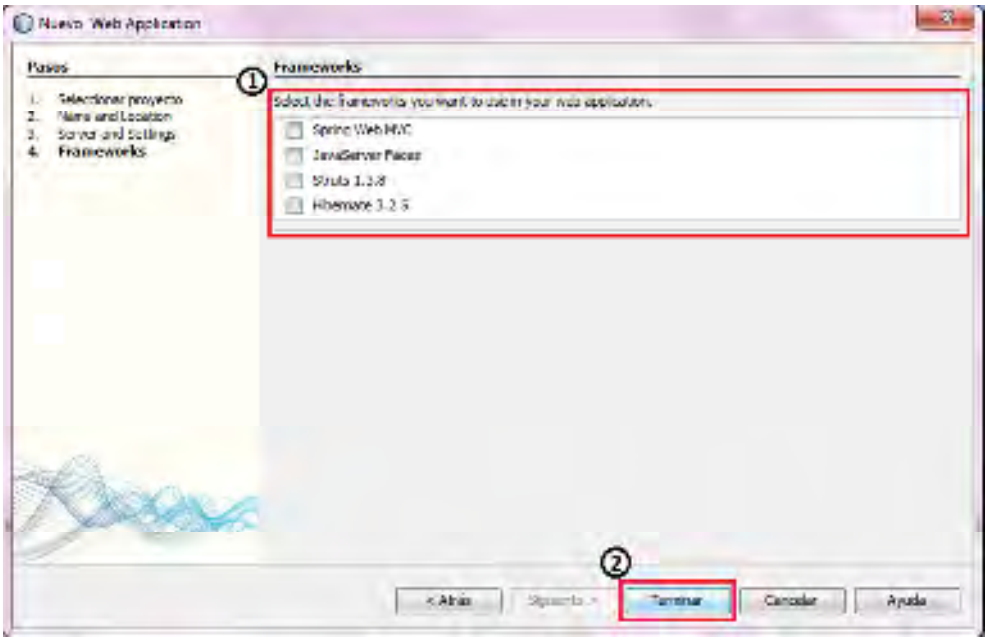
Dar clic en el botón “Siguiente”



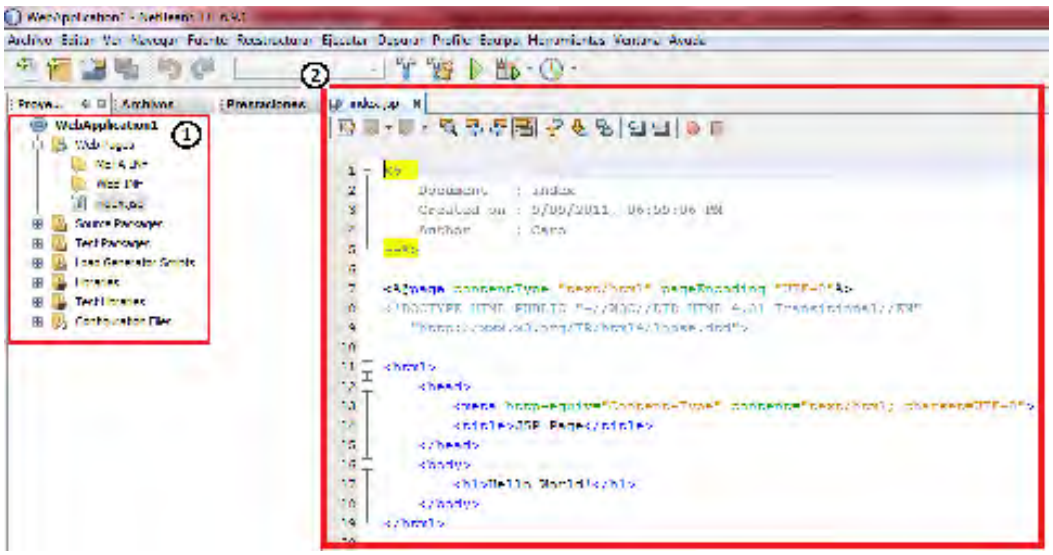
Paso 6: Seleccionar algún framework (opcional)

Lo que sucede al seleccionar alguno de los frameworks disponibles es que se usa la estructura propuesta por el framework para el diseño del sistema.

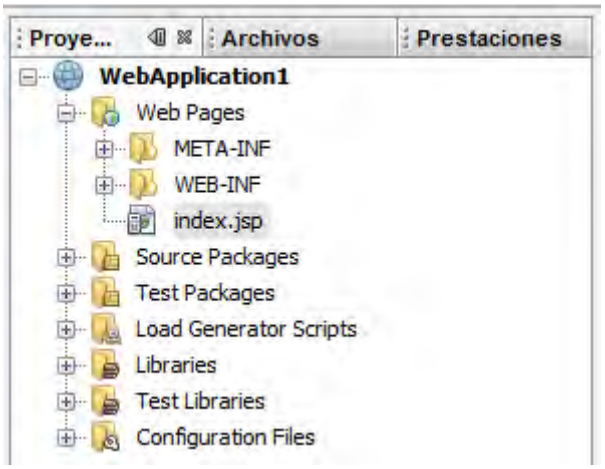
Se pulsa el botón “Finalizar”.



Al finalizar la creación del proyecto se tiene la siguiente vista. En (1) se muestra la estructura en carpetas del proyecto y en (2) se visualiza la plantilla para el archivo principal del proyecto recién creado.



La estructura del proyecto es la siguiente:



Web Pages. En esta carpeta se almacenarán todas las páginas web creadas.

Source Packages. Esta carpeta contendrá todas las clases java para la funcionalidad de la aplicación web.

Test Packages. Contiene las clases java para la prueba del proyecto.

Load Generator Scripts. Se almacenan los scripts para las pruebas de rendimiento del sistema en una carga de trabajo típica.

Libraries. Contiene las bibliotecas necesarias para el correcto funcionamiento del sistema.

Test Libraries. Contiene las bibliotecas necesarias para la correcta ejecución de las pruebas.

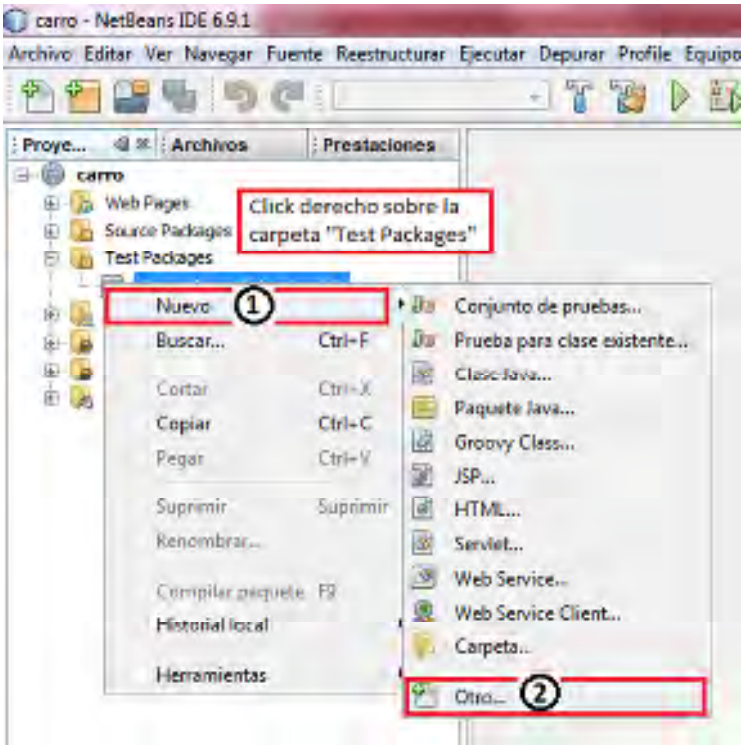
Configuration Files. Están todos los archivos necesarios para la configuración del sistema en un modo de acceso directo.

Cómo crear una prueba JUnit en NetBeans 6.9.1

Paso 1: Seleccionar el tipo de archivo para la prueba

Sobre el proyecto donde se van a realizar las pruebas. Se localiza la carpeta “Test packages”. Con clic derecho sobre la carpeta “Test Packages” en “Nuevo”

De la lista desplegable seleccionar “Otro…”

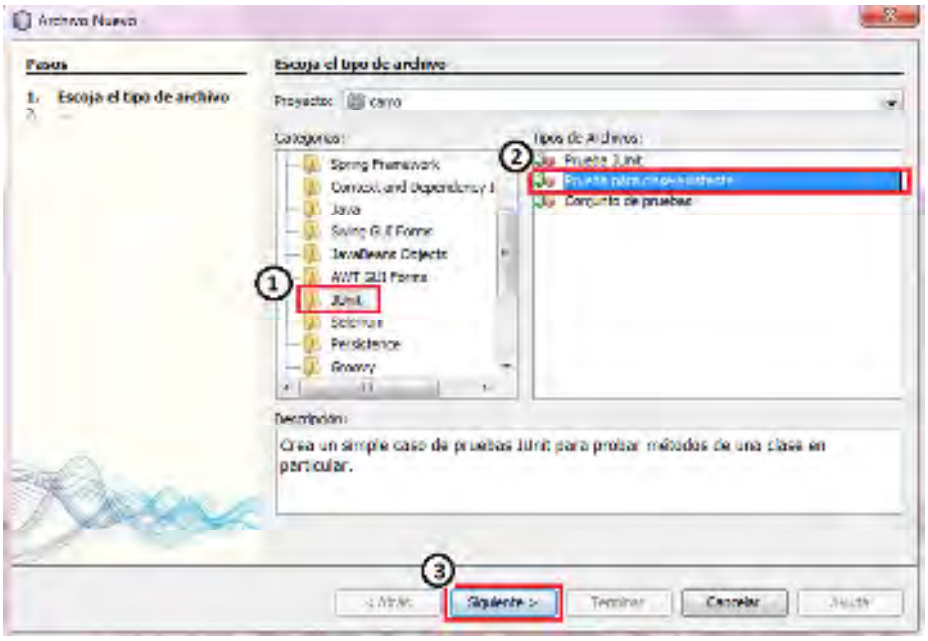


Paso 2: Aparecerá una ventana para seleccionar el tipo de archivo que se desea crear. En este caso se está creando una prueba en JUnit.

En la sección de “Categorías” se busca la carpeta JUnit y se selecciona para que en la sección “Tipos de Archivos” se despliegue la lista de archivos disponibles.

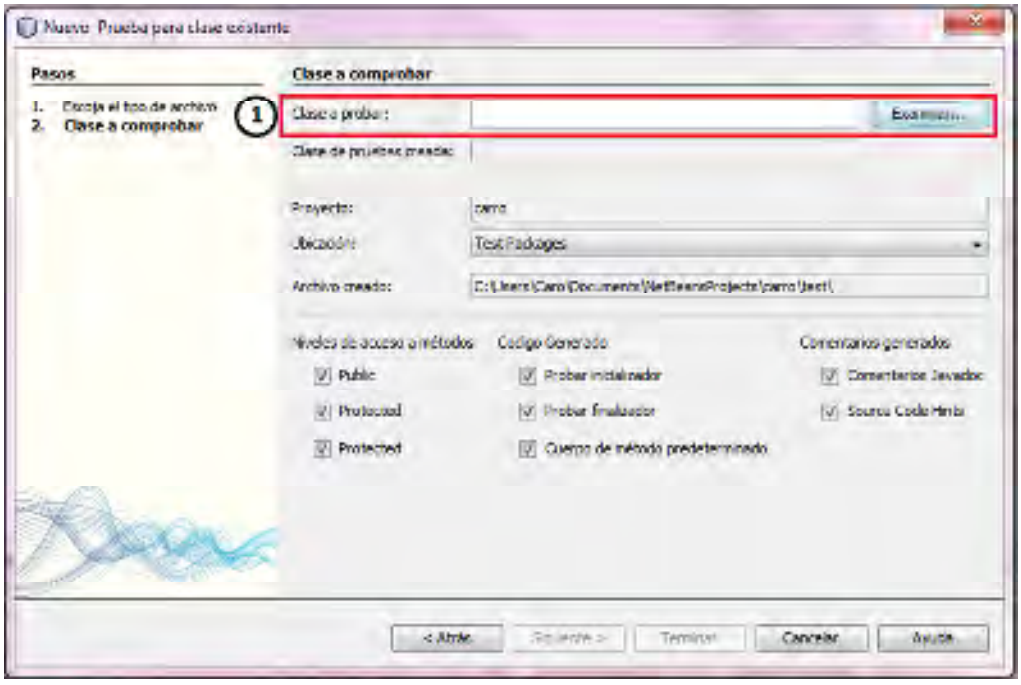
Para crear al plantilla de la prueba se selecciona “Prueba para clase existente”.

Se da clic en “Siguiente”.

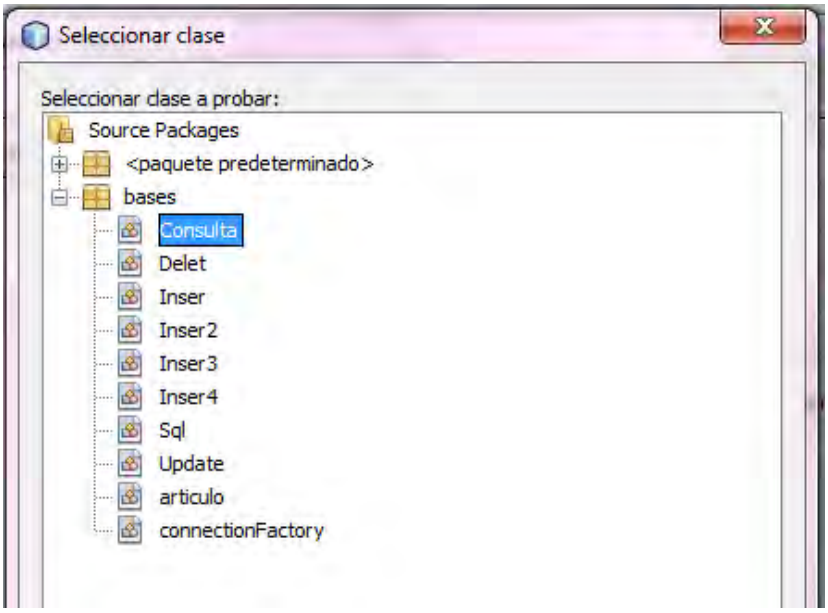


Paso 3: Buscar la clase a la cual se la ve aplicar la prueba.

Se da clic en el botón “Examinar…”



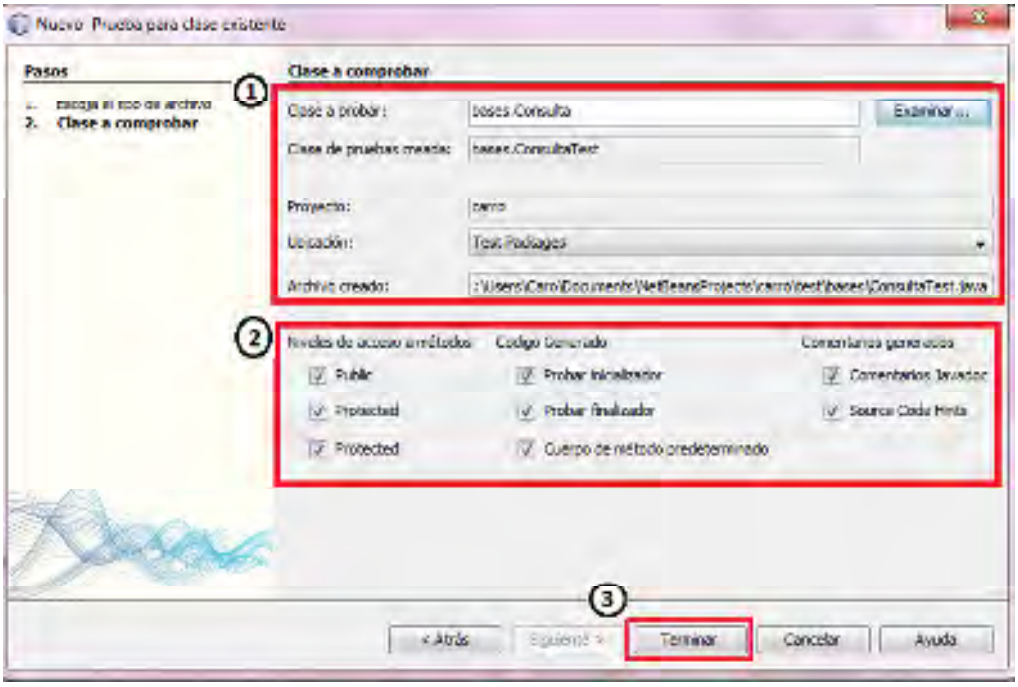
Paso 4: Aparecerá una ventana con las clases existentes en el proyecto ubicadas en el paquete “Source Packages”. Se selecciona la clase a la que se desea crear la prueba.

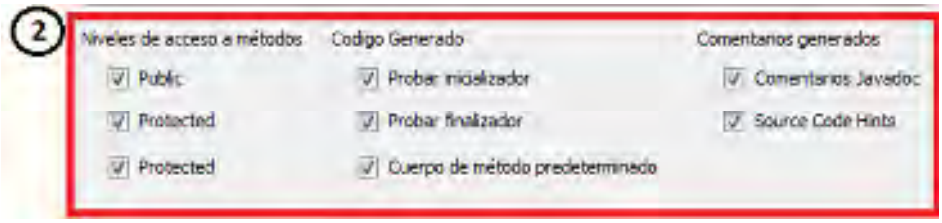


Paso 5: Selección de parámetros a probar.

Automáticamente se llenan los espacios vacíos y por si se tiene más de un paquete de clases en "Source Packages" las pruebas se guardan en paquetes con el mismo nombre. Para el ejemplo hay un paquete bases en "Source Packages" y su equivalente en "Test Packages".

Selección de parámetros que se desean probar.





Niveles de acceso a métodos. Se van a generar pruebas para los métodos que sean:

- Public
- Protected
- Protected (predeterminados)

Código generado. Se puede escribir la prueba desde cero o pedir que se genere código para el inicializador, el finalizador y para los métodos a probar.

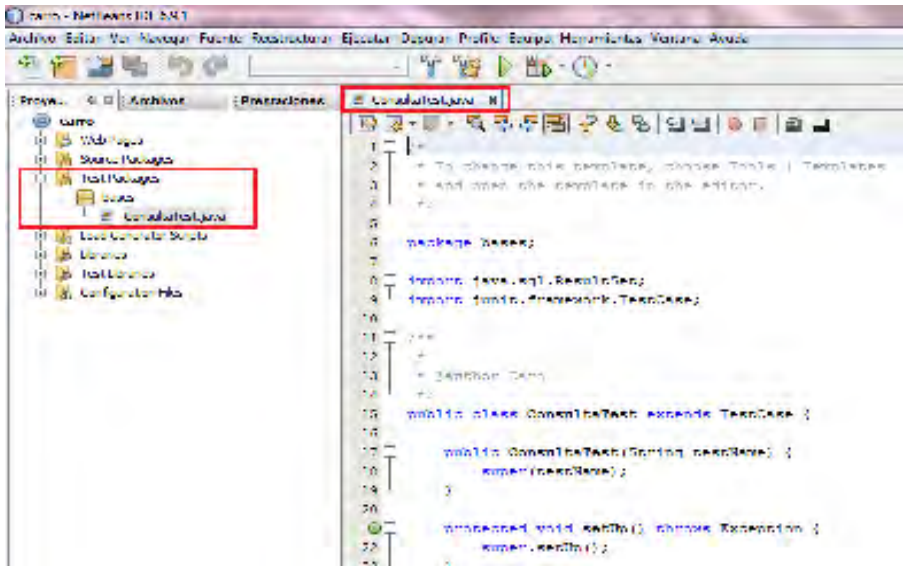
- Probar inicializador
- Probar finalizador
- Cuerpo de método predeterminado

Comentarios generados. Es opcional la generación de cometarios donde están los de Javadoc y las sugerencias para los cuerpos de los métodos.

- Comentarios Javadoc
- Source code hints

Se da clic en el botón “Terminar”.

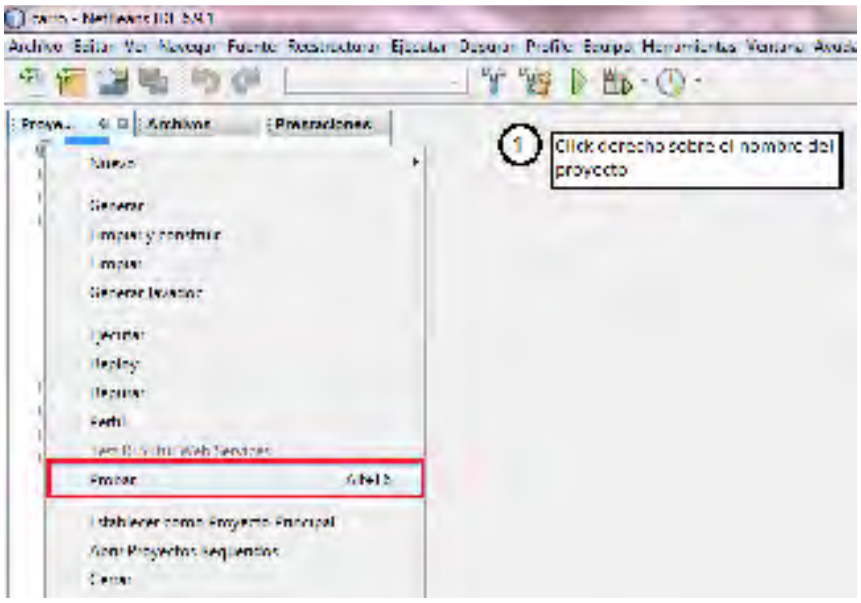
Paso 6: Se puede observar que se creó el paquete para las pruebas y la plantilla para la clase de prueba para la clase existente seleccionada con su respectivo código.



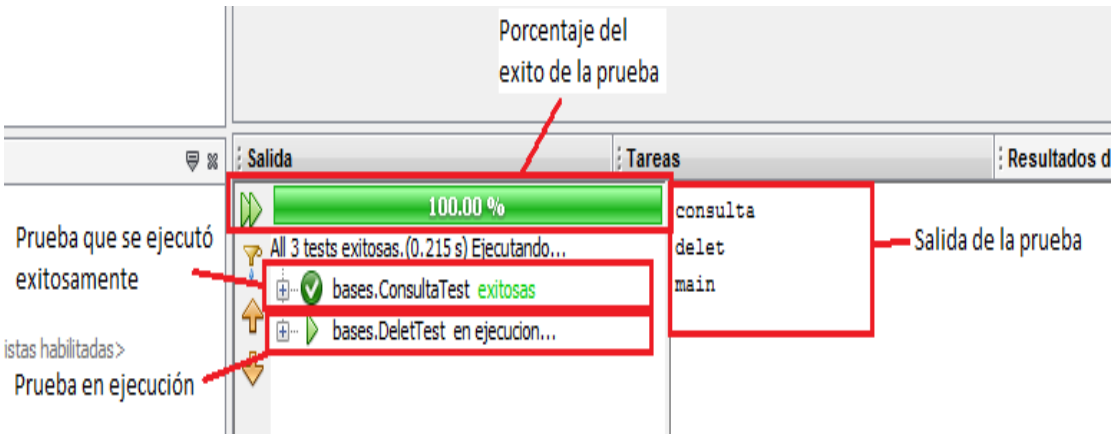
Cómo ejecutar un conjunto de pruebas

Paso 1: Ejecución de las pruebas.

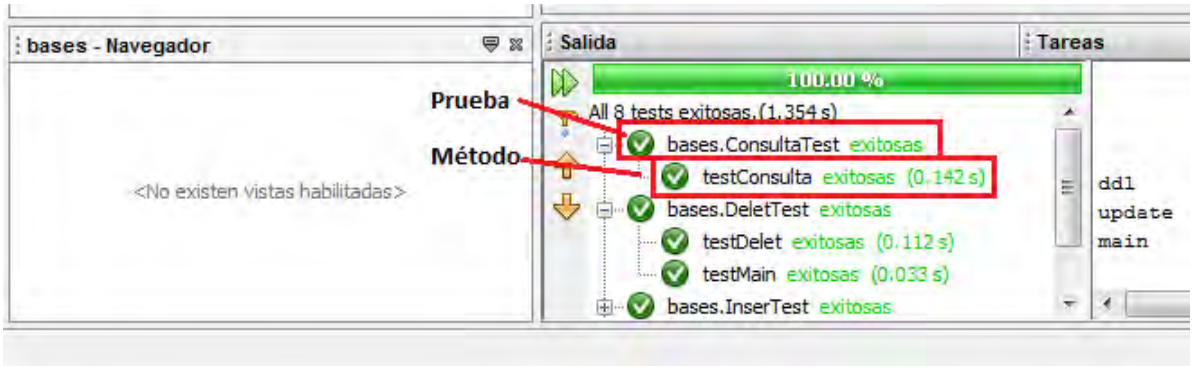
Sobre el nombre del proyecto se da clic derecho. En el menú que se despliega se busca la opción de “Probar”.



Paso 2: En la parte inferior de la ventana de NetBeans se visualiza la salida de la ejecución de la prueba.



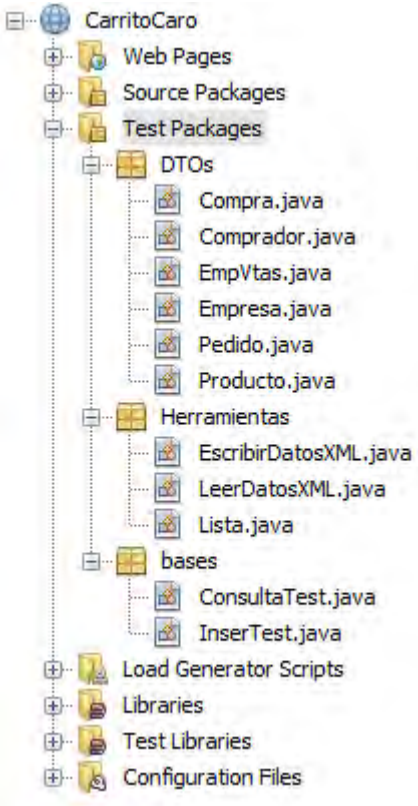
Paso 3: Al finalizar la ejecución el resultado del test se ve de la siguiente forma.



Se muestra el resultado de la prueba (si fue exitosa o no) y el tiempo que se tardó en ejecutar la prueba.

Organizando y creando un paquete de pruebas

Para realizar las pruebas es recomendable separar las clases para dejar en un grupo las clases que están realizando la prueba, en otro grupo las clases que nos sirven para realizar la clase prueba y un tercer grupo para las clases que representan a la base de datos.



En el paquete “Test Packages” hay tres paquetes “DTOs”, “Herramientas” y “bases”. En “DTOs” están ubicadas todas las clases que mapean a la base de datos del sistema y las clases presentan la siguiente estructura:

Una clase es una tabla de la base de datos por lo que en el paquete de “DTOs” habrá tantas clases como tablas en la base de datos. Las variables son los campos de la tabla. Y finalmente los métodos que habrá en la clase son los constructores y los métodos get y set, como se ve a continuación en el código.

```
public class Compra {
private String id_compra;
private BigDecimal precioParcial;
private String id_producto;

   public Compra() { }

   public Compra(String id_compra, BigDecimal precioParcial, String id_producto) {
      this.id_compra = id_compra;
      this.precioParcial = precioParcial;
      this.id_producto = id_producto;
   }

   public String getId_compra() {
      return id_compra;
   }

   public void setId_compra(String id_compra) {
      this.id_compra = id_compra;
   }

   public String getId_producto() {
      return id_producto;
   }

   public void setId_producto(String id_producto) {
      this.id_producto = id_producto;
   }

   public BigDecimal getPrecioParcial() {
      return precioParcial;
   }

   public void setPrecioParcial(BigDecimal precioParcial) {
      this.precioParcial = precioParcial;
   }

}
```

En el paquete “bases” se colocan las clases que van a realizar la prueba. Donde su estructura es la siguiente:

```
package bases;

import Herramientas.Lista;
import Herramientas.EscribirDatosXML;
import DTOs.Producto;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import junit.framework.TestCase;
import org.dbunit.database.DatabaseConnection;
import org.dbunit.database.IDatabaseConnection;

public class ConsultaTest extends TestCase {
   ResultSet consultaPrueba;

   /*Constructor*/
   public ConsultaTest(String testName) {
      super(testName);    }

   /*Iniciallizar objetos a probar*/
   @Override
   protected void setUp() throws Exception {
      super.setUp();
      // Se inicializa la coneccion de la base de datos
      IDatabaseConnection connection = null;
      consultaPrueba=null;           }

   /*Libera los objetos a probar*/
   @Override
   protected void tearDown() throws Exception {
      super.tearDown();   }

   /*Se crea la coneccion a la base de datos*/
   protected IDatabaseConnection getConnection() throws Exception{
        Class driverClass = Class.forName("org:postgresql://127.0.0.1:5432/MiBD");
        Connection jdbcConnection =
             DriverManager.getConnection("jdbc:postgresql://localhost:5432/MiBD",
"postgres", "postgres");
        return new DatabaseConnection(jdbcConnection);    }
/**
    *Prueba del método consulta de la clase Consulta.
    */
   public void testConsulta() throws Exception {

       //Las consultas que se desean probar. En este caso se realizaron bajo la tabla de
producto
       String SQL1="SELECT id_pro, nom_pro, precio, descripcion,pro_rela,
existencias, imagen FROM producto;";
       String SQL2="SELECT nom_pro FROM producto WHERE id_pro=123;";
```

```
        //Se realiza la consulta
        consultaPrueba = new Consulta().consulta(SQL1);

        //Se pregunta si la consulta no es nula. eso quiere decir que la consulta
        //trajo algunos datos.
        if(consultaPrueba!=null){

            Integer idPro=0;
            String nomPro="";
            String descripcion="";
            String proRela="";
            String imagen="";
            int existencias=0;
            int precio=0;
            Lista productosConsultados = new Lista();

            while(consultaPrueba.next())
            {
                //Se separa cada elemento de la consulta
                idPro=consultaPrueba.getInt("id_pro");
                nomPro=consultaPrueba.getString("nom_pro");
                descripcion=consultaPrueba.getString("descripcion");
                proRela=consultaPrueba.getString("pro_rela");
                imagen=consultaPrueba.getString("imagen");
                existencias=consultaPrueba.getInt("existencias");
                precio=consultaPrueba.getInt("precio");

                //Se encansupla en la clase de producto
                Producto proCon = new Producto();
                proCon.setId_producto(idPro.toString());
                proCon.setNom_pro(nomPro);
                proCon.setDescripcion(descripcion);
                proCon.setPro_rela(proRela);
                proCon.setImagen(imagen);
                proCon.setExistencias(new BigDecimal(existencias));
                proCon.setPrecio(new BigDecimal(precio));
    //Se agrega el producto a la lista de productos
                productosConsultados.add(proCon);           }
            //Se escriben todos los producots consultados en un archivo XML
            EscribirDatosXML consultaXML = new
    EscribirDatosXML(productosConsultados);
            System.out.println("Se creó el archivo XML");


        }else{
            //se llama al error por tener una consulta vacía
            fail("El resultado de la consulta fue nula");}
        }
}
```

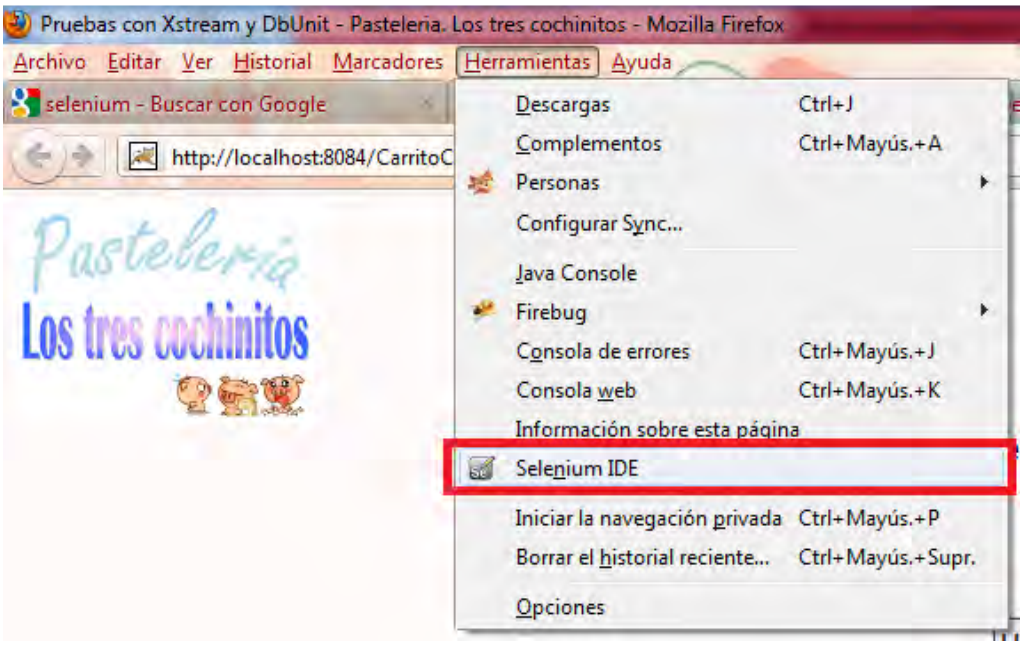
Los métodos principales que debe llevar son el constructor, el setUp(), tearDown() y el método que va a realizar la prueba. En el setUp() se inicializan las variables a utilizar para la prueba y en tearDown() se liberan todos los recursos utilizados durante la prueba. Puede haber más de un método que realice la prueba, todo depende del diseño de la prueba, de qué tan detallada se desea hacer. Un método puede probar toda la clase que se está probando o puede haber un método que pruebe cada método en la clase.

En el paquete “Herramientas” van las clases que sirven para hacer operaciones intermedias entre las clases del paquete “DTOs” y “bases”. Y la estructura de estas clases solo consiste en los métodos que realizan las operaciones deseadas.

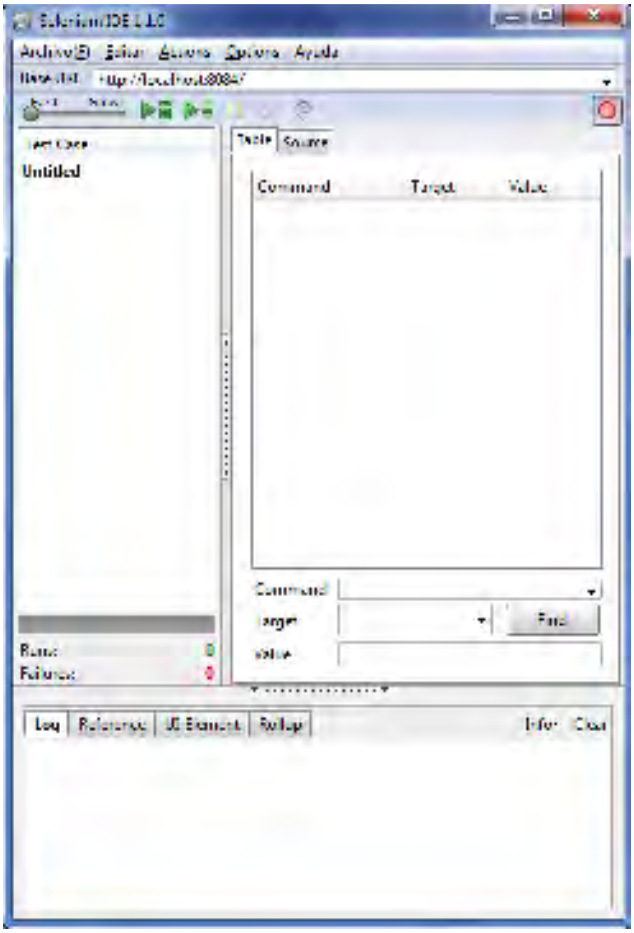
Crear una prueba en Selenium IDE

Paso 1: Iniciar el IDE

Después de la instalación Selenium quedó instalado en el menú de herramientas del navegador Firefox.



En seguida aparecerá la ventana del IDE.



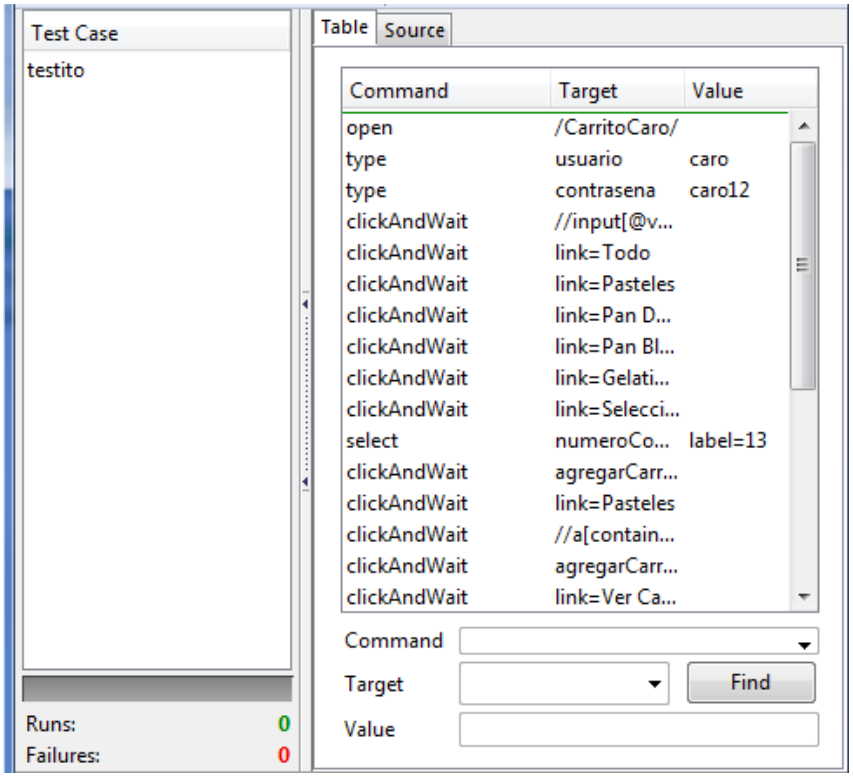
Paso 2: Crear la prueba por medio de grabación de acciones

El botón ubicado al extremo derecho es el que inicia la grabación de la prueba. 

Para iniciar la prueba se debe estar sobre la aplicación que se va a probar. Dar clic en el botón y empezar a realizar las acciones que se desean verificar.

Por ejemplo, tenemos una página web la cual tiene para iniciar debe introducirse el nombre de usuario y contraseña para acceder al contenido y se desea probar que el inicio de sesión funcione correctamente. Entonces se comienza a grabar, se escribe el nombre usuario, la contraseña, se da clic en él botón para iniciar sesión (aceptar, login, entrar, etc.) y se detiene la grabación de la prueba oprimiendo el botón de grabación. Después de realizar esta acción la prueba se puede ejecutar tantas veces se desee con el botón de iniciar prueba  (ejecutará la prueba completa) o con  (que ejecutará la prueba por pasos).

Después de la grabación la prueba se ve así en el IDE.



Del lado izquierdo se muestra la fila de pruebas que se van a ejecutar (en el ejemplo solo hay una prueba) y en la parte inferior se puede ver cuántas fueron ejecutadas exitosamente y cuántas fallaron. Del lado derecho tenemos la lista de comando que se ejecutan en la prueba. En la parte inferior se puede agregar comando de manera manual.