“Facultad 5"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Título: “Automatización de Pruebas Funcionales para Aplicaciones Web.”

Autor(es): Mario Barrientos Rodríguez.
Yamila Nieves Hernández.

Tutor(es): Ing. Amado Espinosa Hidalgo.
Ing. Enerys Mesa Morales.

Ciudad de La Habana, 2010.
Año 52 del Triunfo de la Revolución.

Hay una regla para los industriales que es: haz los productos de la mejor calidad posible al menor costo posible."

Henry Ford

Agradecimientos

De Mario:

A mis padres, por decir presente a mi llamado, por guiarme con paciencia y dedicación, y darme el aliento a seguir adelante cuando me sentía derrotado. Por confiar en mí; sin ustedes, este momento no hubiese sido posible.

A toda mi familia, por ser parte importante en mi vida y sentirse siempre orgullosos de mí.

A mi segunda madre, Susana, por todo el apoyo brindado en el momento más duro de la carrera.

A mis tutores Amado y Enerys, por brindarme sus conocimientos y guiarme durante toda la investigación.

A todas las personas que brindaron apoyo al desarrollo de la investigación, Ludisley, Ana Silvia, Delmys, Yunetsy Medina, Yadira Ramírez. Gracias por sus conocimientos. Fueron imprescindibles para terminar el trabajo.

A todos mis amigos, sin excepción de ninguno, hay un lugar muy especial para cada uno de ellos en mi corazón. Gracias por estar ahí.

A todas las personas que tanto quiero y que han estado a mi lado durante estos largos años…por todos los momentos inolvidables.

A todos los que me han brindado su apoyo y forman parte de este logro.

A todos….Gracias

De Yamila:

A mi mamá, por todos estos años de esfuerzo, de sacrificio y dedicación, porque sin ella no sería lo que soy, por guiarme siempre y llenarme de su inmenso cariño y sensibilidad.

Por enseñarme a crecerme antes las dificultades y por quererme como ella solo sabe hacerlo. Por enseñarme a dar lo mejor de mí y por ser la mejor madre del mundo. Por comprenderme siempre, confiar tanto en mí y darme tantos consejos, este triunfo es tuyo. Te quiero mami.

A mi hermanita del alma Yanelis por ser un gran ejemplo y la mejor del mundo.

A toda mi familia por preocuparse por mí y a mi padrastro por estar siempre para mí.

A mis compañeros de aula que estuvieron junto a mí durante estos cinco años que me hicieron pasar momentos muy agradables.

A todos mis profesores por contribuir en mi formación profesional y por guiarme en esta difícil tarea.

A mis tutores Amado y Enerys por guiarnos en el desarrollo de la investigación.

A todos los que preguntaron por mis estudios alguna vez.

Gracias.

Dedicatoria

A mis padres por todo lo que significan para mí y por su incansable apoyo a lo largo de todos estos años.

A Yamila, mi compañera de tesis, por la paciencia y confianza, sin ella no hubiéramos logrado este triunfo.

A todos los que de una forma u otra me han ayudado a llegar hasta aquí, por su confianza, su apoyo, su amor.

Mario

A mi mamita y a mi hermanita por su apoyo y amor.

A mi compañero de tesis Mario, pues juntos pudimos hacer todo esto.

A la Revolución y a la Universidad.

Yamila

Declaración de autoría

Declaramos ser los únicos autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Mario Barrientos Rodríguez

_____

Firma del autor

Yamila Nieves Hernández

_____

Firma de la autora

Ing. Amado Espinosa Hidalgo

_____

Firma del tutor

Ing. Enerys Mesa Morales

_____

Firma de la tutora

Resumen

Las herramientas de automatización han posibilitado que la intervención del ser humano en muchos procesos sea sustituida por el software automatizado; el flujo de prueba específicamente es muy importante en el desarrollo de un software, por lo se hizo necesario realizar un estudio comparativo de las herramientas existentes, llegando a la conclusión de que la herramienta que más requisitos cumple para las pruebas funcionales automatizadas al sistema que sea seleccionado es Selenium. Este trabajo está centrado en elaborar un procedimiento que guíe el desarrollo de pruebas funcionales automatizadas a las aplicaciones web del centro. Con estas pruebas se examina exhaustivamente el sistema para identificar los posibles errores que se están generando y poder eliminarlos, permitiendo: aumentar la calidad del producto, comprender el sistema que está siendo probado y aumentar el riesgo de encontrar errores. Los aportes principales del mismo consisten en la obtención de una documentación organizada bajo la metodología de desarrollo RUP, Proceso Unificado de Rational, con el objetivo de lograr una óptima organización en el desarrollo de las pruebas.

Palabras claves

Pruebas de software, pruebas automatizadas, herramientas de automatización, aplicaciones Web.

Índice de contenidos

Introducción

La Industria Cubana del Software (ICS) exhibe un lugar de privilegio en la economía cubana gracias al exitoso aprovechamiento del capital humano con el que dispone, posibilitando un aumento significativo en la exportación de software de alta calidad y en la promoción de la misma en el ámbito internacional, aprovechando la enorme credibilidad que tiene Cuba en sectores tales como la salud, la educación y el deporte.

Dentro del programa de informatización de la sociedad cubana se encuentra la Universidad de las Ciencias Informáticas (UCI); universidad de nuevo tipo que une formación y producción, lo que le permite jugar un rol significativo en el avance de esta industria, propiciando la ejecución de los diversos proyectos asociados al programa cubano de informatización. La UCI ha contribuido con el cumplimiento de estas tareas, concentrándose en el desarrollo de proyectos productivos e investigativos de interés, por encargo de la sociedad cubana y de otros países, en campos como educación, salud, deporte, gobierno en línea, Software Libre, sitios y portales Web, productos multimedia y otros en los nuevos centros generadores de soluciones integrales, productos y servicios informáticos creados en la universidad.

En uno de estos centros, el Centro de Desarrollo de Informática Industrial (CEDIN), se consolidan como rubro exportable los productos y servicios ofrecidos, con una alta competitividad basada en principios de Software Libre y de reusabilidad, combinada con una atención y soporte rápidos centrada en los usuarios. Antes de liberar un producto se debe validar el cumplimiento de los requisitos especificados por el cliente, proceso en el que juega un papel significativo las pruebas de software, elemento crítico que influye en la calidad del producto y que constituye una revisión final de las especificaciones del diseño y la codificación.

En las aplicaciones Web desarrolladas en el centro se identifica la siguiente problemática: Las pruebas funcionales, encargadas de verificar si el sistema cumple los requisitos establecidos por los clientes y los desarrolladores del mismo se realizan de forma manual. La etapa de pruebas es una de las más lentas del proceso de desarrollo de un software debido a la complejidad que requiere probar una

aplicación, ya que es necesario controlar la ejecución de las pruebas, comparar los resultados, preparar las precondiciones y confeccionar los informes, actividades demandan mucha paciencia y atención al detalle por parte de las personas encargadas de las pruebas. Entre las desventajas de las pruebas manuales se distinguen el elevado esfuerzo del grupo de pruebas que incrementa los costos de producción, la dilatación de los tiempos de entrega y la limitación al desarrollo de pruebas de mayor nivel de complejidad.

A partir de la situación problémica expuesta anteriormente, se formula el siguiente **problema científico**: ¿Cómo introducir las herramientas de automatización en las pruebas funcionales a las aplicaciones Web desarrolladas en el centro; que facilite el trabajo del probador, posibilitando encontrar errores de mayor magnitud en el menor tiempo y con la mínima cantidad de recursos posibles?

Por tanto, como **objeto de estudio** se plantea las pruebas funcionales a aplicaciones web.

Se establecen como **objetivos generales**: Seleccionar una herramienta de automatización para las pruebas funcionales a las aplicaciones Web desarrolladas en el centro. Elaborar un procedimiento que guíe la utilización de la herramienta seleccionada en las pruebas funcionales a las aplicaciones Web desarrolladas en el centro.

Por lo que el **campo de acción**, elegido es herramientas de automatización de pruebas funcionales.

Para dar cumplimiento al objetivo general se plantean las **tareas de investigación** siguientes:

1. Realización de un estudio del estado del arte de las herramientas de automatización de pruebas funcionales a aplicaciones Web.
2. Definición del alcance de la solución a través de un plan de pruebas funcionales.
3. Diseño de los casos de pruebas funcionales.
4. Incorporación de los casos de pruebas a la herramienta seleccionada.
5. Confección del manual de uso de la herramienta seleccionada.
6. Ejecución de las pruebas diseñadas.
7. Valoración de los resultados obtenidos en las pruebas.

Se plantea como **idea a defender** que: con la ejecución de las pruebas funcionales automatizadas a las aplicaciones web desarrolladas en el centro; se facilitará el trabajo de los probadores, posibilitándoles identificar errores de mayor magnitud en el menor tiempo y con la mínima cantidad de recursos posibles.

Para darle cumplimiento al objetivo general y respuesta al problema científico planteado fueron aplicados los siguientes **métodos científicos**: dentro de los **métodos teóricos** fueron utilizados el analítico-sintético para el procesamiento de la información relacionada con las pruebas de software y las herramientas de automatización de las pruebas funcionales de aplicaciones web recopilada durante la investigación; inductivo-deductivo, para extraer las regularidades presentes en las pruebas manuales a las aplicaciones web y elaborar conclusiones de la automatización de estas pruebas; histórico-lógico para constatar la evolución histórica de las herramientas usadas con estos fines y el matemático, para registrar en una tabla resumen los resultados obtenidos con las pruebas automatizadas y posteriormente compararlos con los resultados de las pruebas manuales, permitiendo demostrar cuál de las formas de realizar las pruebas es más rápida y eficiente. Además se emplean los **métodos empíricos** tales como: el análisis de las fuentes de información, empleado para obtener resultados confiables relacionados con las pruebas de interfaces de usuarios en aplicaciones web, la experimentación para registrar de forma visual todos lo relacionado con las pruebas de software y las herramientas de software libre utilizadas actualmente para automatizar estas pruebas, así como la realización de encuestas (Anexos 7 y 8) que le permitieron a los autores conocer cómo se realizan las pruebas funcionales en las aplicaciones web desarrolladas en el centro.

El trabajo fue dividido en tres capítulos. A continuación se muestra el nombre de cada uno con una breve explicación de su contenido:

Fundamentación teórica de la investigación, se lleva a cabo un estudio de las tendencias actuales relacionadas con la automatización de las pruebas funcionales en aplicaciones Web, se describen y comparan las herramientas candidatas, y se selecciona la más óptima para la automatización de estas pruebas en las aplicaciones web desarrolladas en el CEDIN.

Descripción del procedimiento para pruebas funcionales automatizadas, se explica cómo realizar de forma automatizada las pruebas funcionales en aplicaciones Web, planificando los recursos y

métodos necesarios que posibilitan la ejecución de las pruebas, con el fin de obtener como resultado una disminución sustanciosa del trabajo de los probadores.

Aplicación del procedimiento al Sistema de Información de Perforación de Pozos Petroleros, se ejecutan las pruebas, se comparan los resultados diseñados con los obtenidos y se arriba a conclusiones precisas fundamentadas en los resultados los obtenidos durante la ejecución de las pruebas.

Cada capítulo comienza con una breve introducción sobre el tema a tratar y finaliza con conclusiones parciales donde se abordan los aspectos más relevantes que se trataron.

Por último se tienen las conclusiones finales, las recomendaciones, referencias bibliográficas y los anexos.

Capítulo 1: Fundamentación teórica de la investigación.

En el presente capítulo se lleva a cabo un estudio de las tendencias actuales relacionadas con la automatización de las pruebas de software, abordando específicamente algunos conceptos afines al tema de las pruebas de software en aplicaciones Web. Además se efectúa una búsqueda de información de las principales herramientas utilizadas en los entornos de software libre, se describen y se comparan las herramientas candidatas; y se selecciona la herramienta que cumple los requisitos establecidos para la automatización de las pruebas funcionales en aplicaciones Web.

1.1 Pruebas de Software

Las pruebas de software constituyen una fase del proceso de desarrollo de un software centrada en el favorecimiento a la calidad, fiabilidad y robustez del mismo, dentro del contexto o escenario previsto para ser utilizado (15). Es por ello que han recibido la importancia que merecen, como resultado de los trabajos dedicados por diversos autores a las mismas, en los que se han ofrecido un gran número de definiciones entre las que se encuentran las relacionadas a continuación:

- **Pressman:** Las Pruebas de software pueden definirse como “el *proceso de evaluación de un producto desde un punto de vista crítico, donde el "probador" (persona que realiza las pruebas) somete al producto a una serie de acciones indagadoras, y el producto responde con su comportamiento como reacción”.* Es necesario probar los nuevos programas en un entorno de pruebas separado físicamente del de producción. (1)

- **IEEE, 1990:** Una prueba puede ser: *“Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente”. (2)*

- **RUP:** Una prueba es *“una disciplina en el proceso de ingeniería de software cuyo objetivo es integrar y poner a prueba el sistema” (3).* No es posible garantizar que un sistema esté correcto solo usando pruebas. Así que aunque se hagan esfuerzos extraordinarios, no podemos decir que el sistema está libre de defectos (4).

La importancia de las pruebas dentro del desarrollo del software se puede visualizar teniendo como referencia a los autores aludidos a continuación:

- **Dustin:** Las pruebas de software permiten pasar de forma confiable del cómodo ambiente planteado por la ingeniería de software, es decir del controlado ambiente de análisis, diseño y construcción, al exigente mundo real en el cual los entornos de producción someten los productos a todo tipo de fatiga (5).

- **Zuyu, Tsao, Wu:** Las pruebas de software basadas en componentes permiten la reutilización y por ende la reducción de los ciclos de pruebas, lo cual se ve reflejado en la disminución de costos y tiempos (6).

- **Ilene Burnstein:** La necesidad de productos de software de alta calidad ha obligado a identificar y cuantificar factores de calidad como: capacidad de uso, capacidad de prueba, capacidad de mantenimiento, capacidad de ser medible, capacidad de ser confiable y a desarrollar practicas de ingeniería que contribuyen a la obtención de productos de alta calidad (7).

Relacionado con lo analizado en este epígrafe, los autores de la investigación definen que las pruebas de software constituyen: un proceso que se realiza con la intención específica de encontrar errores previos a la entrega al usuario final, haciendo énfasis especial en los aspectos a tener en cuenta para obtener un efectivo proceso de pruebas; para lograr esta efectividad se hace necesario tener conocimiento de los objetivos de las mismas.

1.1.1 Objetivos de las pruebas

Se considera que la prueba de software es una etapa imprescindible durante todo el proceso de desarrollo, pues una vez que se genera código fuente, el software debe ser probado para descubrir y corregir el máximo de errores posibles antes de su entrega al cliente. Según Pressman el objetivo de la prueba es: *“diseñar pruebas que saquen a la luz diferentes clases de errores con la menor cantidad de tiempo y espacio.” (1)*

Bill Hetzel (8), en “The Complete Guide To Software Testing”, planteó que los objetivos específicos de las pruebas pueden ser:

- Planificar las pruebas necesarias en cada iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba que especifican cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan errores son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los defectos importantes puedan ser arreglados.

Al término del análisis de los objetivos de las pruebas se infiere que el objetivo general de las pruebas es encontrar errores y defectos que atenten contra la calidad final del producto en construcción o en funcionamiento. El cumplimiento de estos objetivos es de gran importancia ya que este proceso posibilita no solo la detección de errores en el código sino la documentación necesaria que facilita que este código pueda ser reutilizado. Una vez conocidos los objetivos de las pruebas, se deben seguir una serie de principios para que el proceso tenga la calidad esperada, el análisis de estos principios se realiza a continuación.

1.1.2 Principios de las pruebas

Para lograr una aplicación adecuada de los diferentes métodos de pruebas que existen se necesita que los ingenieros que trabajan en esta línea conozcan cuales son los principios que los ayudarán y guiarán en la realización del proceso de prueba que se va a efectuar.

Para Roger Pressman los principios básicos que guían el desarrollo de las pruebas de software son (1):

- La prueba puede ser usada para mostrar la presencia de errores, pero nunca de su ausencia.
- La principal dificultad del proceso de prueba es decidir cuándo parar.
- Evitar casos de pruebas no planificados, no reusables y triviales a menos que el programa sea verdaderamente sencillo.
- Una parte necesaria de un caso de prueba es la definición del resultado esperado.
- Los casos de pruebas tienen que ser escritos no solo para condiciones de entrada válidas y esperadas sino también para condiciones no válidas e inesperadas.
- Los casos de pruebas tienen que ser escritos para generar las condiciones de salida deseadas.

- El número de errores sin descubrir es directamente proporcional al número de errores descubiertos.
- Las pruebas deberían empezar por “lo pequeño” y progresar hacia “lo grande“.
- Con la excepción de las pruebas de unidad e integración, un programa deberá ser probado por la persona u organización que lo desarrolló.
- Asigna el programador más creativo a la prueba.

Otros de los principios de las pruebas fueron definidos por Davis (9) en “201 Principles of Software Development”, algunos de ellos se enumeran en los párrafos sucesivos a este comentario:

- ***A todas las pruebas se le debería poder hacer un seguimiento hasta los requisitos del cliente.*** Como se ha visto el principal objetivo es encontrar errores. Para el cliente los errores más graves son los que le impiden al sistema cumplir sus requisitos.

- ***Las pruebas deberían planificarse mucho antes de que empiecen.*** La planificación de las pruebas puede comenzar tan pronto como esté completo el modelo de requisitos. La definición detallada de los casos de prueba puede empezar una vez que se haya aprobado el modelo de diseño. Por tanto, se pueden planificar y diseñar todas las pruebas antes de generar ningún código.

- ***El principio de Pareto (10) es aplicable a la prueba del software.*** El principio de Pareto implica que el 80 por ciento de todos los errores descubiertos durante las pruebas surgen al hacer un seguimiento de sólo el 20 por ciento de todos los módulos del programa. El problema está en aislar estos módulos sospechosos y probarlos.

- ***Las pruebas deberían empezar por lo pequeño y progresar hacia lo más grande***. Las primeras pruebas planeadas y ejecutadas en general se centran en módulos individuales del programa y a medida que avanzan las pruebas, se concentran en encontrar errores en grupos integrados de módulos y finalmente al sistema entero.

- ***No son posibles las pruebas exhaustivas.*** Esto se plantea porque incluso en un programa pequeño la cantidad de permutaciones de caminos es muy grande, por lo que es imposible cubrir todas las combinaciones de caminos. Sin embargo es posible cubrir adecuadamente la

lógica del programa y asegurarse de que se han aplicado todas las condiciones del diseño procedimental.

- ***Para hacer más eficaces, las pruebas deberían ser realizadas por un equipo independiente.*** *Se* ha mostrado que el ingeniero de software que creó el sistema no es el más indicado para realizar las pruebas al sistema.

Luego de haber analizado los principios de las pruebas en los párrafos anteriores, se llega a la conclusión de que el principio fundamental a tener en cuenta a la hora de realizar una prueba es que ésta no puede asegurar la ausencia de errores; solo puede demostrar que existen defectos en el software. Para llevar a cabo las pruebas existen diversos enfoques que determinan qué características del software se probarán, denominados tipos de pruebas (11).

1.1.3 Tipos de pruebas

Los tipos de pruebas determinan qué características del software se probarán durante el proceso de pruebas. Cada uno en general tiene diferentes niveles de dificultad y en cierta medida los errores detectados a través de ellas están relacionados con dicha complejidad. Para relacionar los tipos de pruebas referidos a continuación los autores se han basado en las dimensiones de calidad (12):

Funcionalidad	**Funcional:** Pruebas fijando su atención en la validación de las funciones, métodos, servicios, caso de uso. **Seguridad:** Asegurar que los datos o el sistema solamente es accedido por los actores deseados. **Volumen:** Enfocada en verificar las habilidades de los programas para manejar grandes cantidades de datos, tanto como entrada, salida o residente en la Base de Datos.
Usabilidad	**Usabilidad:** Prueba enfocada a factores humanos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea, suficientes documentación de usuarios y materiales de entrenamiento.

Fiabilidad	**Integridad:** Enfocada a la valoración de la robustez (resistencia a fallos).
	Estructura: Enfocada a la valoración a la adherencia a su diseño y formación. Este tipo de prueba es hecho a las aplicaciones Web asegurando que todos los enlaces están conectados y el contenido que se desea es mostrado.
	Stress: Enfocada a evaluar cómo el sistema responde bajo condiciones anormales. (extrema sobrecarga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponible)
Rendimiento	**Benchmark:** es un tipo de prueba que compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.
	Contención: Enfocada a la validación de las habilidades del elemento a probar para manejar aceptablemente la demanda de múltiples actores sobre un mismo recurso (registro de recursos, memoria, etc)
	Carga: Usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante.
	Performance profile: Enfocadas a monitorear el tiempo en flujo de ejecución, acceso a datos, en llamada a funciones y sistema para identificar y direccional los cuellos de botellas y los procesos ineficientes.
Soportabilidad	**Configuración:** Enfocada a asegurar que el sistema funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema.
	Instalación: Enfocada a asegurar la instalación en diferentes

configuraciones de hardware y software bajo diferentes condiciones (insuficiente espacio en disco, etc)

Al finalizar el análisis de los tipos de pruebas en este epígrafe los autores concluyen que las pruebas funcionales se desarrollan para validar si el sistema cumple los requisitos establecidos. Según RUP un software se va perfeccionando mediante las iteraciones que se van realizando durante su ciclo de vida en varios niveles o escenarios de trabajos, que permiten probar el producto desde la menor unidad creada hasta que se ha finalizado la construcción. A continuación se estarán analizando cuales son los niveles de pruebas utilizados en la realización del proceso de pruebas durante el ciclo de desarrollo de un software.

1.1.4 Niveles de Pruebas

El proceso de prueba es llevado a cabo en varios niveles, cada uno se realizado en un determinado momento del ciclo de desarrollo del software. Se distinguen los siguientes niveles de pruebas:

1. Prueba de Desarrollador: Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas pruebas han sido consideradas solo para la prueba de unidad, aunque en la actualidad en algunos casos pueden ejecutar pruebas de integración. Se recomienda que estas pruebas cubran más que las pruebas de unidad. (12)

2. Prueba independiente: Es la prueba que es diseñada e implementada por alguien independiente del grupo de desarrolladores. El objetivo de estas pruebas es proporcionar una perspectiva diferente y en un ambiente más rico que los desarrolladores. Una vista de la prueba independiente es la prueba independiente de los stakeholder, que son pruebas basadas en las necesidades y preocupaciones de los stakeholders. (12)

3. Prueba de unidad: Es la prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca (tipo de prueba que comprueba los caminos lógicos del software proponiendo casos de prueba que se ejerciten conjuntos específicos de condiciones y/o bucles.). Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del

componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente. (12)

4. Prueba de integración: Es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Es el proceso de combinar y probar múltiples componentes juntos. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Se llama integración incremental cuando el programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y corregir. (12)

5. Prueba de Sistema: Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos de comportamiento de caso de uso son implementados. (12)

Entre los tipos de pruebas que se pueden encontrar dentro del nivel de Sistema se encuentran los siguientes:

- **Pruebas funcionales:** Tipo de prueba que evalúa la funcionalidad del sistema, centradas en asegurar que éste satisface los requisitos funcionales establecidos por el cliente.
- **Prueba de Recuperación:** Es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente.
- **Prueba de Seguridad:** Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de acceso impropios.
- **Prueba de Resistencia:** Están diseñadas para enfrentar a los programas con situaciones anormales.
- **Prueba de Rendimiento:** Está diseñada para probar el rendimiento del software en tiempo de

ejecución dentro del contexto de un sistema integrado. (12)

6. Prueba de aceptación: Prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido. (12)

Los autores de la investigación seleccionan como nivel de pruebas a utilizar en las pruebas automatizadas el sistema, por tener incluidas las pruebas funcionales, aquellas que validan que la funcionalidad del sistema satisface los requisitos especificados por el cliente. Es necesaria la utilización de las metodologías de desarrollo de un software para guiar este proceso (13). Con estas metodologías será posible desarrollar las pruebas del sistema cuando ya se disponga de la especificación de los requisitos que el mismo debe satisfacer. Por la necesidad de ser empleadas para guiar todo proceso que se desarrolle se estará haciendo un análisis en el epígrafe siguiente de las metodologías de desarrollo existentes para seleccionar la que sea más conveniente para la elaboración del procedimiento.

1.2 Metodologías de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Estas van indicando paso a paso todas las actividades a realizar para lograr el producto deseado, indicando además, qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener cada una. Detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla. (14)

El uso de las metodologías constituye un paso significativo para la elaboración de un producto de software. Actualmente podemos encontrar diversas metodologías con diferentes enfoques: metodologías tradicionales como RUP (Rational Unified Process) y MSF (Microsoft Solution Framework) y metodologías ágiles como XP (Extreme Programing), SCRUM, Crystal, DSDM (Dynamic Systems Development Method), ASD (Adaptive Software Development), FDD (Feature-Driven Development), y LD (Lean Development). (15)

RUP no clasifica dentro de las metodologías ágiles por que se generan un gran número de artefactos, pero se recomienda para proyectos extensos que tienen la mayoría de sus funcionalidades críticas y

que necesitan ser divididas en sub-módulos para un mejor rendimiento del sistema; ya que contiene una forma muy disciplinada de asignar tareas y responsabilidades, llevar a cabo un desarrollo iterativo, administrar los requisitos y muy especial verificar la calidad de software. A continuación se realiza un estudio del tratamiento particular que le da la metodología RUP a las pruebas de software.

1.2.1 RUP. Proceso Unificado

RUP es la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Tiene tres características fundamentales: Define el proceso como iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura. RUP ha agrupado las actividades en grupos lógicos definiendo 9 flujos de trabajo principales y en su ciclo de vida cuenta con cuatro fases: inicio, elaboración, construcción y transición, que guían cada uno de los flujos, como se muestra en el Anexo 1.

“La etapa de prueba es un flujo de trabajo fundamental cuyo propósito general es comprobar el resultado de la implementación mediante las pruebas de cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema que van a ser entregadas a terceras partes” (15). Es aconsejable que las pruebas se realicen desde el momento en que comienza el desarrollo y continúen hasta que finalice el mismo. Para que se realice el proceso de pruebas la metodología RUP propone diferentes roles que serán analizados a continuación.

1.2.2 Roles de pruebas en RUP

Un rol se define como el comportamiento y las responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. Lo roles propuestos por RUP para llevar a cabo el desarrollo de las pruebas son (15):

1. Hay un grupo que se responsabiliza de los componentes de pruebas que automatizan algunos de los procedimientos de prueba denominados **Ingenieros de componentes**.

2. Se encuentra también el **Ingeniero de pruebas de integración** que se encarga de realizar las pruebas de integración que se necesitan para cada construcción producida en el flujo de trabajo de la implementación, documentar los defectos en los resultados de las pruebas de integración y prueba el resultado creado por el Ingeniero de sistemas en el flujo de trabajo de la implementación.

3. Están los que se responsabilizan de realizar las pruebas de sistema necesarias sobre una construcción que muestra el resultado (ejecutable) de una iteración completa, los **Ingenieros de pruebas de sistema**, encargados además de documentar los defectos en los resultados de las pruebas de sistema. Estos necesitan saber mucho sobre el funcionamiento interno del sistema.

4. Por último, un grupo que reporta mucho interés para los autores de la investigación, ya que será el rol desempeñado por ellos en la propuesta de solución; los **Diseñadores de pruebas**, los cuales estarán encargados de planear las pruebas lo que significa que deciden los objetivos de pruebas apropiados, planificar las pruebas, seleccionar y describir los casos de pruebas y los procedimientos de pruebas correspondiente que se necesitan, y responsables de la evaluación de las pruebas de integración y de sistema cuando estas se ejecutan.

Para obtener un resultado eficiente en la ejecución de las pruebas funcionales se necesita que estas sean efectuadas de una forma organizada. Esta organización puede ser llevada a cabo con la confección de un plan de pruebas. Al llevar a cabo el flujo de pruebas de RUP se producen, modifican y usan un conjunto de productos tangibles del proyecto entre los que se pueden mencionar el Plan de Pruebas, en el que se explica el alcance, requisitos a probar, estrategia y los recursos requeridos, calendario, herramientas así como los responsables involucrados en el proceso de pruebas así como los Casos de Pruebas utilizados para comprobar que el producto que se encuentra en desarrollo, satisface realmente las peticiones del usuario final, tal y como se describió en la especificación de los requerimientos y los casos de uso.

1.3 Plan de pruebas

Para comprobar el correcto funcionamiento de un producto se hace imprescindible realizar un plan de pruebas, con el cual se procederá a ejecutar una serie de pruebas que nos permitan obtener resultados correctos y erróneos con el fin de garantizar que un producto cumpla con los requerimientos planteados por el cliente (16). Este conjunto de pruebas nos hace capaces de determinar si nuestro programa es erróneo sobre todo en casos extremos y particulares, tanto si estos fallos se producen por la mala implementación del programa o bien por un uso especifico que realiza el usuario.

Plan de prueba: Es un documento de alto nivel que define el proyecto de prueba del software, para poder medirlo y controlarlo fielmente; la estrategia de la prueba y los elementos organizados del ciclo

vital de la prueba, incluyendo requisitos de recurso y proyecciones de horario (15). La precisión con que se definió plan de pruebas en este párrafo, está dada por la importancia que tiene este para una organización, pues para efectuar pruebas al software es ineludible que estas se encuentren documentadas con lo que se permite medir y controlar mejor los resultados.

El plan de pruebas que se utiliza en el CEDIN se compone fundamentalmente por: las Especificaciones del software y hardware, una Descripción de los requisitos y los Casos de uso a analizar, Estrategia de pruebas que se pretende aplicar, Técnica de pruebas a utilizar, El ambiente para el desarrollo de las pruebas , como está compuesto el Proceso, Casos de prueba diseñados, Recursos humanos materiales y/o financieros con que se disponen así como el Calendario y plazos definidos para terminar el proceso.

1.3.1 Estrategias de Prueba

Una estrategia de prueba integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software (1). La estrategia proporciona un mapa que describe los pasos que hay que llevar a cabo como parte de la prueba, cuándo se deben planificar y realizar esos pasos, y cuanto esfuerzo, tiempo y recursos se van a requerir. La estrategia que se ha de seguir a la hora de evaluar dinámicamente un sistema software debe permitir comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Por lo tanto cualquier estrategia de pruebas debe incorporar la planificación de la prueba, el diseño de los casos de prueba, la ejecución de las pruebas y la agrupación y evaluación de los datos resultantes.

Se puede concluir que en un proyecto las pruebas requieren mayor esfuerzo que cualquier otro flujo de trabajo de la Ingeniería de Software, si se efectúa sin un plan estratégico, el tiempo es desaprovechado y el esfuerzo es consumido innecesariamente y, en el peor de los casos, los errores inadvertidos quedarán sin detectar, por tanto, puede ser muy conveniente establecer una estrategia sistemática para probar el software.

1.3.2 Casos de pruebas

Los casos de prueba constituyen una de las herramientas principales en las pruebas de software, ya que mediante ellos, se formalizan las pruebas, describiendo todo lo que puede implicar llevarlas a cabo

y sugiriendo los resultados que ésta debería arrojar. Los casos de prueba (CP) y su reutilización juegan un papel fundamental a la hora de hacer que un proceso de pruebas sea eficiente y cumpla con los tiempos que el mercado demanda. (11). Los CP definen un conjunto específico de entradas de pruebas, ejecución de condiciones y resultados esperados (16).

Un CP es como un reemplazo de un actor del sistema, pues simula las interacciones del actor con el sistema para verificar que este hace lo que se espera de él. Así, el principal artefacto que se necesita analizar para obtener pruebas del sistema son los requisitos funcionales.

Los autores de la investigación llegan a la conclusión de que un CP es una herramienta utilizada para formalizar las pruebas, describiendo como deben ser desarrolladas y sugiriendo que resultados se pueden obtener. Como se definió en el epígrafe Tipo de pruebas, para el proceso a automatizar, se aplicarán las pruebas funcionales también conocidas como pruebas de comportamiento o de caja negra, por lo que a continuación se explica en qué consisten estas pruebas y cuáles son las técnicas que se emplearán.

1.4 Pruebas de Caja Negra

Las pruebas de Caja Negra se centran fundamentalmente en los requisitos funcionales, permitiendo al ingeniero del software derivar conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa.

Las pruebas de caja negra, también denominadas pruebas funcionales o de comportamiento, se centran en los requisitos funcionales del software. O sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (1). Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro. (16)

Estas pruebas se llevan a cabo sobre la interfaz del software y son completamente indiferentes al comportamiento interno y la estructura del programa. Los casos de prueba de Caja Negra pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada, que se produce una salida correcta, y que se mantiene la integridad de la información externa. (17). Al

diseñar una serie de casos de pruebas que tengan una gran probabilidad de encontrar errores se cumple el objetivo fundamental. Para desempeñar el mismo se llevan a cabo técnicas de pruebas del software, las que facilitan una guía sistemática para diseñar pruebas que comprueben la lógica interna de los componentes software y el grado de cumplimiento de los requisitos funcionales. (11)

1.4.1 Métodos de Pruebas de Caja Negra

Mediante las técnicas de prueba de caja negra se obtienen un conjunto de casos de prueba que satisfacen los siguientes criterios (9):

1. CP que reducen, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales que se deben diseñar para alcanzar una prueba razonable.

2. CP que nos dicen algo sobre la presencia o ausencia de clases de errores en lugar de errores asociados solamente con la prueba que estamos realizando.

El empleo de los diferentes métodos que se utilizan en la aplicación de las pruebas de Caja Negra permite dado una serie de valores de entrada, obtener un conjunto de valores de salida y además posibilita realizar una comparación entre varias versiones con los mismos datos de entrada, para poder verificar que las salidas sean las mismas. Los métodos existentes son los siguientes: (14)

1. Particiones de equivalencia
2. Análisis de valores límites
3. Prueba de comparación
4. Tabla Ortogonal
5. Grafo Causa-Efecto

De las técnicas anteriormente mencionadas, se seleccionan para el diseño de los casos de pruebas (DCP) las técnicas de Partición de equivalencia y Análisis de valores límites; pues la primera permite simplificar el número de casos de prueba, obteniendo valores válidos e inválidos de las entradas (14), teniendo como ventaja que cada vez que un usuario usa el programa está a la vez realizando una prueba al mismo, mientras que la segunda complementa a la de Partición equivalente porque en lugar de realizar la prueba con cualquier elemento de la partición equivalente, se escogen los valores en los

bordes de la clase. Por lo antes expuesto, es que los autores de este trabajo se disponen a realizar un breve análisis de los métodos seleccionados.

Particiones de equivalencia: Esta técnica, conocida también como Partición equivalente, utiliza las clases de equivalencia para el diseño de los casos de prueba, es decir, la entrada de una serie de datos válidos y no válidos, cubriendo en cada caso el máximo número de entradas. Define CP que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Una condición de entrada puede ser un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. (1)

Reglas para la identificación de las clases de equivalencia:

- Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.
- Si una condición de entrada es lógica, se define una clase de equivalencia valida y una no válida. (1)

Análisis de valores límites: Esta técnica de prueba complementa a la partición equivalente. En lugar de centrarse solamente en las condiciones de entrada, este obtiene también CP para el campo de salida. (1)

Reglas para el análisis de valores límites:

- Si una condición de entrada específica un rango delimitado por los valores a y b, se deben diseñar casos de prueba para los valores a y b, y para los valores justo por debajo y justo por encima de a y b, respectivamente.

- Si una condición de entrada específica un número de valores, se deben desarrollar casos de prueba que ejerciten los valores máximos y mínimos, uno más el máximo y uno menos el mínimo.

- Aplicar las directrices anteriores a las condiciones de salida.

- Si las estructuras de datos internas tienen límites preestablecidos hay que asegurarse de diseñar un CP que ejercite la estructura de datos en sus límites. (1)

Para obtener un buen proceso de pruebas se hace necesario disponer de un procedimiento en el que se encuentren detallados los pasos que se deben realizar durante el proceso, por la importancia que merece el mismo el epígrafe siguiente estará destinado a realizar un análisis del concepto de procedimiento de pruebas.

1.5 Procedimiento de pruebas

Un procedimiento no es más que la descripción de un conjunto de pasos que se realizan sucesivamente, es decir, la especificación de las actividades que se deben seguir para realizar algún proceso (1). Un procedimiento de pruebas es una especificación de cómo llevar a cabo la preparación, ejecución y evaluación de los resultados de un CP en particular. El procedimiento de pruebas es similar a la descripción del flujo de eventos del caso de uso (CU), aunque incluye información adicional, como los valores de entrada del CU, la forma en la que estos valores son introducidos en la interfaz del usuario y lo que hay que verificar. (1)

Un proceso define quién está haciendo qué, cuándo y cómo alcanzar determinado objetivo (1). Para los ingenieros que se dedican a la construcción de software este objetivo se traduce en construir un producto o mejorar uno existente. (1).

Según Jacobson, Booch y Rumbaugh en su libro Proceso Unificado de Desarrollo del Software un proceso puede describirse en partes llamadas flujos de trabajo. Ejemplo de ello lo es el flujo de trabajo de pruebas. Un proceso de pruebas consta generalmente de cuatro fases: la fase de diseño de pruebas, la fase de codificación, la fase de ejecución y la fase de análisis de los resultados (13). Los procesos de pruebas y en general todos aquellos que apunten a asegurar la calidad del producto, permiten obtener elementos para lograr la madurez del modelo de calidad de las compañías, mediante

actividades y procesos de retroalimentación. Es importante tener presente, que los procesos de pruebas no componen totalmente el área de proceso de aseguramiento de calidad; es así como las pruebas proveen los elementos y prácticas de mejora con el fin de retroalimentar los procesos. (14)

1.6 Estado actual de las pruebas de software

Las tendencias pedagógicas actuales que se ponen de manifiesto en la enseñanza de pruebas de software en el mundo son los cursos optativos de pregrado, cursos de postgrado o mediante asignaturas incluidas en el plan de estudios de diferentes instituciones o universidades en sus carreras vinculadas a Ciencias Informáticas. Estos cursos o asignaturas pueden ser impartidos de forma presencial, utilizando el método convencional de enseñanza u online con la ayuda de Internet o Entornos Virtuales de Aprendizaje (EVA). Universidades como la Biblioteca Nacional de Maestros en Argentina (18) y la Universidad Distrital Francisco José de Caldas (19) ofrecen cursos online sobre pruebas de software.

Dentro de esta esfera de la producción de software se enmarca la Universidad, la cual ha tenido como prioridad mejorar el proceso de prueba, para lograr esto se crea el Centro para la Excelencia en el desarrollo de Proyectos Tecnológicos (CALISOFT), y dentro de este un laboratorio de pruebas, creando de esta forma un entorno adecuado para esta actividad. Dentro de los lineamientos propuestos por el CALISOFT se le dedica un espacio a las pruebas de software, quedando recogida la necesidad de definir roles que generalmente se obvian, como es el caso del ingeniero de componentes, diseñador de pruebas y probador.

En la UCI la enseñanza de pruebas está incluida dentro de la asignatura Ingeniería de Software 2, específicamente en el Flujo de Trabajo Pruebas de RUP. La teoría referente a pruebas de software es recibida por los estudiantes en sus aulas para luego aplicar esos conocimientos a los proyectos que se orientan a medida que el curso va avanzando. Toda la bibliografía básica de la asignatura se encuentra disponible en el EVA de la universidad, a la que acceden los estudiantes a través de la red para utilizar los materiales en el estudio independiente de la asignatura y realizar pruebas online con fines auto evaluativos o exámenes con criterios de evaluación para los profesores que imparten dicha asignatura. Estos son sólo algunos de los servicios que ofrece este entorno pues existen otros como los foros que tienen el objetivo de aumentar la comunicación y la preparación del personal que los utiliza.

Hoy en día han surgido tendencias en virtud de demostrar la importancia de las pruebas de software, así como proponer mejoras a las mismas y de esta forma, incrementar la calidad del proceso de desarrollo. Entre estas tendencias se encuentran vincular el proceso de pruebas a lo largo del ciclo de vida de desarrollo de software y la automatización de las mismas. (14)

1.6.1 Pruebas automatizadas

La automatización de las pruebas es la parte del proceso de desarrollo, en la que software de automatización es utilizado para controlar la ejecución de pruebas, comparación de los resultados, la preparación de precondiciones y la realización de informes (20).

Para automatizar las pruebas de software es necesario usar estrategias, herramientas y artefactos que aumentan o reducen la necesidad de interacción manual o humana en tareas inexpertas, repetidoras o redundantes (20). Con el mismo se está automatizando el proceso de prueba manual que tradicionalmente ha sido utilizado. Para esto se requiere que una “etapa de prueba manual formalizada”, exista en la compañía o la organización.

En general, las pruebas automatizadas son más repetibles que las manuales. Procuran ser exactamente igual todo el tiempo y no olvidan ningún detalle. Consumen menos esfuerzo para ser ejecutadas que las pruebas manuales, por lo que es posible que sean realizadas más a menudo.

Algunos de los objetivos de las pruebas automatizadas son los siguientes (20):

- **Las pruebas nos ayudan a mejorar la calidad:** La razón principal de hacer pruebas es garantizar que las cosas estén funcionando según lo solicitado, asegurando así la calidad.
- **Las pruebas deben ayudarnos a comprender el sistema:** Bloquear los errores y sus consecuencias no es la única cosa que las pruebas pueden hacer por nosotros. Ellas pueden brindarnos una noción del sistema con solo leer lo que el código debería hacer. Los componentes de las pruebas son en realidad una descripción de los requisitos de los componentes de software.
- **Las pruebas deben reducir (y no introducir) riesgo:** Como se mencionó antes, las pruebas deberían aumentar la calidad de nuestro software ayudándonos a obtener una documentación

mejor de los requisitos y prevenir que los errores sean insertados durante el desarrollo incremental.

- **Las pruebas deben ser fáciles de ejecutar:** La mayoría de los desarrolladores sólo desea escribir código, la tarea de probar la funcionalidad es una carga para ellos. Las pruebas automatizadas proporcionan una red de seguridad a fin de que podamos escribir código con mayor rapidez, y ejecutemos las pruebas con frecuencia si ellas son realmente fáciles de ejecutar.

- **Las pruebas deben ser fáciles de ejecutar y mantener:** Codificar es una actividad difícil porque tenemos que mantener una serie de informaciones en nuestras cabezas. Y cuando escribimos pruebas debemos centrarnos en probar y no codificar las pruebas. Esto significa que las pruebas deben ser simples, fáciles de leer y escribir.

- **Las pruebas deben exigir un mantenimiento mínimo, así como el sistema probado:** El cambio es algo normal en nuestras vidas. Deberíamos escribir para hacer las pruebas automatizadas para hacer los cambios en el código más fácil, por lo que debemos tener cuidado para que nuestras pruebas no hagan más difícil los cambios en nuestros códigos.

Luego de analizar los beneficios que brindan las pruebas automatizadas, se puede concluir que estas ofrecen ventajas sobre las pruebas manuales, estas son (20):

- **Confiable:** Las pruebas realizan exactamente las mismas operaciones cada vez que se ejecutan, eliminando el error humano.

- **Repetible:** Se prueba cómo el software reacciona bajo ejecución repetida de las mismas operaciones.

- **Programable:** Se programan las pruebas sofisticadas que ponen en evidencia la información ocultada del uso.

- **Comprensivo:** Se construye una habitación de pruebas que cubra cada característica en tu uso.

- **Reutilizable:** Se reutilizan las pruebas en diversas versiones de uso, aunque los cambios afecten la interfaz utilizada.
- **Software de una calidad mejor:** Se realizan más pruebas en menos tiempo y con menos recursos.

Para alcanzar las ventajas citadas en el párrafo anterior se hace necesario la existencia de un proceso de prueba manual formalizado en la compañía u organización. El uso de las herramientas adecuadas reduce la necesidad de la interacción manual o humana en el proceso de pruebas que se puede convertir en tareas inexpertas, repetidoras o redundantes. El proceso de selección de esta herramienta requiere la realización de un estudio comparativo en el que se analicen siguiendo varios criterios de selección cuales son las ventajas y desventajas que ofrecen cada una de las herramientas candidatas.

1.6.2 Herramientas de automatización

En la actualidad las herramientas de automatización han posibilitado que la intervención del ser humano en muchos procesos sea sustituida por el software automatizado; el flujo de prueba específicamente es muy importante en el desarrollo de un software, por lo que se hizo necesario realizar un estudio de las herramientas existentes, que permitirá seleccionar la que más beneficios aportará a la realización de forma automatizada de las pruebas funcionales a las aplicaciones web desarrolladas en el centro. Las herramientas candidatas a utilizar en la automatización de las pruebas funcionales a las aplicaciones web del centro, se relacionan a continuación:

WatiN (Web Application Testing) (Anexo 2): es un programa fácil de usar, con multitud de funciones para probar aplicaciones Web. WatiN se desarrolla en C # y su objetivo es ofrecerte una manera fácil de automatizar sus pruebas con Internet Explorer y Firefox utilizando .Net. WatiN se comunica con el objeto de Internet Explorer® para iniciar el explorador e interactuar con éste de la misma manera que lo haría el usuario. WatiN puede escribir texto en campos, seleccionar botones de opción, hacer clic en casillas y botones, y leer valores de sus formularios y páginas. El código para su prueba se muestra en el cuadro situado en la parte inferior de la aplicación. Después puede copiar este código o guardarlo directamente del grabador, para copiar o guardar la prueba, necesitará agregar las aserciones. WatiN identifica los elementos por su ID, nombre, clase, texto, url, valor, estilo... en función de su código. (21)

Selenium (Anexo 3): es una herramienta de prueba funcional, que proporciona una plataforma simple para verificar la funcionalidad de la aplicación. Se ejecuta en muchos navegadores y sistemas operativos, y puede ser controlado por muchos lenguajes de programación y frameworks de pruebas. Con las pruebas, Selenium es capaz de probar las aplicaciones web desde la perspectiva del usuario, no desde el punto de vista de código y permitirá tener tantas pruebas diferentes como deseen para un mismo sistema y ejecutarlas una por vez automáticamente. Selenium provee unos Apis en diferentes lenguajes (PHP, Ruby, JAVA, Javascript, etc) que le permitan indicar mediante comandos que pruebas debe hacer. Posee un IDE (aún en versión Beta) que automatiza aun más la tarea, es sencillo y ayuda a aprender los comandos más rápidamente, no es tan flexible como los Apis pero de todos modos es una herramienta muy potente para la automatización de pruebas funcionales. Cada comando es del estilo “Haz clic aquí”, “Espera a que se recargue la página”, “Completa tal campo de formulario”, “Haz Clic en el Botón enviar”, “Verifica el resultado”, etc. (22)

Watir (Web Application Testing en Ruby) (Anexo 4): es una herramienta de pruebas funcionales para aplicaciones Web. Herramienta muy potente para la automatización de pruebas funcionales a una aplicación web por ser compatible con varios navegadores en diferentes plataformas. Es compatible con las pruebas ejecutadas en la capa de navegador web por la conducción de un navegador web e interactuar con objetos en una página Web. Utiliza el lenguaje de programación Ruby. Es de código abierto para la automatización de la colección de los navegadores Web. Te permite escribir las pruebas que sean fáciles de leer y mantener. Es simple y flexible. Se hace clic en enlaces, rellena formularios, botones de prensas. Watir también comprueba los resultados, como si las previsiones de texto aparecen en la página. Puede instalar por separado en Windows, Mac o Linux. (23)

La comparación de las herramientas de automatización seleccionadas para identificar cual se utilizará en las pruebas funcionales a las aplicaciones web del centro, está regida por 6 criterios relacionados con funcionalidades específicas de cada una, la definición de estos criterios es brindada a continuación:

- **Tipo de licencia y precio:** el tipo de licencia delimita la forma de uso y está directamente relacionado con el precio, ya que dependiendo de las capacidades o límites del aplicativo los precios suben o bajan según sea el caso. Generalmente en éste tipo de aplicaciones el precio lo delimita el número máximo de usuarios concurrentes que es posible emular.

- **Acceso al código fuente:** posibilidad de acceder al código fuente de la herramienta y realizar modificaciones en ella. Generalmente esto depende del tipo de licencia que cada una de ellas tenga.
- **Lenguajes de programación utilizados:** el lenguaje de programación usado para la construcción de la herramienta.
- **Plataforma:** sistemas soportados para la instalación de la herramienta que en términos precisos se traduce en la portabilidad entre sistemas, tal caso son las herramientas programadas en java, las cuales generalmente son portables entre sistemas Windows, Linux, UNIX, Solaris, entre otros.
- **Visualización en tiempo real:** función que permite visualizar los resultados de las pruebas, las gráficas y las tablas de datos paralelamente la prueba se está ejecutando.
- **Pruebas funcionales:** en herramientas muy especializadas, permiten inicialmente realizar pruebas funcionales basadas en requisitos, generalmente programables por medio de script.

Los requisitos que debe cumplir la herramienta que los autores seleccionarán para las pruebas funcionales automatizadas al sistema en cuestión son: tener un código abierto a modificaciones necesarias en la automatización de las pruebas funcionales, los sistemas operativos soportados para la instalación de la herramienta deben ser Windows o Linux, contralada por los lenguaje de programación Java, C# o C++, el tipo de licencia que delimite la forma de uso debe ser libre y no debe haber costo alguno para utilizar la herramienta, tener incluidas la mayor cantidad de características que permitan verificar la funcionalidad de la aplicación a prueba.

La comparación de las herramientas para identificar las que serán utilizadas en las pruebas funcionales a las aplicaciones web desarrolladas en el centro se detalla a continuación:

No	Nombre	WatiN	Selenium	Watir
	Tipo de	Licenciado bajo la	Licenciado bajo la	No hay costo alguno

1.	**licencia y precio.**	licencia Apache 2.0.	licencia Apache 2.0.	para utilizar la herramienta.
2.	**Acceso al código Fuente.**	No	Si	Si
3	**Lenguajes de programación utilizados**	C#,	C#, Java, Perl, PHP	Rubí
4.	**Plataforma**	Windows 2000/XP/2003/ Vista	Windows 2000/XP/2003/ Vista, Linux, Solaris y Mac OS	Windows 2000/XP/2003/ Vista, 7 y Linux
5.	**Visualización en tiempo Real**	No	Si	No

6.	**Pruebas Funcionales**	"Texto en campos", "Seleccionar botones de opción", "Hacer clic en casillas y botones", "Leer valores de sus formularios y páginas"	"Haz click aquí", "Espera a que se recargue la página", "Completa tal campo de formulario", "Haz Click en el Botón enviar", "Verifica el resultado"	"Envío de formulario", "Hacer clic enlaces y botones en una página", "Comprobación del título y texto en una página"

Tabla 1.1: Comparación de las herramientas candidatas

Al finalizar el estudio comparativo de las aplicaciones existentes en el mundo que apoyan la realización de las pruebas funcionales en aplicaciones web se obtuvieron los siguientes resultados:

1. Selenium y Watir tienen un código abierto a modificaciones necesarias en la automatización de las pruebas funcionales.
2. WatiN y Selenium son controladas por los lenguajes de programación C# (WatiN), Java, Perl y PHP.
3. Selenium y Watir soportan en su instalación los sistemas operativos Windows 2000/XP/2003/ Vista, 7, Linux (Watir), Solaris y Mac OS.
4. WatiN y Selenium fueron licenciado bajo la licencia Apache 2.0.
5. Selenium tiene incluida la mayor cantidad de características que permitan verificar la funcionalidad de la aplicación a prueba.

Los autores de la investigación definen de que la herramienta que cumple con todos los requisitos establecidos para ser seleccionada para las pruebas funcionales automatizadas a las aplicaciones web del centro es Selenium ya que: tiene un código abierto a modificaciones necesarias, es controlada por

los lenguajes de programación C#, Java, Perl y PHP, soporta en su instalación los sistemas operativos Windows 2000/XP/2003/ Vista, 7, Linux, Solaris y Mac OS, no hay que pagar derechos para ser utilizada por estar registrada bajo la licencia Apache 2.0 y tiene incluida las características siguientes, “Haz clic aquí”, “Espera a que se recargue la página”, “Completa tal campo de formulario”, “Haz Clic en el Botón enviar”, “Verifica el resultado”, que permitan verificar la funcionalidad de la aplicación a prueba. La información detallada de la comparación de las herramientas para identificar las que serán utilizadas en las pruebas funcionales automatizadas a las aplicaciones web del centro puede ser consultada en el documento resultado de esta investigación: “Informe de selección de herramientas de automatización de pruebas funcionales”.

1.7 Conclusiones del Capítulo

En este capítulo se definieron los diferentes niveles y tipos de pruebas, así como la metodología a emplear en la realización del procedimiento de pruebas, para automatizar las pruebas funcionales a las aplicaciones web desarrolladas en el centro. Fueron comparadas las herramientas de pruebas funcionales para aplicaciones web y seleccionada aquella que cumple con los requisitos especificados para la realización de las pruebas funcionales automatizadas en aplicaciones Web.

Capítulo 2: Descripción del procedimiento para pruebas funcionales automatizadas.

El presente capítulo está dirigido a describir los pasos para realizar de forma automatizada las pruebas funcionales a las aplicaciones web desarrolladas en el CEDIN, utilizando la herramienta Selenium. Se definen el proceso para incorporar los casos de pruebas a la herramienta seleccionada, la ejecución de las pruebas y documentación de los resultados, con el fin de obtener como resultado una disminución sustanciosa del trabajo de los probadores.

2.1 Requerimientos necesarios para realizar las pruebas

Los requisitos para aplicar las pruebas funcionales automatizadas al producto seleccionado, constituyen las características más importantes de hardware y software que intervienen en el diseño y aplicación de las pruebas, posibilitando un correcto funcionamiento de la aplicación. Nunca se debe testear el software en un entorno de producción. Para crear un entorno de pruebas es necesario crear las mismas condiciones que en la máquina de producción. Para el desarrollo de las pruebas es necesario disponer de los recursos siguientes:

Requerimientos mínimos de software:

1. Sistema Operativo tanto Windows (win9.x o versión superior) como Linux (cualquiera de sus distribuciones).
2. El Navegador Web compatible con HTML 2.0 y CSS, podrá ser Netscape 3 (o superior), Mozilla 3.5 (o inferior), Internet Explorer 4.2 (o superior) y compatibles.

Requerimientos mínimos de hardware:

1. 256-512 MB de RAM.
2. 40 GB de Disco Duro o superior.

2.2 Metodología a utilizar

La metodología usada para la construcción de algunos de los sistemas desarrollados en el CEDIN es RUP. A continuación se exponen las razones fundamentales de su elección para guiar la elaboración

del procedimiento de automatización de las pruebas funcionales a las aplicaciones web del centro: (25)

1. Metodología estándar “de facto” a nivel mundial.
2. Alta adaptabilidad a las condiciones reales del desarrollo del sistema. Es decir que podemos hacerla más ágil según se necesite.
3. Metodología para entornos de desarrollo orientado a objetos.
4. Alta capacidad de generar documentación relevante del negocio.
5. Alta retroalimentación (feedback) con el cliente. (Iterativo e Incremental).
6. Dominio básico de la metodología por el personal del proyecto.

La utilización de RUP es factible en proyectos de gran tamaño ya que tiene un proceso de pruebas bien definido. La ventaja de ser dirigido por CU posibilita el DCP y la posible automatización de los mismos. Debido a que es iterativo e incremental, las pruebas no se realizan cuando el producto está terminado, sino que este posibilita que en cada iteración pueda ser aplicado al menos un ciclo de pruebas. RUP también propicia el uso intensivo de la documentación, siendo una excelente práctica durante el proceso de prueba que no debe abandonarse.

2.2.1 Artefactos generados

Los artefactos son piezas de información tangible del proyecto, que es creada, modificada y usada por los trabajadores al realizar actividades. Puede ser un modelo, un elemento de un modelo, o un documento (15). Se consideran importantes y necesarios para obtener el procedimiento para efectuar las pruebas funcionales automatizadas a la aplicación web seleccionada los siguientes:

- **Procedimiento de pruebas**
  Este artefacto está integrado por el conjunto de acciones que se deben realizar para utilizar una herramienta de automatización en las pruebas funcionales a las aplicaciones web del centro.

- **Plan de prueba**

  Este artefacto incluye el propósito y alcance del proceso de pruebas, qué requerimientos se van a probar, las herramientas a utilizar, los recursos a emplear, reflejando las características de hardware y software, el cronograma, así como el documento que será entregado.

- **Estrategia de prueba**

  El artefacto estrategia de prueba describe los objetivos generales de las pruebas. Incluye las fases de prueba que se deben seguir y los tipos de pruebas que se deben realizar.

- **Diseño de casos de pruebas**

  Los CP constituyen la especificación formal donde quedan registrados los datos de entrada de la prueba, las condiciones para su ejecución, así como los resultados previstos.

- **Scripts de Prueba**

  Al realizar las pruebas con Selenium se obtienen un fichero tal y como se hace en cualquier otra aplicación, y al compilarlas con la Suite de Pruebas, se genera un archivo cuyo objetivo es ejecutar las pruebas todas las veces que desee y casi siempre responde al nombre de suite test.

- **Resultados de las pruebas**

  En este artefacto se expone un resumen de los resultados de las pruebas aplicadas a los componentes operacionales del sistema, lo cual permite evaluar la funcionalidad del producto que se está desarrollando.

2.3 Realización de pruebas funcionales automatizadas

Un procedimiento de pruebas es una especificación de cómo llevar a cabo la preparación, ejecución y evaluación de los resultados de las pruebas a un sistema en particular (15). El procedimiento a desarrollar está integrado por el conjunto de acciones que se deben realizar al incluir la herramienta de automatización de pruebas seleccionadas en las pruebas funcionales a las aplicaciones web del centro, tomando en consideración cuánto esfuerzo y recursos se van a requerir, con el fin de obtener como resultado una disminución sustanciosa del trabajo de los probadores:

- **Planificación de las pruebas.**
  - Analizar los documentos entregados por el equipo de desarrollo para tener conocimientos plenos del sistema que se estará probando.
  - Elaborar un plan de pruebas para definir el alcance de la investigación.
  - Identificar las herramientas para el desarrollo de pruebas automatizadas.
  - Seleccionar herramienta para las pruebas funcionales automatizadas a los sistemas web del CEDIN.

- **Diseño de las pruebas.**
  - Diseño de los casos de prueba.
  - Incorporar los casos de prueba a la herramienta.
  - Desarrollar un manual de uso de la herramienta.

- **Ejecución de las pruebas.**
  - Ejecutar las pruebas planificadas.
  - Registrar los resultados de la ejecución de las pruebas.
  - Elaborar “Informe de resultado de las pruebas”.

- **Análisis de los resultados de las pruebas.**
  - Comparar los resultados con los obtenidos en las pruebas manuales efectuadas al producto.
  - Determinar si los resultados de las pruebas automatizadas son más efectivos que los alcanzados en las pruebas manuales.
  - Arribar a conclusiones precisas relacionadas con los resultados de la investigación.

Entre los objetivos que se persiguen para el desarrollo de las pruebas funcionales automatizadas al sistema seleccionado, se encuentran:

- Realizar pruebas al sistema de una forma más eficiente a la tradicional, posibilitando encontrar errores de mayor magnitud en el menor tiempo posible.
- Realizar exactamente las mismas operaciones cada vez que se ejecutan las pruebas.

- Probar cómo el software reacciona bajo ejecución repetida de las mismas operaciones.
- Realizar más pruebas en menos tiempo y con menos recursos que identifiquen errores de mayor magnitud.

Para alcanzar los objetivos planteados se debe comenzar con una planificación de cómo se automatizaría el proceso de pruebas, la que debe quedar reflejada en un Plan de Pruebas elaborado para las aplicaciones web que serán probados y la selección de la metodología de desarrollo de software que se utilizará para la planificación y el diseño de las pruebas que se automatizarán. En los sub-epígrafes siguientes se describen las actividades que regirán la automatización del proceso de pruebas ya referenciado.

2.4 Planificación de las pruebas

En la etapa de planificación se definen las pruebas que se les realizarán a la aplicación y se identifican los participantes y sus roles, designando un responsable. En este proceso se debe obtener el esfuerzo y tiempo necesario para las pruebas, productos entregables, recursos requeridos, repositorios y ambientes de prueba, entre otros aspectos. Uno de los principios más importantes de las pruebas de software es que estas deben planificarse mucho antes de que comiencen a desarrollarse.

Dentro de los proyectos existe una serie de documentos que es necesario consultar antes de realizar las pruebas. A continuación se relacionan algunos documentos que deben ser consultados para la realización de las pruebas.

1. Documento de especificación de requisitos.
2. Glosario de términos.
3. Manual de usuarios del software a utilizar para la realización de las pruebas.
4. Requisitos mínimos de hardware y software para realizar las pruebas.
5. Especificación de los casos de uso (CU).

Todos los requisitos funcionales pertenecientes al sistema deben ser profundamente estudiados antes de ponerlos a prueba, de igual forma es importante tener conocimiento de las funcionalidades del sistema. Así se evitan errores que atrasarían las pruebas y podrían influir negativamente en sus

resultados. De igual forma es importante realizar el estudio de la documentación sobre el software que se vaya a utilizar para la realización de las pruebas, así como todos los detalles de lo que son las pruebas funcionales para aplicaciones Web.

La planificación de las pruebas funcionales automatizadas debe quedar reflejada en el plan de pruebas desarrollado, en el mismo se explica alcance, requerimientos a probar, estrategia y los recursos requeridos, las herramientas así como los responsables involucrados en el proceso de pruebas y constan los entregables como salidas generadas en cada una de las actividades. Para realizar el plan de pruebas, se debe emplear la plantilla propuesta por el Centro para la Excelencia en el desarrollo de Proyectos Tecnológicos. En el plan de pruebas se debe describir de forma concisa y clara como se realizarán todas las pruebas. Cómo se desarrollará ese flujo de trabajo, es decir, su planificación, roles que intervendrán en el proceso de pruebas y funcionamiento de los mismos y herramientas que serán empleadas, todos estos elementos deben quedar bien definidos.

2.4.1 Estrategia de pruebas

El proceso de ejecución de Pruebas debe ser considerado durante todo el ciclo de vida de un proyecto, para así influir significativamente en la obtención de un producto con alta calidad. Su éxito dependerá del seguimiento de una estrategia de prueba adecuada (15). La estrategia de prueba de software que se seguirá en el desarrollo de la investigación integra un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba. Antes de comenzar a realizar toda estrategia se debe partir de un diagnóstico en el que se evidencie el problema.

Luego se seleccionan los métodos de pruebas así como las técnicas a utilizar en las pruebas a las aplicaciones desarrolladas para verificar que las funcionalidades de las mismas satisfacen todas las capacidades o condiciones (requisitos funcionales) a cumplir por el sistema que fueron definidas por los clientes, usuarios y miembros del equipo de proyecto.

En la búsqueda de un desarrollo más eficiente de las pruebas funcionales a los sistemas desarrollados en el CEDIN se definirán los pasos a seguir para realizar de una mejor manera las pruebas funcionales, influyendo significativamente en la obtención de un producto con mayor calidad. Para ello es necesario conocer que: toda estrategia de prueba de software debe incluir pruebas de bajo nivel

que verifiquen que todos los pequeños segmentos de código se han implementado correctamente, así como las pruebas de alto nivel que validen las principales funciones del sistema frente a los requisitos del cliente.

Todo lo planificado anteriormente constituye un elemento fundamental en el éxito de las pruebas funcionales automatizadas al sistema seleccionado, ya que para obtener un buen resultado se requiere de una buena planificación, reflejada en el plan de pruebas del sistema elaborado, donde se definieron los pasos a seguir, cuales guías y plantillas serán utilizada así como la estrategia de control y seguimiento de las pruebas.

2.5 Diseño de las pruebas

Una vez que se ha realizado la planificación de las pruebas se pasará a la etapa del diseño de las mismas, ya que el plan de pruebas es una de las entradas más importantes para este proceso. Un buen plan de pruebas, para ser ejecutado correctamente debe contener casos de prueba que cumplan con las siguientes características:

1. Deben ser **precisos**, pues se debe describir específicamente que es lo que verificarán.
2. **Económicos**, ya que sólo deben contener los pasos necesarios para alcanzar su propósito.
3. **Repetibles**, pues deben ser consistentes y recoger todo lo necesario para que los resultados de cualquier ejecución sean los mismos.
4. **Adecuados y trazables**, ya que deben describir la situación en la que son aplicables y deben estar relacionados con los requisitos funcionales que cubren, con objeto de facilitar la identificación del fallo y permitir hacer un seguimiento de las correcciones más fácilmente.

El Centro para la Excelencia en el desarrollo de Proyectos Tecnológicos (CALISOFT) propuso una plantilla para ser tomada como estándar en todos los proyectos. El empleo de esta constituye una ventaja pues además de que está mejor estructurada, en esta se especifican cada una de las variables probadas durante todo el proceso, y los diferentes datos de prueba empleados. Además en la misma se plasman mejor las no conformidades detectadas, y esto le sirve al desarrollador para corregir todos

los defectos encontrados, facilitándole su localización. La descripción de los aspectos por los que está compuesta dicha plantilla es ofrecida a continuación:

Descripción General: En este aspecto se realiza una descripción de los aspectos a tener en cuenta a la hora de realizar el diseño de las pruebas, incidencias en el momento de su desarrollo y otros elementos relevantes.

A este Caso de Uso se le realizaron las siguientes pruebas: Se ofrece una lista de las Pruebas diseñadas para ser aplicadas al Caso de Uso, así como las funcionalidades para las que se diseñaron las mismas.

CPR 1: <Funcionalidad #1 a revisar>: Esta parte se refiere al nombre del caso de prueba diseñado para el caso de uso correspondiente.

Descripción de la Funcionalidad: Se detalla una descripción de la Funcionalidad a Probar.

Flujo Central: Está compuesto por los pasos a desarrollar para probar la Funcionalidad que se indicó.

Condiciones de Ejecución: Constituyen los prerrequisitos del Caso de Uso, imprescindibles para que se pueda ejecutar correctamente el Caso de Prueba.

Iteraciones: Se registran las clases válidas e inválidas según los datos suministrados, resultado esperado, el resultado obtenido y las observaciones necesarias.

Registro de defectos y dificultades detectados: Se registran los defectos encontrados en la aplicación según las pruebas desarrolladas, el elemento donde se encontró el error, una descripción de la no conformidad, la etapa en la que se detecto el error así como la importancia y las recomendaciones.

Para realizar el diseño de los casos de prueba funcionales se debe contar con la especificación de cómo se debe interactuar con el sistema (Especificación de caso de uso) y el sistema, que en fin es lo que se probará. El diseño de las pruebas y los resultados de las mismas, quedarán establecidos en los documentos Diseño de Casos de Pruebas elaborados para cada uno de los CU seleccionados.

Los CP deben verificar si el producto que se esté realizando satisface los requerimientos del usuario final tal y como se describe en la especificación de los requisitos y los casos de uso. Además, CP debe garantizar que el producto se comporta como se describe en las especificaciones funcionales del diseño. En fin, el diseño de un buen CP será el eslabón fundamental del proceso de prueba, pues mediante los mismos se encontrarán la mayor cantidad de defectos posibles y de esta forma se garantizará, que el software llegue a su fase final libre de errores.

2.6 Ejecución de las pruebas automatizadas

Como resultado del estudio de varias herramientas y de acuerdo a una serie de criterios la herramienta Selenium fue seleccionada, para automatizar las pruebas funcionales a los sistemas desarrollados en el centro. A continuación se detalla brevemente como puede ser utilizada de acuerdo a las funcionalidades que ofrece en la verificación de la correspondencia de la apariencia externa del sistema con las especificaciones brindadas por el equipo de desarrollo.

Pasos para la instalación de la herramienta Selenium

1. Se debe conectar a la página de descarga de SeleniumHQ, en la siguiente dirección: **http://seleniumhq.org y seleccionar Selenium IDE.**
2. Luego se instala.
3. Una vez instalado Selenium IDE se reinicia Firefox y ya lo tiene disponible.

También para instalar la herramienta puede ir a **http://eva.uci.cu/** y seguidamente a Todos los cursos->3er.Año->Segundo Semestre->Ingeniería de Software II->Semana 10 ->Herramientas->Selenium_IDE y luego descargar la carpeta con el nombre de **Selenium-ide-1.0.6.xpi** en el Escritorio para después instalarla.

Opciones de Comando

1. **Open (url):** Abre un sitio en el marco de ensayo. Este acepta URL tanto relativas como absolutas URL. El comando "open" espera a que la página se cargue antes de continuar, es decir. El "AndWait sufijo" está implícito.

2. **Type (locator, value):** Establece el valor de un campo de entrada, también puede ser usado para fijar el valor de los cuadros combinados, casillas de verificación, en estos casos, el valor debería ser el valor de la opción seleccionada.
3. **ClickAndWait (locator):** Pincha en un enlace, botón, casilla de verificación o botón de opción. Espera a que la página se recargue, usualmente lo hace con un enlace.
4. **Select (selectLocator, optionLocator):** Selecciona una opción de un menú desplegable mediante un localizador de opciones.

Al seleccionar un comando en el IDE Selenium, en el Área de Información se ofrece una breve referencia de lo que significa cada comando perteneciente al mismo.

Pasos para ejecutar las pruebas con Selenium IDE

1. Para visualizar el IDE primeramente tiene que tener abierta la ventana de Firefox y luego ir a Herramientas (Tools) -> Selenium IDE y se muestra como un Panel dentro de la ventana principal de Firefox.
2. Después de tener abierto el IDE habilite el botón grabar (debe quedar de color rojo claro) y luego escriba la URL que va a usar de base para la prueba. (Ver anexo 10)
3. Luego empiece a navegar por la aplicación a la que se le va a hacer las pruebas y a medida que vaya haciendo esto, el IDE va generando algunos comandos en el Panel de comandos, de acuerdo a lo que se haga en el sitio. (Ver anexo 11)
4. Después de realizar las pruebas se debe desactivar el botón grabar, y para obtener mejor información de la prueba y más detalle de la ejecución de la misma debe ir a Selenium TestRunner y reproducirla. (Ver anexo 12)
5. Luego se puede ver si la prueba fallo o no, en el Selenium TestRunner.

La información completa de cómo puede ser ejecutado un caso de prueba utilizando la herramienta seleccionada se encuentra en el documento "Manual de Uso de la Herramienta", uno de los resultados anexos a esta investigación y que puede ser consultado para obtener mayor información de la misma. Durante esa fase se ejecutan las pruebas con la herramienta seleccionada, se registran los resultados

obtenidos para posteriormente ser analizados y llegar a conclusiones relacionadas con el cumplimiento por el software de los requisitos especificados por el cliente.

2.6.1 Documentación de las pruebas ejecutadas

Después que los CP han sido diseñados y el ambiente de pruebas se encuentra preparado, se puede iniciar la ejecución de las mismas. Durante la tercera fase del procedimiento, se procede a ejecutarlas con la herramienta seleccionada, para ello es necesario que se tengan instaladas las herramientas y el personal este adiestrado con la herramienta y la aplicación.

Las actividades más importantes a desarrollar dentro de la ejecución de las pruebas son ejecutar las pruebas planificadas y registrar los resultados de la ejecución de las pruebas. El probador analiza el CP, siguiendo las especificaciones paso a paso descritas en el mismo. Una vez ejecutadas las pruebas, los resultados se documentarán en la tabla que se especifica a continuación y que forma parte del epígrafe **Iteraciones** perteneciente a cada uno de los documentos de Realización de casos de prueba confeccionados para los caso de uso seleccionados para las pruebas funcionales automatizadas al sistema.

Clases válidas	Clases invalidas	Resultado esperado	Resultado de las pruebas	Observaciones

Tabla 2.2: Plantilla de los resultados de las pruebas

Al terminar el CP, se adjuntarán todos los documentos producidos durante la prueba, como informes, imágenes de errores, etc. La evidencia íntegra de los resultados obtenidos en cada caso de prueba ejecutado será registrada en el epígrafe **Documentación de los Resultados** del Plan de Pruebas confeccionado y en un documento elaborado con los Resultados de las Pruebas, los cuáles pueden ser consultados para obtener mayor información de los mismos.

2.7 Análisis de los resultados

La cuarta y última fase del procedimiento de pruebas es el análisis de los resultados, en él se procede al análisis de los mismos en función de las pruebas realizadas. Los resultados se comparan con los criterios especificados por los diseñadores, documentando los criterios de los probadores respecto a los resultados, para dar las pruebas por satisfactorias o para aceptar como bueno el producto.

El reporte claro y preciso de los defectos encontrados en la aplicación durante el proceso de pruebas, se realizará en el formulario que se muestra a continuación y que se encuentra en el epígrafe **Registro de defectos y dificultades detectados**, de cada uno de los documentos de Realización de casos de prueba que serán confeccionados para cada CU seleccionado, el cual se debe asociar a la documentación generada por esta investigación, quedando registrados en éste todos los resultados de las mismas.

Elementos	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación

Tabla 2.3: Plantilla de Registro de defectos y dificultades detectados.

Además de ser documentados en las plantillas de CP diseñadas para cada CU, los mismos pueden ser consultados en el documento de Resultado de las Pruebas ejecutadas, uno de los resultados más importantes de la investigación, pues el mismo contendrá los resultados de la aplicación del procedimiento a los sistemas deseados.

Durante esta fase se realiza la validación de los resultados de las pruebas verificando si los resultados de la aplicación del procedimiento son superiores a los obtenidos en las pruebas manuales desarrolladas al sistema. Los resultados serán revisados para consumar el respectivo análisis de los mismos, posibilitar que el grupo de desarrollo tenga conocimientos de los errores que pueden ser solucionados con futuras versiones con el fin de alcanzar el objetivo fundamental del mismo: un software que funcione tal como fue requerido por el cliente.

2.8 Conclusiones del Capítulo

En este capítulo se realizó la descripción de una guía para realizar pruebas funcionales automatizadas en aplicaciones Web. Para la misma fue necesario planificar el avance de la solución reflejado en el plan de pruebas confeccionado, así como la estrategia a seguir en el diseño de los casos de pruebas del sistema. Se espera que con los resultados de la aplicación de las pruebas funcionales en el sistema que sea seleccionado se demuestre su fiabilidad, partiendo de que no existe una iniciativa precedente de la aplicación de pruebas funcionales automatizadas en las aplicaciones web de la universidad.

Capítulo 3: Aplicación del procedimiento al Sistema de Información de Perforación de Pozos Petroleros.

El objetivo de este capítulo es ejecutar pruebas funcionales automatizadas en Sistema de Información de Perforación de Pozos Petroleros y documentar los resultados con el fin de obtener conclusiones precisas fundamentadas en un análisis del comportamiento de los mismos. Con el estudio comparativo de los resultados diseñados con los obtenidos durante la ejecución de las pruebas se demostrará si el desarrollo de pruebas automatizadas en aplicaciones web eleva la efectividad de las mismas y por tanto el nivel de satisfacción de los clientes.

3.1 Sistema seleccionado para las pruebas

Uno de los proyectos pertenecientes al centro es desarrollado en conjunto con la Dirección de Intervención y Perforación de Pozos (DIPP), en la que el proceso de gestión, control y flujo de información era realizado manualmente, lo que trae consigo, redundancia en la información, así como errores y desactualización en la misma, además de que no existe un orden y control para el almacenamiento de la información. La solución a los problemas presentes en DIPP fue alcanzada con el desarrollo de un sistema (Sistema de Información de Perforación de Pozos Petroleros) (Anexo 8) que permita gestionar la información generada en los pozos de petróleo en perforación, de manera tal que se eliminen las ambigüedades en la información y se ahorre tiempo en el proceso, ya que del éxito de la realización de estos reportes y/o partes, depende mucho la toma de decisiones de esta entidad. Además este sistema propicia un mejor control de la información generada, así como un acceso más rápido y simple a la información. (24)

Para la modelación del mismo se utilizó RUP por ser una metodología que genera documentación útil sobre el negocio. Al ser iterativo e incremental posibilita tener al cliente en constante retroalimentación, lo cual brinda un desarrollo más rápido y efectivo del sistema. El Lenguaje Unificado de Modelado (UML) es el lenguaje de modelado de objetos empleado para modelar el sistema, debido a sus características, probada eficacia y prefecta complementación con la metodología de desarrollo RUP. La herramienta de ingeniería de sistemas asistida por ordenador (Computer Aided Systems Engineering – CASE) destinada en la automatización del ciclo de vida del software es Visual Paradigm for UML 6.1 Enterprise Edition (Anexo 9) por ser independiente de la metodología que se utiliza, soportar UML 2.0,

generar código directo e inverso en PHP 5.0, permitir el Modelado de Base de Datos, generar automáticamente el modelo entidad-relación, el modelo objeto- relación, a partir de un modelo de clases persistentes así como código directo e inverso para Postgree SQL. Como Framework de trabajo contaremos con Framework Symfony (Anexo 10) y Lenguaje de Programación PHP5, o sea utilización de tecnología orientada a objetos y uso de la Arquitectura MVC con sus ventajas y desventajas. (24)

3.2 Planificación de las pruebas

Cada prueba que se realice debe ser planificada con antelación, para lograr que los resultados alcanzados sean los esperados y evitar la improvisación durante su ejecución (14). Antes de realizar las pruebas fueron consultados una serie de documentos con el objetivo de tener un pleno conocimiento de las funcionalidades del sistema y todos los requisitos funcionales pertenecientes al sistema fueron profundamente estudiados antes de ponerlos a prueba. A continuación se mencionan algunos documentos consultados para la realización de las pruebas.

1. Documento de especificación de requisitos.
2. Especificación de los casos de uso (CU).

La planificación de las pruebas funcionales automatizadas quedó reflejada en el documento “Plan de Pruebas Sistema Perforación de Pozos” (Anexo 11), en el mismo se explica alcance, requerimientos a probar, estrategia y los recursos requeridos, las herramientas así como los responsables involucrados en el proceso de pruebas y constan los entregables como salidas generadas en cada una de las actividades.

3.2.1 Requisitos a probar

Todas las capacidades o condiciones que los clientes, usuarios y miembros del equipo de proyecto definen que el sistema debe cumplir, constituyen requisitos que deben ser probados. Para ser analizados es necesario aplicar pruebas funcionales, de comportamiento o de caja negra como también puede llamarse, sobre el software funcionando para ver si la entrada y salida de los datos se corresponde con lo que ha sido especificado por el equipo de desarrollo. De los requisitos definidos para la construcción del sistema, se listarán aquellos que formaran parte de la estrategia utilizada para

la elaboración de este procedimiento:

RF 1 Gestionar Inventario de Barrena

RF 1.1 Insertar Datos en el Inventario de Barrena

RF 1.2 Modificar Datos en el Inventario de Barrena

RF 2 Generar Reporte Diario Operativo del Pozo

RF 10 Gestionar Inventario de Motor Fondo

RF 1.1 Insertar Datos en el Inventario de Motor Fondo

RF 1.2 Modificar Datos en el Inventario de Motor Fondo

RF 23 Generar Parte Diario de Perforación

RF 32 Autenticar Usuario

3.2.2 Casos de Uso

Los CU agrupan los requisitos funcionales que responden a los requerimientos derivados del modelo del negocio. Partiendo de la base de los requisitos funcionales analizados anteriormente, los CU elegidos para el desarrollo de las pruebas funcionales automatizadas al sistema seleccionado se relacionan a continuación:

1. Autenticar Usuario.
2. Generar Reporte Diario de Perforación.
3. Generar Reporte Diario del Pozo.
4. Gestionar Inventario de Barrena
5. Gestionar Inventario de Motor Fondo.

Para verificar que la funcionalidad del sistema se corresponde con lo especificado por el equipo de desarrollo se efectuarán pruebas a los requisitos funcionales y los casos de uso seleccionados por los

autores de la investigación. Todo lo planificado anteriormente constituye un elemento fundamental en el éxito de las pruebas funcionales automatizadas al sistema seleccionado, ya que para obtener un buen resultado se requiere de una buena estrategia, la cual se encuentra definida en el documento "Plan de Pruebas Sistema Perforación de Pozos", donde se definieron el objetivo de las pruebas, la técnica a emplear así como el entorno donde se desarrollan las pruebas, las actividades que se deben desarrollar durante el proceso de pruebas, los casos de pruebas que posibilitan la verificación de las funcionalidades del sistema, los criterios de términos y las herramientas a usar durante las pruebas.

3.2.3 Estrategia de pruebas

El éxito de la ejecución de las pruebas dependerá del seguimiento de una estrategia de prueba adecuada (15). La estrategia de prueba de software que seguiremos en el desarrollo de las pruebas funcionales automatizadas integra un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba. Antes de comenzar a realizar la estrategia se encuestó (Anexos 12 y 13) a varias personas implicadas con las pruebas en algunos proyectos del CEDIN, con el objetivo de conocer la situación actual de las pruebas en los proyectos del mismo, los resultados obtenidos en ella son los siguientes:

La figura 2.1 representa del total de personas encuestadas aquellas que tienen conocimiento de que es una prueba funcional y la cantidad de personas que las aplican en sus proyectos así como cuántos conocen de otros proyectos donde se aplican.

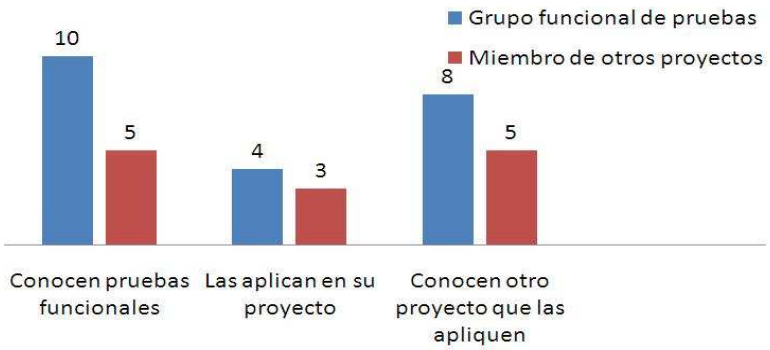


Figura 2.1: Personas que conocen las pruebas y las aplican en su proyecto.

La figura 2.2 representa del total de personas encuestadas aquellas que conocen las herramientas de automatización de pruebas funcionales, cuántas las utilizan en su proyecto y la cantidad que conocen otros proyectos donde se emplean.

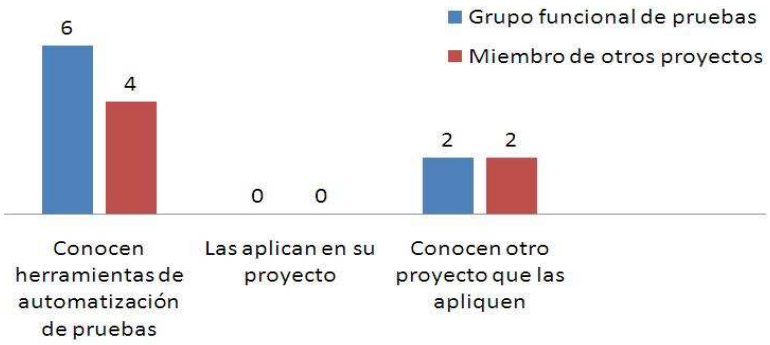


Figura 2.2: Personas que conocen las herramientas y las utilizan en su proyecto.

En la búsqueda de un desarrollo más eficiente de las pruebas funcionales en el sistema seleccionado, se definirán los pasos a seguir para realizar de una mejor manera las pruebas funcionales, influyendo significativamente en la obtención de un producto con mayor calidad. El método de prueba que se decidió utilizar en el proceso de pruebas aplicado al sistema es el método de Caja Negra, con el fin de estudiar la especificación de las de las funciones, la entrada y la salida para poder derivar los casos de prueba, definiendo como algo fundamental el probar todas las posibles entradas y salidas del sistema que se está probando. Las técnicas de prueba de Caja Negra escogidas para el Sistema son la de Partición Equivalente y Análisis de los Valores Límites, las mismas ayudarán a diseñar casos de pruebas efectivos, que permiten cubrir el mayor número de clases válidas y no válidas, con las que se garantizará que la entrada y salida de datos al sistema sea la más correcta posible. (Anexo 14)

3.3 Diseño de las pruebas

Para realizar el diseño de los casos de prueba funcionales se debe contar con la especificación de cómo se debe interactuar con el sistema (Especificación de caso de uso) y el sistema, que en fin es lo que se probará. El diseño de las pruebas y los resultados de las mismas, quedaron establecidos en los

documentos "Diseño de Casos de Pruebas funcionales" elaborados para cada uno de los CU seleccionados.

Por cada CU se realizó un CP exceptuando los dos Gestionar que se componen por dos sesiones y se diseño un CP para cada una de ellas; los CP están formados por las clases de equivalencias diseñadas, algunas válidas y otras no válidas, obteniéndose un resultado en cada una de las pruebas que se ejecutaron. Los CP identificados por cada caso de uso del sistema seleccionado para las pruebas se relacionan a continuación:

Caso de Uso	**Funcionalidades**
Autenticar Usuario.	Autenticar Usuario.
Generar Reporte Diario de Perforación.	Generar Parte Diario de Perforación.
Generar Reporte Diario del Pozo.	Generar Reporte Diario del Pozo.
Gestionar Inventario de Barrena.	Insertar Datos en el Inventario Barrena.
	Modificar Datos en el Inventario Barrena.
Gestionar Inventario de Motor Fondo.	Insertar Datos en el Inventario de Motor Fondo.
	Modificar Datos en el Inventario de Motor Fondo.

Tabla 2.1: Casos de pruebas diseñados por cada caso de uso seleccionado.

La siguiente tabla resume la información derivada del diseño y aplicación de las pruebas, para los 7 CP diseñados se obtuvo un total de 34 clases válidas y no válidas.

Nombre del caso de prueba	**Clases de equivalencias diseñadas**
Autenticar Usuario.	16 Clases de equivalencias, 5 válidas y 11 inválidas.
Generar Reporte Diario de Perforación.	1 Clase de equivalencia válida.

Generar Reporte Diario del Pozo.	4 Clases de equivalencias, 1 válida y 3 inválidas.
Insertar Datos en el Inventario de Barrena.	3 Clases de equivalencias, 1 válida y 2 inválidas.
Modificar Datos en el Inventario de Barrena.	3 Clases de equivalencias, 1 válida y 2 inválidas.
Insertar Datos en el Inventario de Motor Fondo.	3 Clases de equivalencias, 1 válida y 2 inválidas.
Modificar Datos en el Inventario de Motor Fondo.	4 Clases de equivalencias, 1 válida y 3 inválidas.

Tabla 3.2: Resumen del diseño y aplicación de las pruebas.

La información completa del DCP del sistema; puede ser consultada en cada uno de los documentos "Diseño de Casos de pruebas funcionales" confeccionados; el personal encargado de poner en práctica las pruebas, debe tener un dominio completo de la aplicación y una forma de poder acceder a ella.

3.4 Ejecución y documentación de las pruebas

Para mostrar los resultados obtenidos en las pruebas se han confeccionado una serie de gráficos que muestran el comportamiento de los resultados del diseño y de la ejecución de las pruebas. A continuación se realiza un resumen de los principales resultados obtenidos.

La figura 3.1 representa el total de casos de pruebas diseñados y ejecutados (7), destacando la ejecución del 100 % de los casos de pruebas diseñados.

Figura 3.1: Total de casos de pruebas ejecutados.

La figura 3.2 muestra el total de clases de equivalencia probadas (34) para los 7 CP, diferenciado en estas las clases válidas (11) y las clases no válidas (23) identificadas para los CP ejecutados.

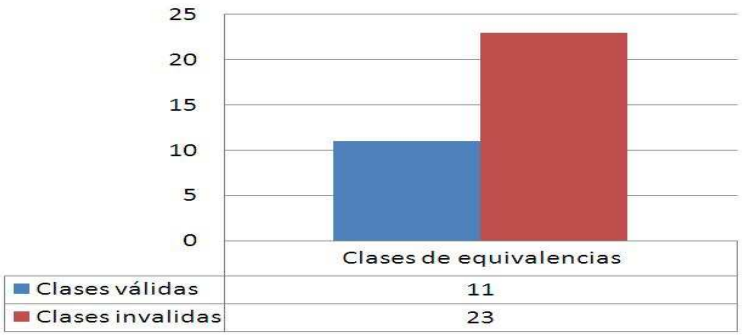


Figura 3.2: Total de clases de equivalencia ejecutadas.

La figura 3.3 expone la cantidad de errores encontrados en el sistema teniendo en cuenta los CU seleccionados para aplicar las pruebas diseñadas. Para los 5 CU se detectaron 10 errores, 3 para el CU Autenticar Usuario, 3 para el CU Generar Reporte Diario de Pozo, 2 para el CU Gestionar Inventario de Barrena y 2 para el CU Gestionar Inventario de Motor Fondo lo que representa el 30 % de errores para el Autenticar Usuario, el 30 % de errores para el Generar Reporte Diario de Pozo, el 20 % de errores para el CU Gestionar Inventario de Barrena y el 20 % de errores para el CU Gestionar Inventario de Motor Fondo, del total de errores encontrados.

Figura 3.3: Total de errores en cada caso de uso probado.

La figura 3.4 expone los resultados de los errores encontrados en los CP a los que se le aplicaron las pruebas diseñadas. Los 7 CP ejecutados contaron con 34 clases de equivalencia probadas, encontrándose 8 errores de Importancia alta y 2 de Importancia baja con un total de 10 errores, lo que representa un 80 % de errores de Importancia alta y un 20 % de errores de Importancia alta, del total de errores encontrados.

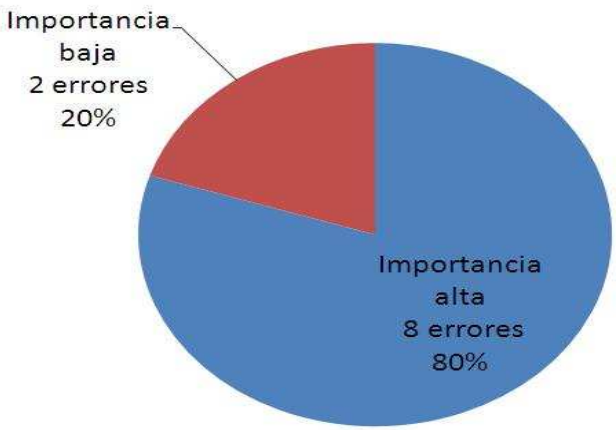


Figura 3.4: Total de errores en las pruebas aplicadas.

Al terminar el CP, se adjuntarán todos los documentos producidos durante la prueba, como informes, imágenes de errores, etc. La evidencia íntegra de los resultados obtenidos en cada caso de prueba ejecutado también será registrada en cada uno de los documentos "Realización de Casos de pruebas funcionales" confeccionados y en el documento "Resultados de las Pruebas", los cuáles pueden ser consultados para obtener una mayor información de la misma.

3.5 Análisis de los resultados de las pruebas

Para analizar si los resultados obtenidos en las pruebas automatizadas son los esperados se realiza una comparación con los resultados de las pruebas manuales, la misma se ofrece a través de una serie de gráficos confeccionados para mostrar el comportamiento de las pruebas al ser ejecutadas en las dos formas existentes. A continuación se realiza un resumen de los principales gráficos derivados de los resultados antes mencionados:

La figura 3.5 representa el tiempo utilizado (segundos), en ambos tipos de pruebas, para realizar los casos de pruebas teniendo en cuenta cada uno de los casos de uso seleccionados.

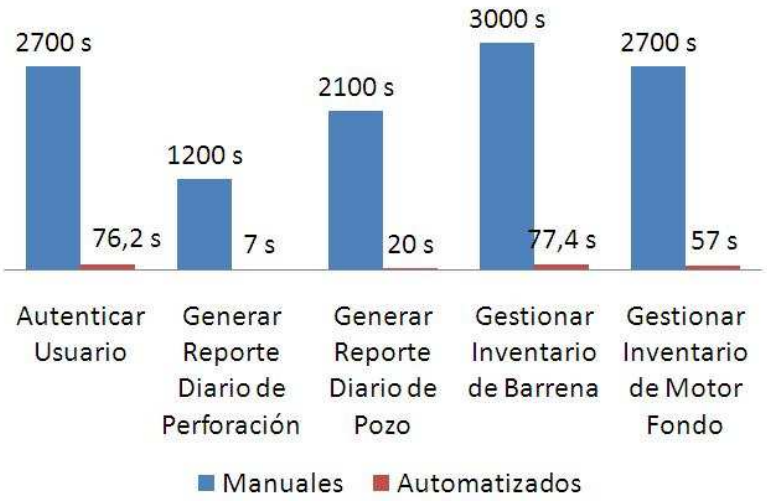


Figura 3.5: Tiempo necesitado en cada caso de uso.

La figura 3.6 muestra el tiempo total empleado (minutos), para realizar los casos de pruebas en cada forma de ejecutar las pruebas.

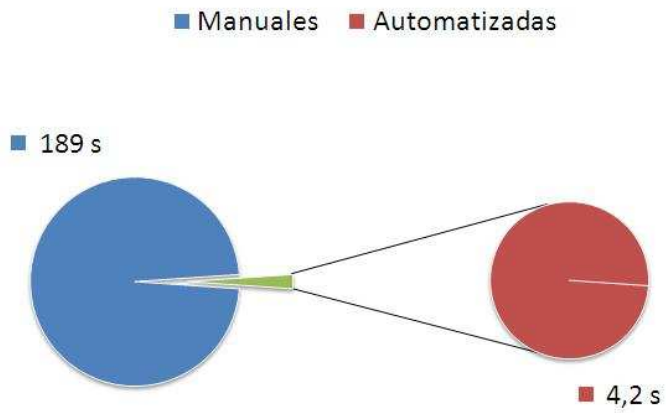
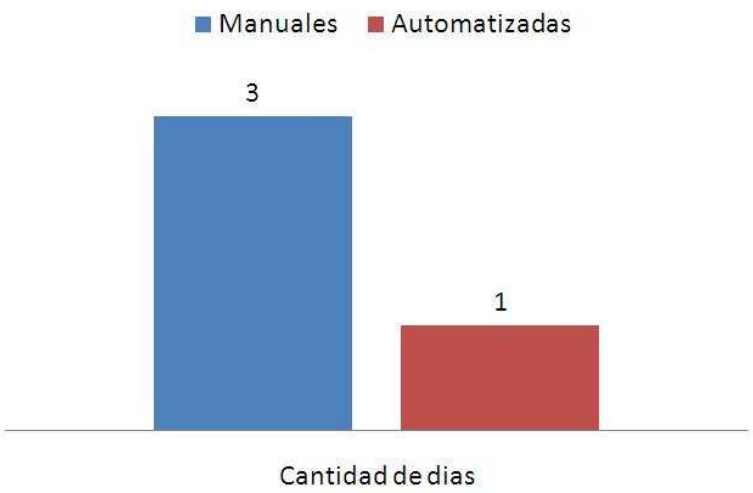


Figura 3.6: Tiempo total empleado para las pruebas.

La figura 3.7 expone la cantidad de días empleados para realizar las pruebas, en las dos formas existentes.



. **Figura 3.7: Cantidad de días empleados para las pruebas.**

La figura 3.8 expone la cantidad de computadoras utilizadas por los probadores para realizar las pruebas en las dos formas existentes.

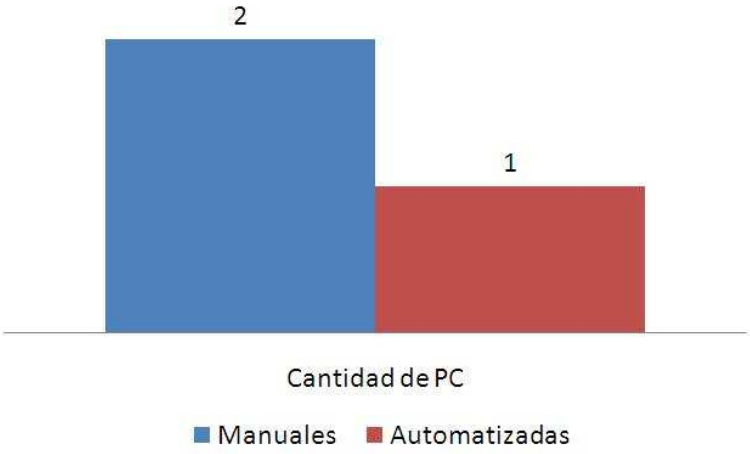


Figura 3.8: Cantidad de computadoras utilizadas para las pruebas.

La figura 3.9 expone la cantidad de probadores que realizaron las pruebas, en las dos formas existentes.

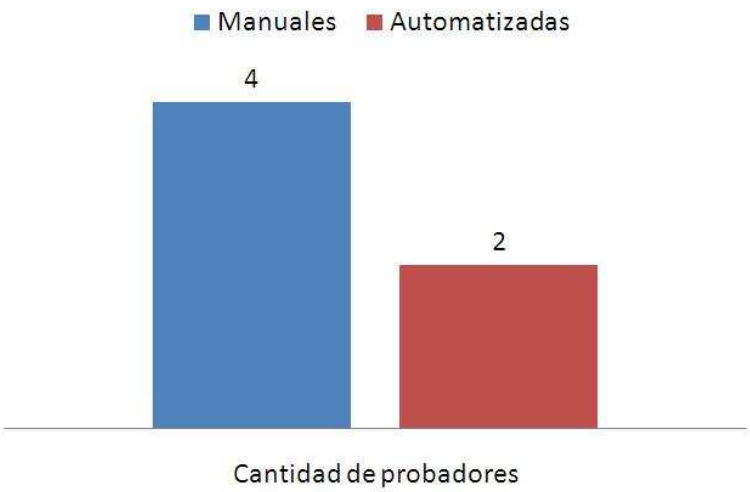


Figura 3.9: Cantidad de probadores que realizaron las pruebas.

La totalidad de la información derivada de la comparación de las pruebas manuales y automatizadas, se encuentra en el documento "Resultados de las Pruebas", que contiene los resultados del proceso desarrollado, los cuales pueden ser consultados para obtener mayor información de los mismos.

3.6 Conclusiones del Capítulo

En este capítulo se ejecutaron las pruebas funcionales automatizadas en el Sistema de Información de Perforación de Pozos Petroleros y se realizó un análisis de los resultados obtenidos durante las mismas fundamentadas en un estudio comparativo de los resultados diseñados con los obtenidos permitiendo demostrar que las pruebas automatizadas permiten facilitar el trabajo del probador, posibilitando ejecutar las pruebas en menos tiempo y con menos recursos que las pruebas manuales.

Conclusiones

Al aplicar el procedimiento de las pruebas funcionales automatizadas en el Sistema de Manejo Integral de Perforación de Pozos se llegó a las siguientes conclusiones:

Fueron analizadas varias herramientas de automatización y seleccionada la más apropiada para las pruebas funcionales a las aplicaciones web, y se elaboró un procedimiento para utilizar la herramienta de automatización elegida en las pruebas funcionales a las aplicaciones web del centro.

Los casos de prueba diseñados cubrieron el 50 % de las funcionalidades del sistema, posibilitando la detección de un conjunto de no conformidades que fueron solucionadas de manera satisfactoria por el equipo de desarrollo.

La documentación generada de las pruebas realizadas, servirá de apoyo para la corrección de los errores presentes en los futuros productos que se desarrollen.

Los resultados obtenidos en las pruebas automatizadas demuestran que requieren menos recursos y tiempo que las manuales, por lo que constituyen una vía eficaz para facilitar el trabajo de los probadores permitiéndoles encontrar errores de mayor magnitud en el menor tiempo posible.

Recomendaciones

Al término de la investigación y luego del análisis de los resultados obtenidos se recomienda que:

1. La herramienta propuesta sea investigada a fondo y extendida al proceso de pruebas funcionales a todos los proyectos desarrollados en el centro.

2. La herramienta sea integrada a las restantes herramientas de automatización y gestión de pruebas utilizadas en el centro.

3. Se realice el estudio de las herramientas que puedan surgir en el futuro y que puedan servir de apoyo a las pruebas funcionales desarrolladas a los sistemas web de la universidad.

4. Teniendo en cuenta que solo se realizan las pruebas al 50 % de las funcionalidades del sistema, se recomienda que se desarrollen al 50 % restante.

Referencias bibliográficas

1. Pressman, R. S. (2005). Ingeniería de Software. Un enfoque práctico. Mc Graw Hill.
2. IEEE. 1990. Standard 610, Computer Dictionary. 1990.
3. RUP. 2003. Ayuda del Rational Unified Process. 2003.
4. Jr, F. A. (21 de Abril de 2008). Jaipan Group. Recuperado el 25 de Enero de 2010, de Jaipan Group: http://www.yaipan.com/joomla/index.php/editorial/61
5. Dustin E. 2003. Effective Software testing. Pearson Education.
6. Zuyu J., Tsao J., Wu Y. 2003. Testing y Quality assurance for component-based software.
7. Ilene Burnstein. Practical Software Testing.Springer. NY Estados Unidos.
8. Hetzel, Bill. The Complete Guide to Software Testing. 1988.
9. Davis, A. 201 Principles of Software Development. s.l.: MaGraw-Hill, 1995.
10. Myers, G. The art of software testing. 1979.
11. Lianet Aguilera Reyes, A. C. (Julio de 2007). Sistema Asistente para Gestionar Pruebas. Boyeros, Ciudad de la Habana, Cuba.
12. Conferencia 7 de Ingeniería de Software ll, Curso 2009-2010.Disponible en: http://eva.uci.cu/file.php/259/Curso_20092010/Conferencia_7/Materiales_basicos/Conferencia_7_Disciplina_Prueba.doc
13. Rodríguez, J. J. (2006). Generación de pruebas del sistema a partir de la especificación funcional. Sevilla, España. Obtenido de http://www.lsi.us.es/docs/doctorado/proyectos_tesis/Memoria%20JavierGutierrez.pdf

14. Susana González Espinosa, D. D. (Junio de 2008). Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías. Boyeros, Ciudad de la Habana, Cuba.

15. Jacobson, Booch, Rumbaugh. El Proceso Unificado de Desarrollo de Software, Madrid, 2003, Pearson Education S.A.

16. Teruel, Alejandro El Plan de Pruebas, 2001, [Disponible en: http://www.ldc.usb.ve/~teruel/ci4713/clases2001/planPruebas.html]

17. Mañas, José A. Pruebas de Programas, 1994. [Disponible en: http://www.it.uc3m.es/tsps/testing.htm]

18. Biblioteca Nacional de Maestros. [En línea] 2008. [Citado el: 18 de Febrero de 2010.] http://www.me.gov.ar/bnm/.

19. Universidad Distrital Francisco José de Caldas. [En línea] 2008. [Citado el: 2 de Marzo de 2010.] http://www.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/.

20. Díaz, H. M. (s.f.). Recuperado el 5 de Diciembre de 2009, de http://hamd.galeon.com

21. (s.f.). Recuperado el 20 de Mayo de 2010, de http://watin.sourceforge.net

22. (8 de Mayo de 2008). Recuperado el 18 de Mayo de 2010, de http://www.jourmoly.com.ar

23. (s.f.). Recuperado el 20 de Mayo de 2010, de http://watir.com/comunidad

24. Cuevas, D. T. (Julio de 2009). Análisis y diseño del sistema de manejo integral de perforación de pozos. Boyeros, Ciudad de la Habana, Cuba.

Bibliografía

1. Cuevas, D. T. (Julio de 2009). Análisis y diseño del sistema de manejo integral de perforación de pozos. Boyeros, Ciudad de la Habana, Cuba.
2. Davis, A. 201 Principles of Software Development. s.l.: MaGraw-Hill, 1995.
3. Dustin E. 2003. Effective Software testing. Pearson Education.
4. Hetzel, Bill. The Complete Guide to Software Testing. 1988.
5. IEEE. 1990. Standard 610, Computer Dictionary. 1990.
6. Ilene Burnstein. Practical Software Testing.Springer. NY Estados Unidos.
7. Jacobson, Booch, Rumbaugh. El Proceso Unificado de Desarrollo de Software, Madrid, 2003, Pearson Education S.A.
8. Jr, F. A. (21 de Abril de 2008). Jaipan Group. Recuperado el 25 de Enero de 2010, de Jaipan Group: http://www.yaipan.com/joomla/index.php/editorial/61
9. Lianet Aguilera Reyes, A. C. (Julio de 2007). Sistema Asistente para Gestionar Pruebas. Boyeros, Ciudad de la Habana, Cuba.
10. Mañas, José A. Pruebas de Programas, 1994. [Disponible en: http://www.it.uc3m.es/tsps/testing.htm]
11. Myers, G. The art of software testing. 1979.
12. Pressman, R. S. (2005). Ingeniería de Software. Un enfoque práctico. Mc Graw Hill.
13. Rodríguez, J. J. (2006). Generación de pruebas del sistema a partir de la especificación funcional. Sevilla, España. Obtenido de http://www.lsi.us.es/docs/doctorado/proyectos_tesis/Memoria%20JavierGutierrez.pdf

14. RUP. 2003. Ayuda del Rational Unified Process. 2003.

15. Susana González Espinosa, D. D. (Junio de 2008). Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías. Boyeros, Ciudad de la Habana, Cuba.

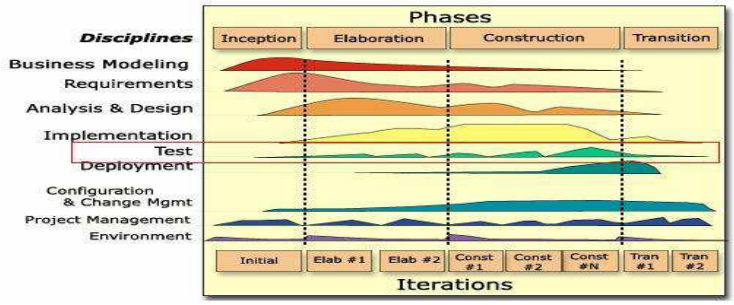
16. Teruel, Alejandro El Plan de Pruebas, 2001, [Disponible en: http://www.ldc.usb.ve/~teruel/ci4713/clases2001/planPruebas.html]

17. Velásquez, C. D. (2009). Propuesta metodológica para la realización de Pruebas Funcionales... Minas, Medellín, Colombia.

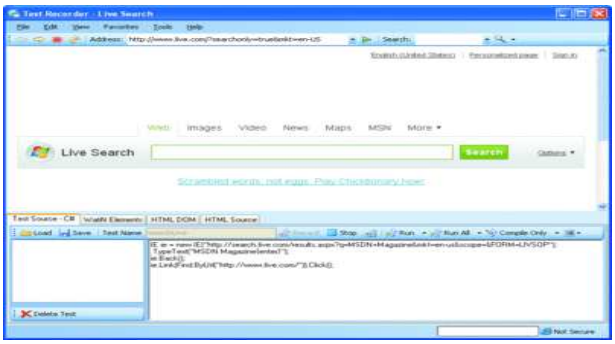
18. Zuyu J., Tsao J., Wu Y. 2003. Testing y Quality assurance for component-based software.

Anexos

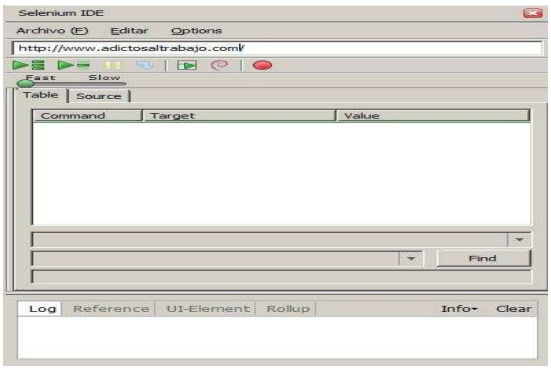
Anexo 1: Fases y flujos de trabajo de la Metodología RUP.



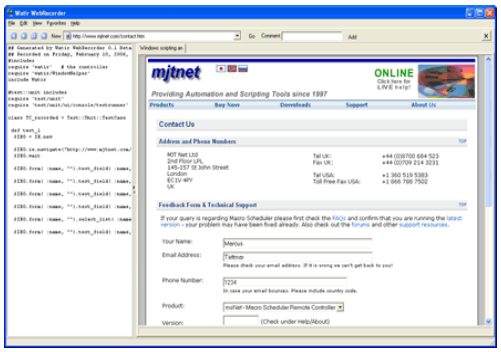
Anexo 2: WatiN (Web Application Testing)



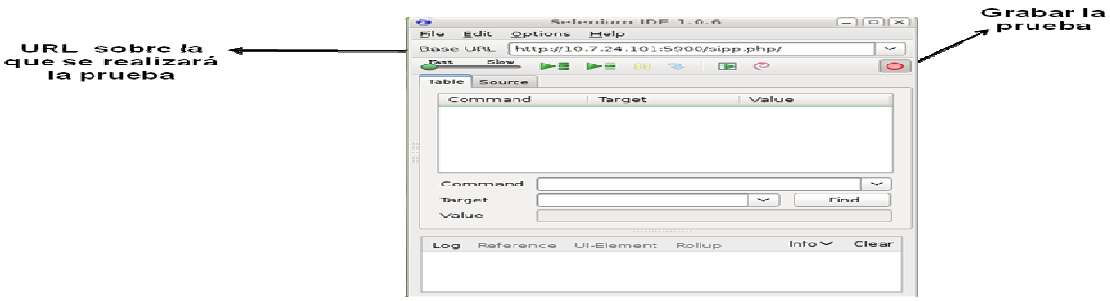
Anexo 3: Selenium



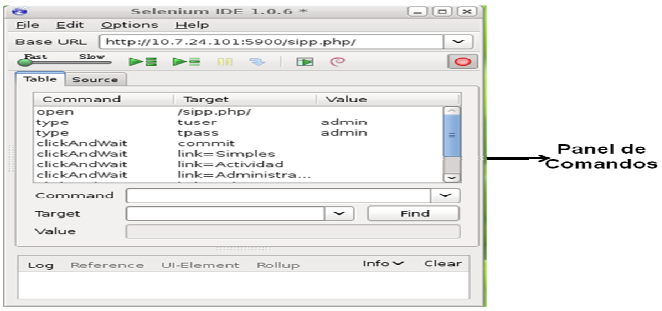
Anexo 4: Watir (Web Application Testing en Ruby)



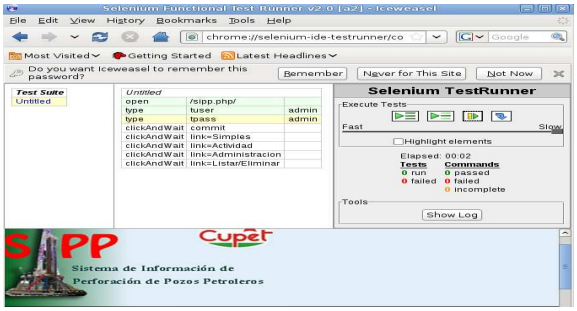
Anexo 5: Habilitar botón grabar y escribir URL sobre la que se realizara la prueba.



Anexo 6: Comandos generados en el IDE de acuerdo a lo que se haga en el sitio.

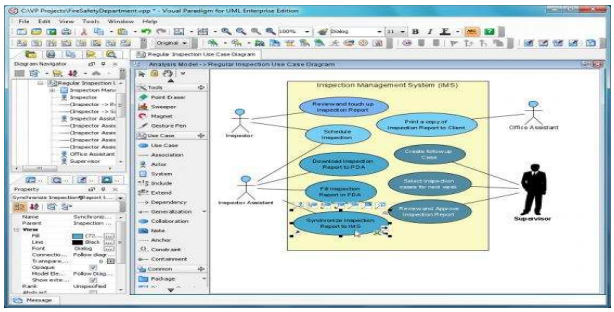


Anexo 7: Reproducción de la prueba con Selenium TestRunner

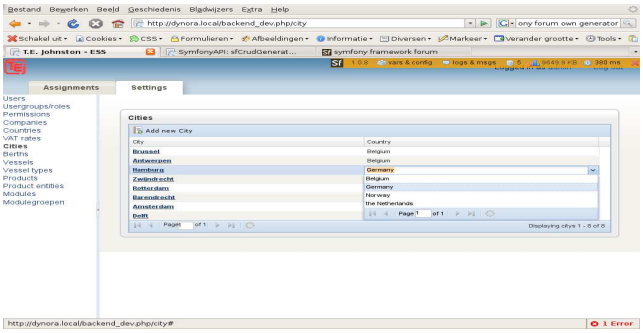


Anexo 8: Sistema de Información de Perforación de Pozos Petroleros.

Anexo 9: Visual Paradigm for UML 6.1 Enterprise Edition.



Anexo 10: Symfony, Framework para el desarrollo de aplicaciones web.



Anexo 11: Plan de Pruebas Sistema Perforación de Pozos

En el plan de pruebas desarrollado, se explica alcance, requerimientos a probar, estrategia y los recursos requeridos, calendario, herramientas así como los responsables involucrados en el proceso de pruebas y se mencionan los entregables que son salidas generadas en cada una de las actividades.

Alcance

La realización de pruebas funcionales es de suma importancia, cuando se habla de influir en la calidad del producto, específicamente en la validación del correcto funcionamiento del sistema frente a los requisitos del cliente. Durante el proceso de automatización de pruebas funcionales al se aplicarán las pruebas de Sistema, con el objetivo de verificar si la apariencia externa de este coincide con lo que especificó el equipo de desarrollo. De esta forma, todos los proyectos tendrían en sus manos un procedimiento para utilizar un software automatizado en pruebas interfaz de usuarios, para disminuir los problemas en cuanto al correcto funcionamiento de los sistemas web.

Organización del Equipo de Pruebas

El equipo de prueba que participará en el proceso de automatización de las pruebas funcionales al Sistema de Información de Perforación de Pozos Petroleros. estará compuesto por los dos autores de la investigación los cuales se comportarán fundamentalmente como, Diseñadores de pruebas, responsabilizados de definir la estrategia que guiará el proceso y asegurar así su aplicación exitosa. En menor medida

desempeñarán el rol de Ingenieros de pruebas de sistema, pues es el responsable de verificar el funcionamiento del sistema como un todo.

Descripción del Plan de Pruebas

El plan de pruebas estará compuesto por los requerimientos funcionales y los casos de uso a utilizar en las pruebas, la estrategia de pruebas que se aplicará al Sistema de Información de Perforación de Pozos Petroleros, los recursos necesarios (humanos, materiales y/o financieros), los procedimientos, guías y plantillas que se utilizarán, el esfuerzo necesario y el ambiente de prueba; así como quién debe aplicar las pruebas y cuándo estas deben aplicarse.

Anexo 12: Encuesta realizada a personas implicadas en el Proyecto SCADA-ETECSA.

Nombre y Apellidos _____ Proyecto _____

1. ¿Cómo evalúas el proceso de pruebas en el proyecto SCADA-ETECSA?
2. ¿Cuáles son las mayores dificultades presentadas en la etapa de prueba del proyecto SCADA-ETECSA?
3. ¿Existe actualmente en el proyecto alguna herramienta para la automatización de las pruebas?
4. ¿Cuánto crees que influye el proceso de pruebas en la calidad final del producto?
5. ¿Cuáles son las principales observaciones realizadas al proceso?

Anexo 13: Encuesta realizada a estudiantes y profesores miembros de proyectos productivos.

Nombre y Apellidos _____ Proyecto _____

1. ¿Sabes que son las pruebas de interfaz de usuario? Sí ___ No __
2. ¿Las aplicas a tu proyecto? Sí ___ No ___ No se __
3. ¿De qué forma se desarrolla el proceso? Manual ___ Automática ___
4. ¿Conoces algún proyecto al que se les apliquen? Sí ___ No ___

   Nombre del Proyecto _____ Manual ____ Automática ___
5. ¿Conoces alguna herramienta utilizada en el proceso de automatización de las pruebas interfaz de usuario?

   Sí ___ No __ Nombre de la herramienta _____
6. ¿Cuáles son las principales observaciones realizadas al proceso?

Anexo 14: Estrategia de pruebas

Con la automatización de las pruebas funcionales para el Sistema de Información de Perforación de Pozos Petroleros, todos los proyectos tendrían en sus manos un procedimiento para desarrollar las pruebas funcionales a aplicaciones web mediante un software automatizado, una vía efectiva para disminuir los problemas detectados durante la realización de las pruebas y que atentan contra el correcto funcionamiento de la interfaces de usuarios de los sistemas web.

Objetivo

Desarrollar una planificación de las pruebas funcionales para verificar que la interfaz de la aplicación a probar coincida con lo especificado por el equipo de desarrollo.

Técnica

El método de prueba que se decidió utilizar en el proceso de pruebas aplicado al sistema es el método de Caja Negra, con el fin de estudiar la especificación de las de las funciones, la entrada y la salida para poder derivar los casos de prueba, definiendo como algo fundamental el probar todas las posibles entradas y salidas del sistema que se está probando. Las técnicas de prueba de Caja Negra escogidas para el Sistema son la de Partición Equivalente y Análisis de los Valores Límites, las mismas ayudarán a

diseñar casos de pruebas efectivos, que permiten cubrir el mayor número de clases válidas y no válidas, con las que se garantizará que la entrada y salida de datos al sistema sea la más correcta posible.

Casos de Prueba

Para cada uno de los casos de uso seleccionados se realizaron los casos de pruebas que se relacionan a continuación:

Caso de Uso	**Funcionalidades**
Autenticar Usuario.	Autenticar Usuario.
Generar Reporte Diario de Perforación.	Generar Parte Diario de Perforación.
Generar Reporte Diario del Pozo.	Generar Reporte Diario del Pozo.
Gestionar Inventario de Barrena.	Insertar Datos en el Inventario Barrena.
	Modificar Datos en el Inventario Barrena.
Gestionar Inventario de Motor Fondo.	Insertar Datos en el Inventario de Motor Fondo.
	Modificar Datos en el Inventario de Motor Fondo.

Herramientas

Selenium: tiene código abierto a modificaciones necesarias, es controlada por los lenguajes de programación C#, Java, Perl y PHP, soporta en su instalación los sistemas operativos Windows 2000/XP/2003/ Vista, 7, Linux, Solaris y Mac OS, no hay que pagar derechos para ser utilizada por estar registrada bajo la licencia Apache 2.0 y tiene incluida las características siguientes, “Haz clic aquí”, “Espera a que se recargue la página”, “Completa tal campo de formulario”, “Haz clic en el Botón enviar”, “Verifica el resultado”, que permitan verificar la funcionalidad de la aplicación a prueba.

Anexo 15: Realización de Casos de Pruebas

CPR: Autenticar Usuario

Descripción de la Funcionalidad:

La funcionalidad permite verificar que se proporcione el acceso inicial al sistema solo a aquellos usuarios registrados.

Condiciones de Ejecución:

- Se Muestra un formulario solicitando usuario y contraseña.

Iteraciones.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones

Se llenan los campos siguientes: **Usuario:** admin **Contraseña:** admin>		<El sistema debe verificar que los datos introducidos son del tipo 1(Administrador del Sistema) y activar las ventanas del menú correspondiente a este usuario según sus privilegios>	Open /sipp.php/ Type tuser admin Type tpass admin ClickAndWait commit	La funcionalidad ha sido probada de forma eficiente, verificándose que el sistema verifica los datos introducidos y activa las opciones correspondientes a los privilegios del usuario.
<Se llenan los campos siguientes: **Usuario:** memartinez **Contraseña:** memartinez>		<El sistema debe verificar que los datos introducidos son del tipo 2(Secretaria de Despacho) y activar las ventanas del menú correspondiente a este usuario según sus	Open /sipp.php/ Type tuser memartinez Type tpass memartinez ClickAndWait commit	La funcionalidad ha sido probada de forma eficiente, verificándose que el sistema verifica los datos introducidos y activa las opciones correspondient

		privilegios**>**		es a los privilegios del usuario.

<Se llenan los campos siguientes: **Usuario:** rcalderon **Contraseña:** rcalderon **>**		<El sistema debe verificar que los datos introducidos son del tipo 3(Supervisor del Pozo) y activar las ventanas del menú correspondiente a este usuario según sus privilegios**>**	Open /sipp.php/ Type tuser rcalderon Type tpass rcalderon ClickAndWait commit	La funcionalidad ha sido probada de forma eficiente, verificándose que el sistema verifica los datos introducidos y activa las opciones correspondientes a los privilegios del usuario.
<Se llenan los campos siguientes: **Usuario:** agonzalez **Contraseña:** agonzalez >		<El sistema debe verificar que los datos introducidos son del tipo 4(Directivo) y activar las ventanas del menú correspondiente a este usuario según sus privilegios>	Open /sipp.php/ Type tuser agonzalez Type tpass agonzalez ClickAndWait commit	La funcionalidad ha sido probada de forma eficiente, verificándose que el sistema verifica los datos introducidos y activa las opciones correspondient

				es a los privilegios del usuario.
<Se llenan los campos siguientes: **Usuario:** jreyes **Contraseña**: jreyes >		<El sistema debe verificar que los datos introducidos son del tipo 5(Geólogo del Pozo) y activar las ventanas del menú correspondiente a este usuario según sus privilegios>	Open /sipp.php/ Type tuser jreyes Type tpass jreyes ClickAndWait commit	La funcionalidad ha sido probada de forma eficiente, verificándose que el sistema verifica los datos introducidos y activa las opciones correspondientes a los privilegios del usuario.
	<Se llenan los siguientes campos: **Usuario:** admin **Contraseña:** 123	<El sistema debe verificar la correspondencia de los datos y mostrar en pantalla el mensaje de error: “Usuario o Contraseña	Open /sipp.php/ Type tuser admin Type tpass 123 ClickAndWait commit verifyTextPresent Usuario o contraseña no	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema a un usuario cuya

	>	no válidos ">	válidos	contraseña no se corresponda con la que se encuentra almacenada en el sistema.

	<Se llenan los siguientes campos: **Usuario:** memartinez **Contraseña:** 123 >	<El sistema debe verificar la correspondencia de los datos y mostrar en pantalla el mensaje de error: “Usuario o Contraseña no válidos ”>	Open /sipp.php/ Type tuser memartinez Type tpass 123 ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema a un usuario cuya contraseña no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los siguientes campos: **Usuario:** rcalderon **Contraseña:** 123>	<El sistema debe verificar la correspondencia de los datos y mostrar en pantalla el mensaje de error: “Usuario o Contraseña no válidos ”>	Open /sipp.php/ Type tuser rcalderon Type tpass 123 ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema a un usuario cuya contraseña no se corresponda con la que se encuentra

				almacenada en el sistema.

	<Se llenan los siguientes campos: **Usuario:** agonzalez **Contraseña:** 123>	<El sistema debe verificar la correspondencia de los datos y mostrar en pantalla el mensaje de error: “Usuario o Contraseña no válidos ”>	Open /sipp.php/ Type tuser agonzalez Type tpass 123 ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema a un usuario cuya contraseña no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los siguientes campos: **Usuario:** jreyes **Contraseña:** 123>	<El sistema debe verificar la correspondencia de los datos y mostrar en pantalla el mensaje de error: “Usuario o Contraseña no válidos ”>	Open /sipp.php/ Type tuser jreyes Type tpass 123 ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema a un usuario cuya contraseña no se corresponda con la que se encuentra

				almacenada en el sistema.
	<Se llenan los campos siguientes: **Usuario:** **Contraseña:** admin >	<El sistema no debe permitir el acceso con el campo usuario en blanco>	Open /sipp.php/ Type tuser Type tpass admin ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo usuario sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los campos siguientes: **Usuario:** **Contraseña:** memartinez >	<El sistema no debe permitir el acceso con el campo usuario en blanco>	Open /sipp.php/ Type tuser Type tpass memartinez ClickAndWait commit verifyTextPresent	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo

			Usuario o contraseña no válidos	usuario sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los campos siguientes: **Usuario:** **Contraseña:** rcalderon >	<El sistema no debe permitir el acceso con el campo usuario en blanco>	Open /sipp.php/ Type tuser Type tpass rcalderon ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo usuario sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los campos siguientes:	<El sistema no debe permitir el acceso con el campo usuario	Open /sipp.php/ Type tuser Type tpass	La funcionalidad ha sido validada correctament

	Usuario: **Contraseña:** agonzalez >	en blanco>	agonzalez ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	e para sea denegado el acceso al sistema cuando el campo usuario sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los campos siguientes: **Usuario:** **Contraseña:** jreyes >	<El sistema no debe permitir el acceso con el campo usuario en blanco>	Open /sipp.php/ Type tuser Type tpass jreyes ClickAndWait commit verifyTextPresent Usuario o contraseña no válidos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo usuario sea dejado en blanco no se corresponda con la que se encuentra almacenada

				en el sistema.
	<Se llenan los siguientes campos: **Usuario:** admin **Contraseña:** >	<El sistema no debe permitir el acceso con el campo contraseña en blanco>	Open /sipp.php/ Type tuser admin Type tpass ClickAndWait commit	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo contraseña sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los siguientes campos: **Usuario:** memartinez **Contraseña:**	<El sistema no debe permitir el acceso con el campo contraseña en blanco>	Open /sipp.php/ Type tuser memartinez Type tpass ClickAndWait commit	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema

	>			cuando el campo contraseña sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los siguientes campos: **Usuario:** rcalderon **Contraseña:** >	<El sistema no debe permitir el acceso con el campo contraseña en blanco>	Open /sipp.php/ Type tuser rcalderon Type tpass ClickAndWait commit	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo contraseña sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.

	<Se llenan los siguientes campos: **Usuario:** agonzalez **Contraseña:** >	<El sistema no debe permitir el acceso con el campo contraseña en blanco>	Open /sipp.php/ Type tuser agonzalez Type tpass ClickAndWait commit	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo contraseña sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los siguientes campos: **Usuario:** jreyes **Contraseña:** >	<El sistema no debe permitir el acceso con el campo contraseña en blanco>	Open /sipp.php/ Type tuser jreyes Type tpass ClickAndWait commit	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando el campo contraseña

				sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.
	<Se llenan los siguientes campos: **Usuario:** **Contraseña:** >	<El sistema no debe permitir el acceso con los campos usuario y contraseña en blanco>	Open /sipp.php/ Type tuser Type tpass ClickAndWait commit verifiTextPresente Debe llenar todos los campos	La funcionalidad ha sido validada correctamente para sea denegado el acceso al sistema cuando los campos usuario y contraseña sea dejado en blanco no se corresponda con la que se encuentra almacenada en el sistema.

Registro de defectos y dificultades detectados

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendació
<Inicio>	<1>	<Campos Usuarios en blanco>	Cuando no se llena el campo correspondiente a nombre de usuario para acceder al sistema, este en vez de mostrar un mensaje de no puede dejar ese campo vacío, muestra el mensaje “Debe llenar todos los campos”.	<Pruebas>	<Alta>	<Poner un mensaje que diga que ese campo no puede estar vacío >
	<2>	<Campos Contraseña en blanco>	Cuando no se llena el campo correspondiente a la contraseña de un usuario para acceder al sistema, este en vez de mostrar un mensaje de no puede dejar ese	<Pruebas>	<Alta>	<Poner un mensaje que diga que ese campo no puede estar vacío>

			campo vacío, muestra el mensaje “Usuario o Contraseña no válidos”.			
	<3>	<Campos Usuario y Contraseña en blanco>	Cuando no se llenan los campos correspondiente al nombre de usuario y la contraseña para acceder al sistema, este en vez de mostrar un mensaje de no puede dejar esos campos vacíos, muestra el mensaje “Debe llenar todos los campos”.		<Alta>	<Poner un mensaje que diga que esos campos no pueden estar vacíos>

CPR: Gestionar Inventario de Barrena

Insertar Datos en el Inventario Barrena

Descripción de la Funcionalidad:

La funcionalidad permite verificar que se inserte de manera correcta los datos de barrena de los pozos.

Condiciones de Ejecución:

- El usuario debe estar previamente autenticado en el sistema.

Iteraciones.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
<El supervisor despliega la pestaña Inventarios, en la opción Inventario Barrenas selecciona la funcionalidad Crear, introduce los siguientes datos: **No.:** 4B **Tipo:** ATJ-05 **No. Serie:** 866824 **Fabricante:** HUGHES **IADC:** 437 **OD(mm):** 331		<El sistema debe analizar los datos y crear el inventario>	Open /sipp.php/ Type tuser rcalderon Type tpass rcalderon ClickAndWait commit ClickAndWait link=Crear Type numero 4B Type tipo ATJ-05 Type numeroSerie 866824 select fabricante HUGHES	La funcionalidad ha sido desarrollada de forma correcta para el envió de los datos sea realizado exitosamente a la hora de crear un inventario.

Longitud (m): 0.30 y da clic en el Botón Enviar>			select iadc 437 Type od 331 Type longitude 0.30 ClickAndWait //input[@value='Enviar'] ClickAndWait link=Crear	
	<El supervisor despliega la pestaña Inventarios, en la opción Inventario Barrenas selecciona la funcionalidad Crear, introduce los siguientes datos: **No.:** 4B **Tipo:** ATJ-05 **No. Serie:** 866824 **Fabricante:**	<El sistema no debe permitir crear el inventario pues hay las letras insertadas en los campos OD y Longitud que deben ser números>	Open /sipp.php/ Type tuser rcalderon Type tpass rcalderon ClickAndWait commit ClickAndWait link=Crear Type numero 4B Type tipo ATJ-05 Type numeroSerie	La funcionalidad no ha sido validada para que en los campos que solo se pueda introducir números no acepte otro tipo de caracteres.

	HUGHES **IADC:** 437 **OD(mm):** a **Longitud (m):** a y da clic en el Botón Enviar>		866824 select fabricante HUGHES select iadc 437 Type od a Type longitude a ClickAndWait //input[@value='Enviar'] ClickAndWait link=Crear assertAlert Debe llenar todos los campos	
	<El supervisor despliega la pestaña Inventarios, en la opción Inventario Barrenas	<El sistema debe analizar los datos y mostrar un mensaje de error en pantalla: “Debe llenar todos los	Open /sipp.php/ Type tuser rcalderon Type tpass rcalderon ClickAndWait	La funcionalidad ha sido validada correctamente para que al entrar datos

	selecciona la funcionalidad Crear, introduce los siguientes datos: **No.:** **Tipo:** **No. Serie:** **Fabricante:** **IADC:** **OD(mm):** **Longitud (m):** y da clic en el Botón Enviar>	campos”>	commit ClickAndWait link=Crear Type numero Type tipo ATJ- Type numeroSerie select fabricante select iadc Type od Type longitude ClickAndWait //input[@value='Enviar'] ClickAndWait link=Crear assertAlert Debe llenar todos los campos	para crear un inventario no se dejen campos vacios.

Modificar Datos del Inventario Barrena

Descripción de la Funcionalidad:

La funcionalidad permite verificar que se modifiquen de manera correcta la información de barrena de los pozos en perforación.

Condiciones de Ejecución:

- El usuario debe estar previamente autenticado en el sistema.

Iteraciones.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
<El supervisor despliega la pestaña Inventarios, en la opción Inventario Barrenas selecciona la funcionalidad Listar y Modificar, marca el inventario que desee modificar, cambia los datos existentes por: **No.:** 3B **Tipo:** ATJ-05 **No. Serie:**		<El sistema debe analizar los datos, modificar el inventario seleccionado y mostrar el mensaje “Los datos se han actualizado correctamente”>	Open /sipp.php7supervisor ClickAndWait link = listar y modificar Click inputtext49 Type inputtext50 3B Select inputtext 53 label= SMITH Select inputtext 54 label=111 ClickAndWait //input[@value='modificar']	La funcionalidad ha sido desarrollada de forma correcta para el envió de los datos sea realizado exitosamente a la hora de modificar un inventario.

866824 **Fabricante:** SMITH **IADC:** 111 **OD(mm):** 331 **Longitud (m):** 0.30 y da clic en el botón Modificar>				
	<El supervisor despliega la pestaña Inventarios, en la opción Inventario Barrenas selecciona la funcionalidad Listar y Modificar, marca el inventario que desee modificar, deja los datos de los campos a modificar vacios: **No.:**	<El sistema debe analizar los datos y mostrar un mensaje de error en pantalla: “Debe llenar todos los campos”>	ClickAndWait link = listar y modificar Click inputtext49 Type inputtext50 Type inputtext51 Type inputtext52 Type inputtext55 Type inputtext56 ClickAndWait	La funcionalidad ha sido validada correctamente para que al entrar datos para modificar un inventario no se dejen campos vacios.

	Tipo: **No. Serie:** **Fabricante:** **IADC:** **OD(mm):** **Longitud (m):** y da clic en el Botón Modificar>		//input[@value='modificar'] assertAlert Debe llenar todos los campos	
	<El supervisor despliega la pestaña Inventarios, en la opción Inventario Barrenas selecciona la funcionalidad Listar y Modificar, y da clic en el botón Modificar>	<El sistema debe analizar los datos y mostrar un mensaje de error en pantalla: "Debe marcar la(s) filas(s) a modificar">		La funcionalidad ha sido validada correctamente para que si se desea modificar un inventario no se no se pueda realizar la acción sin antes seleccionar el inventario que se desea modificar.

Registro de defectos y dificultades detectados

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
<Crear>	< 1>	<Creado inventario con letras en los campos OD y Longitud >	Cuando se introducen los datos para crear un inventario, si los datos de los campos OD y Longitud son letras, el sistema crea el inventario y cambia las letras a números, cuando no debería crearlo pues los datos entrados no son validos.	<Prueba>	<Alta>	<Validar para que el sistema no cree un inventario con datos incorrectos para los campos OD y Longitud >
<Listar y Modificar >	< 2>	<Modificado inventario con datos vacíos >	Cuando se dejan los campos del inventario vacío y se da clic en el botón enviar el sistema envía los datos y actualiza los datos del inventario con los datos vacíos.	<Prueba>	<Alta>	<Validar para que el sistema no envié los datos vacíos cuando se dé clic en el botón enviar >

Glosario de términos

Centro de Investigaciones del Petróleo (CEIPET): instituto dedicado a la investigación aplicada en la Industria del Petróleo Cubana, y al desarrollo de programas y proyectos de investigación, servicios científico - técnicos y producciones especializadas.

Centro de Desarrollo de Informática Industrial (CEDIN): Centro generador de soluciones integrales, desarrollo de tecnologías, productos y servicios asociados a la Informática Industrial. Contribuye a la formación especializada y al desarrollo de investigaciones afines que garanticen un alto valor agregado.

Computadora Personal (PC): microcomputadora diseñada para ser usada por una sola persona a la vez, generalmente de tamaño medio y suele estar equipada para cumplir tareas comunes de la informática moderna, es decir permite navegar por Internet, escribir textos y realizar otros trabajos de oficina además de escuchar música, ver videos, jugar, estudiar; y en cuanto a su movilidad podemos distinguir entre computadora de escritorio y computadora portátil.

Cuello de botella (Bottleneck): Límite en la capacidad de transferencia de información de un sistema que puede reducir el tráfico en condiciones de sobrecarga. Suele producir una baja del rendimiento y la velocidad general tanto en un sistema como en una conexión. Los cuellos botella (llamados restricciones) condicionan la salida de toda la producción.

Dirección de Intervención y Perforación de Pozos (DIPP): entidad que dirige la perforación e intervención de pozos de petróleo, controla todas las operaciones y demás actividades realizadas en los pozos en perforación e intervención.

Empresa de Producción y Extracción de Petróleo del CENTRO (EPEPC): entidad con la finalidad principal de regular la explotación de los yacimientos del centro del país.

Entorno de Desarrollo Integrado (IDE): es un programa que agrupa un conjunto de herramientas que viabilizan la labor de los programadores.

Entorno Virtual de Aprendizaje (EVA): entorno de aprendizaje que ofrece servicios con el objetivo de aumentar la comunicación y la preparación del personal que lo utiliza

Ingeniería de sistemas asistida por ordenador (CASE): aplicación de tecnología informática a las metodologías propias de desarrollo de sistemas con el objetivo de automatizar una o más fases del ciclo de vida del desarrollo de software.

Industria Cubana del Software (ICS): industria cubana encargada de la producción de software de alta calidad en prestaciones, imagen y soporte para satisfacer las necesidades nacionales e internacionales de diversos sectores.

Lenguaje de marcado extensible (XML): lenguaje de marcas que ofrece un formato para la descripción de datos estructurados, permitiendo definir nuestro propio lenguaje de presentación y, a diferencia del HTML, que se centra en la representación de la información, XML se centra en la información en sí misma.

Lenguaje de marcación de hipertexto (HTML): lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, actualmente este lenguaje se ha convertido en uno de los formatos más populares que existen para la construcción de documentos.

Proceso Unificado de Rational (RUP): metodología de desarrollo de software que captura varias de las *mejores prácticas* en el desarrollo moderno de software que garantiza a cada miembro de un equipo un fácil acceso a una base de conocimiento con guías, plantillas y herramientas para todas las actividades críticas de desarrollo.

Stakeholders: Personas u organizaciones que están activamente implicadas en el negocio ya sea porque participan en él o porque sus intereses se ven afectados con los resultados del proyecto. Pueden ser los propietarios, la dirección, quienes financian, los clientes, los trabajadores, los proveedores, la competencia, la comunidad local, etc.

Unidad Central de Procesamiento (CPU): parte de una computadora que realiza el procesamiento de la información.

Universidad de las Ciencias Informáticas (UCI): centro universitario surgido al calor de la Batalla de Ideas, en el que se vincula docencia y producción, vanguardia en la producción de software a nivel nacional.