Universidad de las Ciencias Informáticas

Facultad 5

Título: "Automatización de las pruebas de interfaz de usuario para componente HMI GTK del SCADA"**.**

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autores: René Iván Rodríguez Infante
Wilsón Alfonso González

Tutores: Ing. Amado Espinosa Hidalgo
Ing. Mariela Cepero

Ciudad de La Habana, Junio 2010

DECLARACIÓN DE AUTORÍA

Declaramos ser 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 firmo la presente a los ____ días del mes de _____ del año _____.

René Iván Rodríguez Infante.

_____
Firma del Autor

Amado Espinosa Hidalgo

_____
Firma del Tutor

Wilsón Alfonso González.

_____
Firma del Autor

Mariela Cepero Nuñez

_____
Firma del Tutor

Datos de Contacto

Amado Espinosa Hidalgo: Ingeniero Informático y profesor asistente del Departamento de Ingeniería y Gestión de Software de la Facultad 5. Posee 7 años de experiencia en la actividad docente y productiva. Correo electrónico: aespinosa@uci.cu.

Mariela Cepero Núñez: Ingeniera en Ciencias Informáticas perteneciente al Departamento Integración y Despliegue del Centro de Desarrollo de Informática Industrial. Profesor instructor con 3 años de experiencia docente y 4 años de experiencia en la producción.

Agradecimientos

A mis padres por ser ejemplo para mí en todo los aspectos de la vida. En especial a mi madre que ha sido el motor impulsor de todos mis triunfos. A mis hermanos (Renier y Yiselle) por su apoyo incondicional, mis abuelos que desde niño me han mostrado el camino correcto en la vida por su experiencia. Mi familia que nos mantiene unidos día a día y cuando alguno de sus integrantes triunfa es una victoria de todos. A todos los profesores que han contribuido con mi formación profesional y decirle a todos que sin ellos no hubiera podido llegar hasta donde estoy hoy.

Mis amigos también toman un papel protagónico a la hora de los agradecimientos, los que vienen desde el pre-universitario que más que amigos los considero como hermanos, también a los nuevos que he conocido aquí en la Universidad y me han brindado su amistad incondicional. Les quiero agradecer a todos los que de una forma u otra han contribuido con que este sueño se haya hecho realidad. Gracias.

René Iván.

A mi familia que siempre me ha apoyado en todo, y que mucho de lo que soy se lo debo a ellos, en especial a mi madre que desgraciadamente no está pero igual esta victoria se la dedico especialmente a ella, a mi padre que en todo este tiempo ha sido la figura que me ha mantenido firme y que ha luchado incansablemente para que me pueda graduar de ingeniero, gracias papá de verdad parte de esto también es tuyo para mí nos graduamos los dos.

A Olguita que eres la mejor madrastra del mundo sin que me quede nada por dentro te quiero mucho, a mi hermana Yanny, a mi novia que me ha ayudado en todo.

A mis abuelos que también me han ayudado mucho son los mejores del mundo, en especial a mi abuela Zenaida que ha sido una madre para mí, a mis tíos que los quiero con la vida a todos, me han dado muy buenos consejos de verdad se los agradezco mi hicieron mucha falta. A mis primos gracias por todo. A todos mis amigos que son muchos, gracias por todo. A los inolvidables amigos de la universidad no voy a mencionar ninguno porque son tantos que haría falta mucho espacio para mencionarlos a todos, los quiero mucho a todos, gracias a todos los profesores que han contribuido con mi formación profesional y decirle a todos que sin ellos no hubiera podido llegar hasta donde estoy hoy.

Wilsón Alfonso González

Dedicatoria

A nuestros padres,
por ser ellos la inspiración de este trabajo.

Resumen

Esta investigación propone el uso de la herramienta LDTP (Linux Desktop Testing Project) para su utilización dentro del Centro de Desarrollo de Informática Industrial (CEDIN). Se describen brevemente aspectos vinculados con este tema como los tipos, niveles de prueba y características de valoración de los resultados.

Se realiza una comparación entre un grupo de herramientas que son utilizadas a nivel mundial en el proceso de ejecución de pruebas automatizadas, se justifica la utilización de la herramienta propuesta como herramienta para la automatización del proceso de pruebas en los proyectos productivos del CEDIN. Se realiza una descripción en forma de manual de usuario de la herramienta para que las personas interesadas en aprender a trabajar con dicha herramienta tengan la posibilidad de hacerlo mediante este trabajo.

Se aplica el proceso de pruebas con la herramienta al SCADA Guardián del Alba del Centro de Desarrollo de Informática Industrial, quedando esta como un ejemplo más de la utilización de estas nuevas herramientas. Por último se estableció una comparación entre las pruebas manuales y las pruebas automatizadas obteniendo significativos resultados en cuanto a ahorro de tiempo y recursos.

PALABRAS CLAVES

Pruebas automatizadas, herramientas para automatización de pruebas, LDTP (Linux Desktop Testing Project).

Índice

Índice de Tabla

Introducción

Las pruebas de Software son los procesos que permiten verificar y revelar la calidad de un producto. Básicamente es una fase en el desarrollo del software que consiste en probar las aplicaciones construidas para identificar posibles fallos de implementación, calidad o usabilidad de un software. Hay muchos criterios a la hora de abordar el proceso de pruebas de software, pero para verificar productos complejos de forma efectiva se requiere de un proceso de investigación, más que seguir un procedimiento al pie de la letra. Una definición de "testing" es: proceso de evaluación de un producto desde un punto de vista crítico, donde el "tester" (persona que realiza las pruebas) somete el producto a una serie de acciones, y el producto responde con su comportamiento como reacción. (Pressman, 2002).

Verificar el buen funcionamiento de un software aplicándole los diferentes tipos de prueba es una acción que lleva implícito un conjunto de pasos y formalidades para demostrar la fiabilidad del proceso, existen diferentes metodologías para realizar pruebas aunque todas ellas en síntesis convergen a: definir un plan de pruebas, determinar los casos de prueba, aplicar estos casos de prueba y analizar los resultados obtenidos de las pruebas.

De acuerdo a lo que se desee probar las pruebas van a tomar diferentes clasificaciones. Se encuentran las pruebas de caja blanca, estrés, carga y rendimiento, validación, aceptación, caja negra, etc. Las pruebas de caja negra o las llamadas Pruebas de Sistema se centran en analizar dado un conjunto de datos de entrada la respuesta de la aplicación, es decir no interviene en el funcionamiento interno del software.

Comúnmente las pruebas de Software son llevadas a cabo por humanos de forma manual, lo que dificulta la cobertura de una cantidad importante de condiciones que permiten descubrir errores. En la actualidad existen un conjunto de herramientas para realizar todo este proceso de prueba de forma automática lo cual brinda facilidades sustanciales al trabajo y hace que este sea más completo, podríamos mencionar dentro de estas herramientas a:

- BuilBot.
- TinderBox.
- Deajagnu.
- FunkLoad.

- HammrrdHead.
- GNU/LDTP.
- CPPtest.
- OpenLoad.

En el Centro de Desarrollo de Informática Industrial, el proyecto SCADA desarrolla el componente HMI GTK+ para todo lo referente a la interfaz de usuario por su alta demanda tanto para clientes nacionales como extranjeros, al igual que los demás componentes del SCADA, este debe pasar por un proceso de prueba que garantice la calidad de sus productos, dicho proceso es desarrollado por el grupo de prueba del centro de forma manual, elevando el esfuerzo de este equipo de trabajo, los costos de producción, el tiempo de entrega y los recursos, además de la limitación al desarrollo de pruebas de un alto nivel de complejidad que permitan identificar errores de mayor magnitud.

Después de analizar la **situación problémica** existente se define como **problema científico** ¿Cómo automatizar las pruebas para la interfaz de usuario del HMI GTK del SCADA? Por tanto se define como **objeto de estudio** proceso de pruebas de software. Para proseguir la investigación en vista a resolver el problema planteado se propone como **idea a defender:** con la utilización de una herramienta para desarrollar pruebas automáticas de software al componente HMI GTK del SCADA se lograrán ahorros sustanciales en tiempo y en recursos además de una mayor veracidad en la ejecución de las pruebas.

Para el desarrollo de la investigación nos trazamos los siguientes objetivos:

Objetivo general

Aplicar una herramienta para hacer pruebas automatizadas de software al componente HMI GTK del SCADA.

Se define como **campo de acción** herramientas para pruebas automatizadas de interfaz de usuario.

Tareas de la investigación:

1. El estudio del arte centrado en pruebas de software y herramientas para la automatización de pruebas de interfaz de usuarios para aplicaciones desktop.

2. Realización de un informe que fundamente la propuesta, donde se describan las herramientas candidatas, principales funcionalidades y características.

3. Prueba de la herramienta seleccionada a través de un ejemplo sencillo del componente HMI GTK con el objetivo de demostrar el correcto funcionamiento de la misma.

4. Desarrollo de informe o manual de usuario para facilitar el uso de la herramienta en otros proyectos del Centro de Desarrollo de Informática Industrial.

5. Elaboración del plan de pruebas definiendo el alcance de la solución.

6. Diseño de los casos de pruebas para su posterior incorporación a la herramienta seleccionada.

7. Incorporación de los casos de pruebas a la herramienta seleccionada comenzando con la solución práctica del problema.

8. Ejecución de las pruebas para continuar con el desarrollo de la solución.

9. Análisis de los resultados de las pruebas para demostrar las ventajas de utilizar herramientas de pruebas automatizadas.

Para el cumplimiento de estos objetivos se llevan a cabo varios métodos y técnicas en la búsqueda y procesamiento de la información, estos se clasifican en niveles: teóricos y empíricos. Dentro de los teóricos y poniendo en práctica los métodos de análisis-síntesis se realizó un estudio exhaustivo de los diferentes tipos de pruebas así como de las herramientas para pruebas automatizadas a aplicaciones de escritorio. Analizando la documentación de lo anteriormente expuesto y sintetizando con los rasgos más relevantes de acuerdo a la investigación. Además de los métodos de análisis históricos-lógico donde se analizó la historia de las herramientas automatizadas así como sus principales características.

En los empíricos se utilizó la entrevista para obtener información a partir de conversaciones planificadas con especialistas del Centro de Calidad para Soluciones Tecnológicas (CALISOFT) de la UCI.

1.1 Capítulo 1: Fundamentación teórica.

1.2 Introducción.

No importa lo avanzado que sea el código y lo flexible que sean los sistemas. Si una aplicación no es funcional, es inútil. Es por esto que las pruebas funcionales son consideradas como la parte más importante del desarrollo.

Las pruebas funcionales están desarrolladas bajo la perspectiva del usuario, confirmando que el sistema hace lo que los usuarios esperan que haga. Un error funcional en su aplicación puede tener consecuencias catastróficas, desde la no credibilidad de sus clientes, hasta grandes pérdidas económicas.

Existe una gran variedad de herramientas automatizadas de pruebas y frameworks disponibles. Esto junto con las nuevas metodologías en desarrollo de aplicaciones, han creado un reto a la hora de diseñar frameworks de pruebas que sean mantenibles.

1.2 Prueba de software.

Bajo el nombre de pruebas de software se agrupan un conjunto de prácticas correctivas (frente a las prácticas preventivas que se aplican durante el proceso de construcción de software) cuyo objetivo es determinar la calidad de los sistemas software. (Pressman., 2002.).

La prueba es el proceso de ejecución de un programa con la intención de descubrir un error (Pressman, 2002).

Según lo anteriormente planteado las pruebas de software son la actividad más común de control de calidad realizada en los proyectos de desarrollo o mantenimiento de aplicaciones y sistemas. El éxito de una prueba esta dado por la cantidad de errores que le sean detectados al software.

1.2.1 Objetivos de las pruebas.

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

- Diseñar e implementar las pruebas cuando los casos de pruebas que especifican qué probar, creando los procedimientos de pruebas que especifican como realizar las pruebas y creando si es posible componentes de pruebas ejecutables para automatizar las pruebas.

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

Según Pressman: "Las pruebas deberían empezar por lo pequeño y progresar hacia lo grande".
(Pressman., Ingenieria del Software. Un enfoque práctico., 2002.)

1.2.2 Tipos de pruebas de software.

Para saber que un sistema una vez lanzado funcionará correctamente, es necesario realizar una comprobación crítica del mismo. Muchos grupos de trabajo se limitan a dedicar una o dos personas a probar el sistema. Para lograr una mayor calidad en el sistema se llevan a cabo las **Pruebas de Unidad** durante la fase de construcción, específicamente en el flujo de trabajo de implementación; las cuales se basan en probar los componentes implementados como unidades individuales.

Aplicar pruebas de unidad a un sistema existente será difícil. Si el sistema es mediano o grande, es conveniente planear sus pruebas y aplicar los cambios necesarios a través de varias versiones futuras.

Las pruebas unitarias aíslan cada parte del programa y muestran que las partes individuales son correctas. A pesar de esto no descubrirán todos los errores del código, no descubren errores de integración, de rendimiento ni otros problemas que afectan a todo el sistema en su conjunto.

Las pruebas de unidad están divididas en dos grupos, en Pruebas de Caja Blanca y Pruebas de Caja Negra.

En las **pruebas de Caja Blanca** se observa siempre el código, las mismas se realizan para probarlo todo. Estas pruebas se aplican mediante diferentes métodos.

- ✓ **Prueba de Condición**: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.

- ✓ **Prueba de Flujo de Datos**: Es un método en el que se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- ✓ **Prueba de Bucles**: Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.
- ✓ **Prueba del Camino Básico**: Esta permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

Según RUP, las pruebas de Caja Blanca verifican la implementación interna de la unidad. Para cada componente se estudiará su implementación interna y se tratará de verificar su correcto comportamiento algorítmico. Se comprobarán los caminos comunes, los críticos, los menos conocidos y otros asociados con altos riesgos.

Lograr un buen método con pruebas de caja blanca es un objetivo deseable pero no suficiente a todos los efectos. Un programa puede estar perfecto en todos sus términos y sin embargo no servir a la función que se pretende.

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.

Las pruebas de caja negra están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario (ejemplo: canales de comunicaciones, etc.).

Según RUP estas pruebas verifican el comportamiento de la unidad observable externamente. Cuando el número de entradas y salidas es grande, se dividen estas en clases de equivalencias. Una clase de equivalencia es un conjunto de valores de entradas o salidas para los que se supone que un componente se comporta de forma similar.

Durante la lectura de los requisitos del sistema, se pueden encontrar una serie de valores singulares que marcan diferencias de comportamiento. Estos valores son claros candidatos a marcar clases de equivalencia: por abajo y por arriba.

Una vez identificadas las clases de equivalencia significativas en dicho módulo, se procede a coger un valor de cada clase, que no esté justamente al límite de la clase. Este valor aleatorio, sustituye a cualquier valor normal que se le pueda pasar en la ejecución real.

Al realizar pruebas funcionales lo que se pretende es ponerse en los pies del usuario, usar el sistema como él lo usaría, sin embargo el analista de pruebas debe ir más allá que cualquier usuario, generalmente se requiere apoyo de los usuarios finales ya que ellos pueden aportar mucho en el desarrollo de casos de prueba complejos, enfocados básicamente al negocio, posibles particularidades que no se hayan contemplado adecuadamente en el diseño funcional.

Otro tipo de prueba que se llevan a cabo durante esta fase (Construcción) son las **Pruebas de Integración**, las mismas involucran a un número creciente de módulos y terminan probando el sistema como conjunto.

Estas pruebas se pueden plantear desde un punto de vista estructural o funcional. Las pruebas estructurales de integración son similares a las pruebas de caja blanca pero trabajan a un nivel conceptual superior. No se refieren a sentencias del lenguaje sino a llamadas entre módulos. Se trata pues de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura de segmentos o de ramas.

Las pruebas funcionales de integración son similares a las pruebas de caja negra. Con estas se trata de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). Según se van acercando al sistema total, estas pruebas se basan cada vez más en la especificación de requisitos del usuario.

En todas estas pruebas funcionales se siguen utilizando las técnicas de partición en clases de equivalencia y análisis de casos límite (fronteras).

Las **Pruebas del Sistema** propuestas por la Metodología RUP, prueban que el sistema funciona globalmente de forma correcta. Cada prueba del sistema prueba combinaciones de casos de uso bajo condiciones diferentes. Se prueba el sistema como un todo probando casos de uso unos detrás de otros y si es posible, en paralelo.

Cuando se construye software a la medida para un cliente, se llevan a cabo una serie de **Pruebas de Aceptación** para permitir que el cliente valide y verifique todos los requisitos pactados. Estas pruebas las realiza el usuario final en lugar del responsable del desarrollo del sistema.

El cliente es quien impone los requisitos, quien mejor que él para dar fe de su satisfacción, además no se debe dejar de la mano la existencia de la calidad percibida que por cruel que parezca, determina en cómo el software será aceptado.

1.2.3 Artefactos generados en el flujo de trabajo Prueba.

Un sistema de prueba está compuesto por un Modelo de prueba y este a su vez contiene a uno o muchos:

1. Casos de prueba.
2. Plan de Prueba.
3. Procedimiento de prueba.
4. Componente de prueba.

Ver Anexo 1.

Un **caso de prueba** especifica una forma de probar el sistema, incluyendo la entrada o resultado con el que se ha de probar y las condiciones bajo las que ha de probarse ver Anexo 2.

En la práctica, lo que se prueba puede venir dado por un requisito o colección de requisitos del sistema cuya implementación justifica una prueba que es posible realizar y que no es demasiado costoso. Los siguientes son casos de prueba comunes:

- ✓ Un caso de prueba que especifica cómo probar un caso de uso o un escenario específico de un caso de uso. Un caso de prueba de este tipo incluye la verificación del resultado de la interacción entre los actores y el sistema, que se satisfacen las precondiciones y las poscondiciones especificadas por el caso de uso y que se sigue la secuencia de acciones especificadas por el caso de uso. Obsérvese que un caso de prueba basado en un caso de uso especifica típicamente una prueba del sistema como “caja negra”, es decir una prueba del comportamiento observable externamente del sistema.

- ✓ Un caso de prueba especifica cómo probar una realización de caso de uso-diseño o un escenario específico de la realización. Un caso de prueba de este tipo puede incluir la verificación de la interacción entre los componentes que implementan dicho caso de uso. Obsérvese que los casos de prueba basados en una realización de caso de uso basado típicamente especifican una prueba del

sistema como “caja blanca”, es decir, una prueba de la interacción interna entre los componentes del sistema.

Algunos casos de uso pueden ser parecidos y diferenciarse únicamente en un solo valor de entrada o resultado. Esto se da a veces para casos de prueba que verifican diferentes escenarios del mismo caso de uso. En estos casos, puede ser apropiado especificar los casos de uso en forma de tabla, donde cada caso de prueba está representado por una fila y cada rango de valores de entrada y de salida se representa con una columna.

Se pueden especificar otros casos de prueba para probar el sistema como un todo. Por ejemplo:

- ✓ Las pruebas de instalación verifican que el sistema puede ser instalado en la plataforma del cliente y que el sistema funcionará correctamente cuando sea instalado.
- ✓ Las pruebas de configuración verifican que el sistema funciona correctamente en diferentes configuraciones: por ejemplo en diferentes configuraciones de red.
- ✓ Las pruebas negativas intentan provocar que el sistema falle para poder así revelar sus debilidades.
- ✓ Las pruebas de tensión o de estrés identifican problemas con el sistema cuando hay recursos insuficientes o cuando hay competencia por los recursos.

Un **procedimiento de prueba** especifica cómo realizar uno o varios casos de prueba o parte de éstos. Se puede manejar el término procedimiento como:

- ✓ Una instrucción para un individuo sobre cómo ha de realizar un caso de prueba manualmente.
- ✓ Una especificación de cómo interaccionar manualmente con una herramienta de automatización de pruebas para crear componentes ejecutables de prueba.

Un procedimiento de prueba especifica cómo llevar a cabo un caso de prueba, pero a menudo es conveniente utilizar un mismo procedimiento para varios casos de prueba. Ver Anexo 3.

Un **componente de prueba** automatiza uno o varios procedimientos de pruebas o partes de ellos. Dichos componentes pueden ser desarrollados utilizando un lenguaje de guiones o un lenguaje de programación, o pueden ser grabados con una herramienta de automatización de pruebas. Ver Anexo 4.

Estos se utilizan también para probar los componentes en el modelo de implementación, proporcionando entradas de pruebas, controlando y monitorizando la ejecución de los componentes a probar y, posiblemente, informando de los resultados de las pruebas.

En el **plan de pruebas** se describe las estrategias, recursos y planificación de la prueba. La estrategia de prueba define el tipo de prueba a realizar para cada iteración y sus objetivos.

Un **defecto** es una anomalía del sistema, como por ejemplo un síntoma de un fallo software o un problema descubierto en una revisión. Un defecto puede ser utilizado para localizar cualquier cosa que los desarrolladores necesitan registrar como síntoma de un problema en el sistema que ellos necesitan controlar y resolver.

Una **evaluación de prueba** es una evaluación de los resultados de los esfuerzos de prueba, tales como la cobertura del caso de prueba, la cobertura de código y estados de los defectos.

1.2.4 Pruebas Automatizadas.

Los software de automatización son utilizado para controlar la ejecución de pruebas, comparación de resultados, preparación de precondiciones y realización de informes. Las pruebas automatizadas son efectivas en entornos donde los cambios son frecuentes o en aplicaciones en las que se esperan builds y releases críticos. Es por ello que es necesario destinar la automatización de pruebas a un equipo con experiencia en el sector.

1.2.5 Pruebas Manuales.

Las pruebas manuales son el método de pruebas de software más antiguo y riguroso. Requieren a un tester que ejecute de manera manual operaciones en la aplicación sin ayuda de herramientas de automatización. Las pruebas manuales requieren que el tester sea paciente, observador, creativo e innovador.

Las pruebas manuales ayudarán a descubrir cualquier problema relacionado con la funcionalidad de su producto, especialmente defectos relacionados con la usabilidad y el interfaz gráfico de su aplicación.

Las herramientas de automatización ejecutan únicamente scripts diseñados para testear una requisito o funcionalidad específicos y no poseen la habilidad de "toma de decisiones" ni grabación de discrepancias no incluidas en el script. Es por ello que es necesario realizar pruebas manuales a lo largo de toda la aplicación, antes de realizar la automatización de actividades.

1.2 .6 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 software.

Estas te van guiando paso a paso sobre las actividades que debes hacer para alcanzar el producto informático deseado, indicando además que personas deben participar en el desarrollo de las actividades y qué papel debe de tener. Detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

1.2.7 Metodología RUP.

El proceso de software propuesto por RUP cuenta con tres características esenciales: está dirigido por Casos de Uso, centrado en la arquitectura y es iterativo e incremental.

Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema. (Juristo,, 2003).

En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo.

En el caso de RUP además de utilizar los Casos de Uso para guiar el proceso, presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento.

Una iteración puede realizarse por medio de una cascada como se muestra en el Anexo 5. Se pasa por los flujos fundamentales (Requisitos, Análisis, Diseño, Implementación y Pruebas). También existe una planificación de la iteración, un análisis de la iteración y algunas actividades específicas de la iteración. Al finalizar se realiza una integración de los resultados con lo obtenido de las iteraciones anteriores.

RUP divide el proceso en cuatro fases (Inicio, Elaboración, Construcción y Transición), dentro de las cuales se realizan varias iteraciones en número variable según el proyecto. Ver Anexo 6.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos y al establecimiento de una línea base de la arquitectura. Durante la fase de inicio las iteraciones ponen mayor énfasis en actividades de captura de requisitos y modelado del negocio.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones. Para cada iteración se seleccionan los Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realizan tantas iteraciones como sean necesarias hasta que se termine con la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

En cada fase participan todas las disciplinas, pero dependiendo de la fase, el esfuerzo dedicado a una disciplina varía.

Además RUP propone nueve flujos de trabajo los cuales combinados con las cuatro fases explicadas anteriormente le dan culminación al todo el proceso de desarrollo del software estos flujos de trabajo son: Modelado de negocio, Requisitos, Análisis y Diseño, Implementación, Prueba, Despliegue, Configuración y manejo del cambio, Administración del proyecto y Entorno. Para ver gráficamente la combinación fase/flujos Anexo 7.

Uno de los flujos de trabajo que contempla RUP es el relacionado con las pruebas de software. En él se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema entregadas a terceros.

Las pruebas, vistas desde el marco de un proceso de desarrollo de software, son los diferentes procesos que se deben realizan durante un desarrollo, con el objetivo de asegurar completitud, correctitud, calidad, entre otros factores de gran importancia.

Estas, al contrario de lo que muchas personas creen, no se deben dejar para el final de la etapa de construcción del software. Las pruebas se deben empezar a realizar desde la misma etapa de análisis y Diseño, ya que desde un principio se puede incurrirse en malas interpretaciones de las "reglas del negocio", lo que finalmente tendrá como consecuencia incongruente entre lo que el cliente quiere y lo que se ha desarrollado.

Según RUP, existen cuatro niveles de prueba: unidad, integración, sistema y aceptación. Un software se va perfeccionando mediante las iteraciones que se van realizando durante su ciclo de vida. Durante este proceso, las pruebas de software son fundamentales, permitiendo un fuerte acercamiento a los requerimientos establecidos con el cliente. En cada iteración, el equipo de desarrollo obtiene un resultado, siendo éste el principal candidato para las pruebas.

1.2.8 Roles de pruebas según RUP.

Un rol define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. (Kruchten, 2000)

RUP propone fundamentalmente 4 trabajadores para el Flujo de trabajo de Pruebas.

El **diseñador o ingeniero de prueba** es responsable de la integridad del modelo de prueba, asegurando que el modelo cumple con su propósito. Los diseñadores de pruebas también planean las pruebas, lo que significa que deciden los objetivos de pruebas apropiados y la planificación de las pruebas. Además seleccionan y describen los casos de prueba y los procedimientos de prueba correspondientes que se necesitan, y son responsables de la evaluación de las pruebas de integración y de sistemas cuando estas se ejecutan. Véase anexo 8.

Los diseñadores de prueba realmente no llevan a cabo las pruebas, sino que se dedican a la preparación y evaluación de las mismas.

El **ingeniero de componentes** son responsables de los componentes de pruebas que automatizan algunos procedimientos de prueba (no todos los procedimientos de prueba pueden ser automatizados).Esto es así porque la creación de dichos componentes puede necesitar de substanciales habilidades como programador.

El i**ngeniero de pruebas de integración** son los responsables de realizar las pruebas de integración que se necesitan para cada construcción producida en el flujo de trabajo de la implementación. Las pruebas de integración se realizan para verificar que los componentes integrados en una construcción funcionan

correctamente juntos. Por esto las pruebas de integración se derivan a menudo de los casos de prueba que especifican cómo probar realizaciones de casos de uso-diseño. Además es el encargado de documentar los defectos en los resultados las pruebas de integración.

El **ingeniero de pruebas de sistemas** es responsable de realizar las pruebas de sistemas necesarias sobre una construcción que muestra el resultado (ejecutable) de una iteración completa. Dichas pruebas se llevan a cabo principalmente para verificar las interacciones entre los actores y el sistema. Por esto, las pruebas de sistemas se derivan a menudo de los casos de prueba que especifican cómo probar los casos de usos. Aunque también se aplican otros tipos de pruebas al sistema como un todo. Este se encarga además de documentar los defectos de los resultados de las pruebas. Debido a su naturaleza, los individuos que desempeñan dicho rol no necesitan saber mucho sobre el funcionamiento interno del sistema. Por el contrario, éstos deberían tener familiaridad con el comportamiento observable extremamente del sistema. Por tanto, algunas de estas pruebas pueden ser realizadas por otros miembros del proyecto, como los especificadores de casos de uso, o incluso por personas externas al proyecto, como usuarios de versiones betas.

1.3 Herramientas para pruebas automatizadas de software.

Cualquier programador con experiencia en el desarrollo de aplicaciones web conoce de sobra el esfuerzo que supone probar correctamente la aplicación. Crear casos de prueba, ejecutarlos y analizar sus resultados es una tarea tediosa. Además, es habitual que los requisitos de la aplicación varíen constantemente, con el consiguiente aumento del número de versiones de la aplicación y la refactorización continua del código. En este contexto, es muy probable que aparezcan nuevos errores.

Este es el motivo por el que la automatización de pruebas es una recomendación útil para crear un entorno de desarrollo satisfactorio. Los conjuntos de casos de prueba garantizan que la aplicación hace lo que se supone que debe hacer. Incluso cuando el código interno de la aplicación cambia constantemente, las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación. Además, este tipo de pruebas obligan a los programadores a crear pruebas en un formato estandarizado y muy rígido que pueda ser procesado por un framework de pruebas.

En ocasiones, las pruebas automatizadas pueden reemplazar la documentación técnica de la aplicación, ya que ilustran de forma clara su funcionamiento. Un buen conjunto de pruebas muestra la salida que produce el software para una serie de entradas de prueba, por lo que es suficiente para entender el propósito de cada método.

En la actualidad existen un conjunto de herramientas automatizadas para realizar pruebas de software. Las cuales brindan soporte a las pruebas de integración, pruebas de sistema, diagnóstico o afinado de las aplicaciones en los entornos de preproducción o certificación y los de producción, que generalmente son difíciles de realizar de una forma integrada, comprenden herramientas de análisis estático, automatización de pruebas funcionales, de carga, de rendimiento, de estrés,el afinado del código, etc. Dichas herramientas de acuerdo al objetivo que persiguen tienen características propias que las hacen diferenciar unas de otras. A continuación se muestran una serie de herramientas con sus principales características.

1.3.1 Herramientas en Windows.

RoutineBot es un software de comprobación de automatización de la interfaz. Este permite crear scripts para probar la interfaz de sistemas de software tal como los probadores humanos lo harían. RoutineBot simula los clics del ratón o de botones, permite al usuario especificar una muestra de imagen a ser encontrada en la pantalla y ejecuta cierta acción con esta muestra. Por ejemplo, puede encontrar la imagen de prueba y entonces situar el ratón encima de ella o moverlo a una posición relativa a ella. El programa es representado en dos módulos - uno es el diseñador de RoutineBot y otro es el intérprete de RoutineBot. El diseñador de RoutineBot ayuda al usuario a diseñar los scripts de comprobación, mientras que el intérprete permite ejecutar estos scripts. Además fue diseñado para probar la interfaz visual del software. Puede correr en un lote y procesar un conjunto de pruebas.

RoutineBot permite simular acciones que el usuario normalmente hace:

- ✓ Ejecutar y terminar aplicaciones
- ✓ Situar el ratón sobre una imagen de muestra.
- ✓ Simular el clic, doble clic, y los eventos de presionar (MouseDown) y soltar (MouseUp) el botón del ratón.
- ✓ Mover el ratón a unas coordenadas absolutas o relativas.
- ✓ Entrar un texto o simular la opresión de teclas en el teclado.

DwgPlotX es una herramienta para trazar por lotes y procesar los archivos de AutoCAD, que está diseñada para hacer de tus sesiones de AutoCAD más rápidas y más productivas. Cuenta con un plotter, un exportador y un reemplazador de cadenas para la versión de AutoCAD completa. Cuenta con un comando de AutoCAD que captura la secuencia de comandos, pasando por un depurador de secuencias,

una prueba de ejecución, y también en paquetes de un cambiador de nombre de archivo y programador. (Compatible con AutoCAD LT y la versión completa). Dispone de una interfaz fácil de usar con búsqueda de sub-carpetas, selección múltiple de carpetas, filtro de archivos y administrador de listado de archivos para permitirle elegir planos de muchas maneras. Archivos y carpetas pueden arrastrarse y soltarse en la vista del archivo e importantes carpetas pueden llamarse y renombrarse en Favoritos.

Para la versión completa, DwgPlotX cuenta con un trazador por lotes de secuencia de comandos totalmente gratuito que opera tanto con archivos dwg como dxf. Todos los parámetros de trazado como plotter, tamaño del papel, estilo de trazo, tabla, orientación del trazado y el margen de impresión son explorados automáticamente por el sistema y estarán disponibles para que el usuario elija - al igual que en AutoCAD. La parcela de impresión (Visualización / Extensión / Límites) y escala de trazado también se puede seleccionar desde la interfaz. También puedes elegir el tipo de disposición del trazado, el número de veces por archivo que va a imprimirse y el número de veces que el lote completo va a repetirse.

SQA4.Net.Tester puede grabar de forma instantánea los eventos de un control desde cualquier aplicación Win32, .Net, WPF. Durante una sesión de grabación, todos los eventos desde los controles son completamente grabados como casos de test. Tan pronto como la grabación está terminada, los casos de prueba grabados son completamente generados y están listos para su ejecución. Los casos de test ya predefinidos pueden ser ejecutados sin la necesidad de modificación alguna.

SQA4. Net.Tester además proporciona completas capacidades de autoría de casos de prueba. Los desarrolladores pueden crear casos de prueba no grabados con un número mínimo de pulsaciones de teclas y clic de ratón.

Los resultados detallados de los tests son generados en formato XML, el cual puede ser exportado a formatos Microsoft Word (DOC), Microsoft Excel (XLS), página web (HTM), ideal para crear manuales, guías de usuario, material técnico, y otros tipos de copias de seguridad de documentación o material de referencia.

vTest es una herramienta automática para pruebas funcionales y de regresión para aplicaciones web Incorpora capacidades de registro, verificación, reproducción y reporte. Soporta tanto Microsoft Internet Explorer como Mozilla Firefox. Soporta todas las tecnologías populares como HTTP, HTTPS/SSL, JavaScript, DHTML, ActiveX, Java & Flash. vTest también soporta todos los principales entornos de aplicaciones web incluyendo ASP, ASP.NET, Java Servlets y JSP, PHP y CGI. vTest le permite registrar y probar su aplicación completa en minutos simplemente apuntando y haciendo clic sobre su aplicación Web. Sin ninguna programación, todos los objetos son capturados y registrados automáticamente como

un programa gráfico que muestra los pasos en su programa en forma de un árbol en basado en iconos. Para aquellos usuarios que desean usar programación, usa JavaScript como lenguaje de programación y provee de una API JavaScript comprensiva que puede utilizar para crear programas de prueba personalizados. Crea los programas que se adaptan muy bien cuando la interfaz de usuario de su aplicación cambia. La mayor parte de la competencia crea programas que simplemente usan el objeto name/ID para identificar un objeto en su aplicación. Usa un algoritmo de identificación de objeto sofisticado para identificar los objetos en su aplicación. Esto permite que los programas de vTest se adapten muy bien a los cambios de la interfaz de usuario de su aplicación.

Beneficios:

- ✓ Pruebas automáticas de tareas de funcionalidad y regresión y así se reduce el plazo de comercialización.
- ✓ Mejora la eficiencia de los ingenieros de aseguramiento de calidad permitiéndoles dedicar menos tiempo a pruebas manuales y más tiempo para probar el diseño.
- ✓ Maximiza la confiabilidad y la robustez proporcionando la capacidad de ejecutar pruebas bajo demanda o según un programa determinado sin necesidad de ninguna intervención del usuario.
- ✓ No requiere una programación de fondo. Para aquellos usuarios que desean usar sus habilidades de programación, vTest usa el estándar de la industria, el lenguaje JavaScript
- ✓ Administra la evolución del producto efectivamente con la generación de pruebas fácilmente modificables.

IMacros es un poderoso complemento de un excelente browser, obviamente estoy hablando de firefox. Este complemento permite al usuario crear macros de flujos sobre cualquier pagina o aplicación web dentro del firefox. Con ello se pueden automatizar casos de pruebas o simplemente flujos repetitivos para generar data. Este complemento es una buena opción de uso para empresas que no pueden gastar en licencias para realizar automatizaciones de casos de pruebas. En la empresa en la que trabajo tenemos herramientas propietarias para realizar ese tipo de trabajos, pero consumen más recursos. Es muy simple de usar tiene una interfaz muy intuitiva y además se permite la parametrización de los datos a usar en cada script.

Es muy útil para grabar cualquier acción que no puedas guardar como Favorito o Marcador. Por ejemplo, acceder a tus cuentas de correo, extraer enlaces de una web, rellenar formularios determinados, generar descargas o hacer una serie de clips determinados.

Su uso es muy sencillo. Para crear una macro, basta con pulsar el botón de grabar y realizar todas las acciones que deseas guardar. Y para reproducirlas, sólo hay que seleccionar la macro correspondiente y dar la orden.

En definitiva, iMacros es una interesante extensión para automatizar todas esas operaciones repetitivas que sueles hacer con Mozilla Firefox.

IPS Performance optimizer aprovecha al máximo su inversión en monitorización y carga de soluciones de pruebas para unir el hueco entre las pruebas de laboratorio y los entornos de producción, asegurando la ejecución de las aplicaciones antes de ponerlas en funcionamiento.

Con Performance Optimizer se puede:

- ✓ Comprobar que los procesos de negocios de la aplicación conocerán los requerimientos de niveles de servicio antes de la puesta en marcha. Mediante la construcción de elementos del entorno de producción como servidores compartidos, networks, número de usuarios y carga de trabajo, usted gana una visión realista de cómo su aplicación actuará en el entorno de producción.
- ✓ Ajustar el tamaño de la infraestructura para mantener los niveles y costes de servicio.
- ✓ Trabajar en múltiples escenarios "what if" utilizando el modelo IPS de más de 2.000 componentes hardware para determinar los óptimos requerimientos de infraestructura para admitir la aplicación y su crecimiento hacia los niveles de servicio requeridos.
- ✓ Predecir el impacto del cambio. Evalúa el impacto del cambio en la carga del trabajo y la infraestructura tiene un tiempo de respuesta en la aplicación.

1.3.2 Herramientas en Linux.

TestGen4Web es una herramienta para el registro de las acciones del usuario en un navegador Web Firefox, el ahorro de las acciones a un archivo XML, y reproducción de éstos. Para las páginas Web y las interfaces que no tienen JavaScript. TestGen4Web es una solución de pruebas mucho más fácil que el Selenium. El paquete incluye los medios para ejecutar las pruebas como las pruebas de unidad o simplemente generar la producción de Python que se puede utilizar en secuencias de comandos, etc.

OpenLoad Tester: Es la primera solución rápida de optimización de rendimiento basada en navegador, fácil de usar, para las pruebas de carga y estrés de aplicaciones y sitios web dinámicos. Dicha herramienta reduce sustancialmente el tiempo y el conjunto de habilidades necesarias para las pruebas de estrés y ajusta el rendimiento de las aplicaciones y servicios basados en Web, simplificando así el proceso de construcción de escenarios reales del mundo del usuario.

Beneficios:

- ✓ Consta simplemente de 4 pasos para la optimización de procesos.
- ✓ Permite descubrir rápidamente oportunidades para mejorar el rendimiento de sus aplicaciones y la infraestructura en cuestión de minutos.
- ✓ Va mas allá de limitarse a registrar el tráfico de Web, sino que inspecciona el contenido de cada respuesta y parametriza las variables de sección adecuado para el usuario automáticamente, eliminando así la necesidad de secuencias de comandos complejos y depuración.
- ✓ Posee un grabador que brinda soporte completo para el manejo de secciones basado en java Script, applets de Java, SOAP, XML, Flash, DHTML, marcos, ventanas emergentes, redireccionamientos, SSL, autenticación basada en formularios, autenticación básica, Oracle Forms, WebForms Net y más.
- ✓ El motor de análisis OpenLoad correlaciona automáticamente los usuarios, la aplicación y medición del lado del servidor en una única vista de informes simplificados que se clasifican por el rendimiento de aplicaciones, disponibilidad, fiabilidad, escalabilidad y capacidad.
- ✓ Permite ver el código HTML real o respuestas XML devueltas por el servidor durante la grabación y reproducción para garantizar la correcta visualización de los contenidos.
- ✓ Escala fácilmente a través de Windows, Linux y Unix y permite que se siga el mismo guión para múltiples entornos sin pérdidas funcionales.

Obtenido de: http://www.als-es.com/

C++Test es una herramienta multiplataforma, es el líder en unidad de pruebas automatizada y acceso al código para desarrollar una aplicación en C/C++. Genera de forma automática casos de pruebas, y strubs. Ejecuta pruebas unitarias para una verificación instantánea, permitiendo a los usuarios retocar y ampliar estas pruebas según sus necesidades. Además revisa el seguimiento del código mediante 700 reglas. Es

utilizado para prevenir errores y asegurar que el código satisface los estándares de código designados en los equipos. Para descubrir la fiabilidad, seguridad y problemas de funcionalidad con cada unidad, C++Test examina cada clase o función. Después genera y ejecuta los casos de prueba de unidad diseñados para alcanzar la meta de pruebas seleccionada. Además, proporciona rápidas y sencillas maneras de añadir y ejecutar casos de pruebas definidas por el usuario. Para asegurar la continuidad de la funcionalidad, C++Test automatiza la regresión de las pruebas e identifica los problemas introducidos por modificaciones del código. Posee un editor gráfico de casos de prueba y genera reportes de pruebas en archivos HTML y XML.

Beneficios:

- ✓ Reduce el tiempo de mercado mediante la identificación y reparación rápida de defectos en el ciclo de desarrollo.
- ✓ Mejora la eficiencia del equipo de desarrollo por medio de la automatización de las buenas prácticas y despliega un consistente flujo de trabajo para el análisis de código estático y unidad de pruebas.
- ✓ Implementa y hace cumplir automáticamente al equipo o a la compañía los estándares de código y directrices.
- ✓ Soporta desarrollo en multiplataforma con una herramienta de gestión de código de calidad.

SOATest es una herramienta multiplataforma, proporciona verificación instantáneas de servicios Web, simplifica el desarrollo SOA, automatiza las pruebas funcionales de cliente/servidor, pruebas de regresión, pruebas de carga/rendimiento y más. SOATest es el estándar de Arquitectura orientada a servicio más aceptado por la industria para su seguridad y fiabilidad. SOATest es una solución probada con clientes como: Yahoo, Sabre, LexisNexis e IBM. Las empresas seleccionan SOATest de Parasoft por su gran variedad de funcionalidad, incluyendo validación WSDL four-tier, pruebas unitarias y funcionales del cliente, pruebas unitarias y funcionales del servidor, pruebas de desarrollo así como la habilidad de modelar y probar gráficamente escenarios complejos. Es también la única solución de servicios Web que crea automáticamente pruebas de intrusión de seguridad en servicios Web, como inyecciones SQL, inyecciones Xpath, sobrecarga de parámetros, bombas XML y uso de entidades externas.

Beneficios:

- ✓ Asegura la fiabilidad, calidad, seguridad e interoperabilidad de un servicio web.

- ✓ Pruebas de intrusión integradas con pruebas funcionales, para completar la cobertura.
- ✓ Prevención de errores e identificación de posibles.
- ✓ Verificación de la integridad de los datos y la funcionalidad servidor/cliente.
- ✓ Acelera el tiempo de mercado.

WebKing es una herramienta de pruebas Web automatizada, que proporciona pruebas exhaustivas y análisis de los sitios y aplicaciones Web para asegurar que cumplen con la fiabilidad, seguridad y metas de desarrollo necesarios para sostener su negocio con eficiencia. Automatiza las pruebas Web de una aplicación y el análisis de riesgo del sitio, utilizando pruebas funcionales, pruebas de carga y ejecución de las pruebas de análisis de seguridad.

Beneficios:

- ✓ Pruebas web complejas y análisis en una sola prueba integrada.
- ✓ La automatización de una tarea completa proporciona resultados inmediatos.
- ✓ La automatización de la creación de las pruebas amplía la cobertura de las mismas.
- ✓ Permite a los desarrolladores, al equipo de pruebas y analistas programadores, colaborar y aprovechar el trabajo del otro.
- ✓ Visibilidad anticipada de errores y vulnerabilidades

LDTP: Pretende producir un framework de calidad de prueba de automatización con herramientas de última generación que pueden ser utilizados para probar y mejorar el sistema GNU / Linux o Solaris de escritorio. Utiliza las bibliotecas de Accesibilidad para registrar los casos de prueba basados en las acciones del usuario en una aplicación. GNU / LDTP puede probar cualquier GNOME o aplicaciones basadas en interfaz gráfica de usuario que tenga habilitada la accesibilidad por ejemplo: Mozilla, Openoffice.org, cualquier aplicación Java con una interfaz de usuario basada en Swing y aplicaciones basadas en GTK.

1.4 Sistemas SCADAS.

Los sistemas SCADA utilizan la computadora y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos, ya que pueden recoger la información de una gran cantidad de fuentes muy rápidamente, y la presentan a un operador en una forma amigable. Los sistemas SCADA mejoran la eficacia del proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas. (Montero, 2004.)

Los sistemas SCADA se utilizan en lugares mayormente industrializados en los que sea una necesidad automatizar y controlar los procesos. En estos sistemas los datos son manejados de forma tal que no existan errores fatales, pues como un sistema crítico, estos errores podrían equivaler a la pérdida de vidas humanas.

Un sistema SCADA está compuesto por 3 elementos fundamentales:

- ✓ Múltiples Unidades de Terminal Remota (conocidas como RTU)
- ✓ Estación Maestra y ordenador con interfaz hombre-máquina (HMI)
- ✓ Infraestructura de comunicación

1.4.1 HMI.

El término Interfaz Hombre-Máquina es usado frecuentemente en el contexto de los Sistemas de Computación y los Sistemas Electrónicos, constituye una capa intermedia que los independiza y permite la intercomunicación entre ambas partes. Existen en la actualidad diferentes tipos de HMI, se pueden identificar claramente dos de ellas: Interfaces Gráficas de Usuario (GUI) y las Interfaces de Usuario basadas en la Web.

El módulo de HMI en el SCADA se encarga de representar, en un ordenador, los procesos que ocurren en el campo en tiempo real, muestra los componentes implicados, los sensores, las estaciones remotas, y el sistema de comunicación dándole al operador total control. Éste módulo es el que permite al operador estar en contacto directo con el sistema y realizar la supervisión y el control del proceso en general. Está compuesto por dos partes fundamentales, el ambiente de configuración o editor y el ambiente de ejecución, éstas pueden ser aplicaciones separadas o estar incluidas en una sola.

El primero de estos permite configurar varios procesos o partes de ellos, aquí se definen y gestionan las variables, los drivers, los comandos, las alarmas y variadas opciones adicionales. Este ambiente funciona como una aplicación de diseño tradicional, con la peculiaridad que los sinópticos se confeccionan a partir de objetos y primitivas básicas predefinidas, que se pueden agrupar, combinar, transformar, importar y exportar entre otras.

El ambiente de ejecución se puede comparar con un reproductor multimedia, se encarga de visualizar las animaciones y los objetos definidos en el editor, muestra lo que está ocurriendo en el campo en tiempo real, es el que envía los comandos a las estaciones remotas, quién recibe los valores de las variables, es el que interactúa con la mayoría de los operadores pues se emplea para supervisar el proceso de manera directa.

El HMI (GUI) será componente al cual se le realizarán las pruebas automatizadas.

1.4.2 GTK+.

GTK+ son las siglas de GIMP Toolkit, es una biblioteca que permite crear interfaces gráficas de usuario, se distribuye bajo la licencia pública general (GPL), lo que hace de GTK+ un producto completamente libre. (GTK, 2007).

GTK+ es multi-plataforma, se ha extendido hasta Microsoft Windows y muchos derivados de Unix, como Linux y Mac OS. Está escrita en C y tiene extensiones en varios lenguajes como C++, Python, Perl y muchos más. Se empleó inicialmente en el proyecto Gnu Image Manipulation Program Tool Kit, por eso su nombre: GTK+, se ha extendido rápidamente por su estabilidad y una de las implementaciones que más se ha usado es la de C++, llamada Gtkmm (2007).

Gtkmm es la implementación oficial de GTK+ escrita en C++, le adiciona a GTK+ las potencialidades del paradigma orientado a objetos, la herencia para crear nuevos componentes, polimorfismo, manejo de memoria para construir y destruir objetos, elimina el uso de las macros de C y muchas otras mejoras.

GTK+ depende de otras bibliotecas que hacen de GTK+ un éxito total:

- ✓ Glib, una biblioteca de propósito general, no destinada a interfaces gráficas en sí, provee tipos de datos, macros y utilidades de conversión, tratamiento de cadenas y abstracciones muy útiles.
- ✓ Pango, se encarga de la manipulación de los textos internacionalizados, provee wigdets que se encargan de la representación de los textos.

- ✓ Atk, es el paquete de accesibilidad, provee un conjunto de interfaces que permiten a las interfaces de usuario interactuar con las tecnologías de accesibilidad.

- ✓ Gdk, es la capa de abstracción que permite a GTK+ ser portable a múltiples plataformas.

- ✓ GTK+, ella en sí contiene las definiciones de todos los widgets.

GTK+ tiene soporte para bases de datos, el proyecto Gnome-DB ofrece una arquitectura basada en CORBA que permite acceso totalmente transparente a distintas fuentes de datos, incluyendo datos que se encuentren en servidores LDAP o en ficheros XML, entre otros. GTK+ provee también mecanismos para comunicar aplicaciones de red mediante los protocolos TCP, UDP y para el trabajo con la tecnología XML.

1.5 Conclusiones.

En el capítulo se han abordado los elementos teóricos sobre los cuales se sustentará la aplicación de una herramienta para hacer Pruebas Automáticas de Software. Se definen los conceptos generales de proceso de pruebas, las ideas fundamentales de los sistemas SCADA y en especial el HMI GTK utilizado en el CEDIN para la construcción de interfaces del SCADA. De la investigación realizada se puede concluir que no existe una receta única para enfrentar esta tarea en la industria de software, su implementación está condicionada por diversos factores que van desde las capacidades de los recursos humanos hasta los recursos económicos disponibles.

En este capítulo se dan un conjunto de características de algunas herramientas existentes para hacer pruebas automatizadas de software. Dejando todo listo para definir cuál de estas herramientas candidatas es la que se ajusta a las pruebas funcionales de aplicaciones de escritorio de acuerdo a las necesidades del Centro de Desarrollo de Informática Industrial.

Capítulo 2: Desarrollo de la solución.

"Al desarrollar nuestra estrategia de prueba, debemos minimizar el impacto causado por los cambios en las aplicaciones que estamos probando, y los cambios en las herramientas que utilizamos para ponerlos a prueba".

Carl J. Nagle

2.1 Introducción.

Una manera de optimizar un proceso de Pruebas de Software es la introducción de herramientas que ayudan a agilizarlo y hacerlo menos tedioso. Dada la existencia de un gran número de dichas herramientas se necesitan criterios sobre los cuales establecer una comparación y así poder seleccionar el software de prueba que más se adapte a las necesidades del producto a probar. En el presente capítulo se describe la forma de selección de dicho software teniendo en cuenta las ventajas y desventajas de las herramientas mostradas y las necesidades del software a probar en este caso el HMI del SCADA. Además de la confección de un manual de usuario para el estudio y aprendizaje de la herramienta elegida.

2.2 Ventajas y desventajas del uso de herramientas.

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

2.2 .1 Ventajas:

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

ejecución en versiones anteriores del producto, sólo se necesita seleccionarla y volverla a ejecutar con un mínimo esfuerzo manual, posibilitando la ejecución de un mayor número de pruebas en una unidad de tiempo finita y por consiguiente la entrega de un producto de mayor calidad.

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

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

2.2.2 Desventajas:

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

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

No es recomendable la práctica de las pruebas automáticas por equipos que tengan mal organizada la realización de éstas. Ejemplos de malas prácticas son: poca o inconsistente documentación, pruebas que no son muy efectivas en la búsqueda de defectos, etcétera.

2.3 Proceso de Selección de herramienta para automatizar las pruebas.

En el proceso de selección de herramientas se evalúan cuáles son las apropiadas para las necesidades del Proyecto. Para que el proceso tenga éxito no se puede comenzar la búsqueda centrándose en las

herramientas que existen, lo primero y más importante, es identificar los requerimientos, características del proyecto al cual se le van a hacer las pruebas, en este caso (el HMI del SCADA), tipos de prueba empleados, lenguajes de programación, entre otros aspectos que fueron obtenidos a través de una entrevista realizada a los líderes del proyecto.

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

Para seleccionar la herramienta se debe tener en cuenta también, que no se cuenta con el conocimiento de ninguna herramienta precedente para hacer pruebas automáticas en el proyecto. Además se realiza especial énfasis en aquellas que sean software libre y gratis, sin dejar de reconocer la calidad de las que no son de libre distribución. Para este proceso se siguieron algunos criterios de selección, que a continuación serán mostrados en la siguiente tabla:

Criterio	**Descripción**
Plataforma.	La herramienta debe ser compatible con Software libre.
Tipo de prueba.	La herramienta está destinada para pruebas de interfaz de usuario a aplicaciones de Desktop.
Bibliotecas de desarrollo.	La herramienta debe permitir la realización de las pruebas a aplicaciones construidas en GTK.
Documentación	Existencia de documentación para el estudio de la herramienta.
Complejidad	La herramienta debe ser lo más fácil de usar para los usuarios y al mismo tiempo ser potente y robusta.

Tabla 1: Criterios de selección.

Criterios / Herramientas.	Plataforma.	Tipo de prueba.	Biblioteca de desarrollo.	Documentación.	Complejidad.	Puntuación.
TestGen4Web	Si	No	No	3	3	6
OpenLoad Tester	Si	No	No	5	5	10
C++Test	Si	No	No	3	3	6
SOATest	SI	No	No	3	4	7
RoutineBot	No	Si	No	3	5	8
vTest	No	No	No	4	4	8
WebKing	Si	No	No	5	3	8
LDTP	Si	Si	Si	5	5	10

Tabla 2: Criterios/Herramientas.

Nota: Los criterios resaltados en rojo, son necesarios a la hora de seleccionar la herramienta, y el criterio de documentación está contemplado sobre la base de que con 3 es poca, con 4 es buena y con 5 es excelente, el de complejidad se define como un 3 muy complejo, 4 complejo y 5 sencillo.

2.4 Propuesta de la herramienta.

Al llevar a cabo la incorporación de herramientas para pruebas automatizadas en las etapas de desarrollo del proceso de software es necesario tener en cuenta todos los aspectos mencionados en los epígrafes anteriores, pues éstos constituyen la base para lograr este objetivo. A partir de un estudio realizado en materia de herramientas para la automatización de pruebas y las características del software del Centro de Desarrollo de Informática Industrial se propone el uso de una herramienta para la realización de pruebas funcionales automatizadas a aplicaciones de escritorio.

A partir de los resultados arrojados por la tabla número 2, por sus características se seleccionó la herramienta LDTP (Linux desktop Testing Project) ya que esta cumple con todos los criterios necesarios.

LDTP (Linux desktop Testing Project) es una herramienta muy potente que corre sobre software libre y está bajo la licencia GPL. Está destinada a probar aplicaciones basadas en interfaz gráfica de usuario. Incluye además un lenguaje de programación de alto nivel (python) para la realización de pruebas automatizadas. Contiene un editor y un mapa de aplicaciones que permite grabar todas las acciones del usuario utilizando las bibliotecas de accesibilidad y facilita así la creación de los script de prueba. Soporta las plataformas siguientes:

- OpenSuSE.
- OpenSolaris.
- Debian.
- Ubuntu.
- Fedora.
- FreeBSD.
- Embedded Platform (Palm Source / Access Company)

LDTP brinda un camino fácil a sus usuarios para realizar scripts de pruebas, los probadores no necesitan conocer la jerarquía de objetos. Podemos monitorizar el uso del CPU por la aplicación que se esté probando. Al reporte XML generado por la herramienta se le puede aplicar un estilo XSLT y obtenemos un archivo en formato HTML, una buena práctica para ver los resultados de las pruebas.

2.4.1 Manual de usuario de la herramienta.

Para la utilización y aprendizaje de LDTP ver anexo número 9.

2.4.2 Instalación de LDTP.

Para realizar la instalación de esta aplicación es preciso tener instalado previamente en la computadora GNOME 2.10 *o* KDE 4.0, como mínimo**.** Además debe estar habilitada la accesibilidad.

Requerimiento del sistema:

- ✓ Requiere 450 KB de espacio en disco.

Requerimiento de software:

- ✓ Cspi-1.0 y bibliotecas de desarrollo >= 1.2.0, glib-2.0 >= 2.2.0, gobject-2.0 >= 2.2.0.
- ✓ Gail y bibliotecas de desarrollo.
- ✓ Libgail y bibliotecas de desarrollo o libgail-gnome basado en la distribución que uses.
- ✓ Python como ambiente de desarrollo.
- ✓ Libxml-2.0 >= 2.0.0 y bibliotecas de desarrollo.

Paquetes opcionales:

- ✓ Herramienta de importación ImageMagic, para realizar captura de pantalla.
- ✓ Bibliotecas de imágenes de python, para comparar imágenes.
- ✓ Pystatgrab, permite monitorizar la utilización de memoria del CPU.
- ✓ LTFX (Linux test for x) para operar sobre una ventana que no tenga habilitada la accesibilidad.

2.5 Resumen de las pruebas.

Objetivos trazados:

Correcto funcionamiento de:

- ✓ Los componentes del sistema y detección de errores.
- ✓ Las interfaces de los distintos subsistemas que la componen.

Para satisfacer los objetivos se elaboró un plan de prueba que establece como operar sobre las pruebas funcionales al Guardián del Alba. Además de los recursos humanos y materiales necesarios para ejecutar el proceso de prueba. Ver anexo 10.

Se describieron también las técnicas que serían utilizadas para llevar a cabo las pruebas así como el proceso de las mismas, el cual constó de varios pasos:

- ✓ Preparación para la ejecución de las pruebas.
- ✓ Medición.
- ✓ Evaluación.

Las pruebas funcionales fueron aplicadas específicamente sobre el Editor del proyecto Guardián del Alba. Todo lo planteado anteriormente constituye el pilar fundamental de la estrategia de pruebas al HMI.

2.5.1 Plantillas.

Propósito de los diseños de Casos de prueba: Realizar una descripción detallada de los pasos a seguir por el probador para probar un caso de uso.

Dicho formulario se genera por el diseñador de casos de prueba y es usado por el probador.

Partes de la plantilla:

Encabezado:

Esta parte contiene la información del nombre del proyecto, nombre del caso de prueba, versión, revisiones históricas y descripción general donde se da una breve descripción del número de casos de pruebas que se le desarrollarán al caso de uso correspondiente.

Detalle:

En esta parte de la plantilla se encuentra contenido todo el proceso paso por paso de cómo el probador debe ejecutar cada caso de prueba, contando este proceso con un flujo central, un flujo alterno si fuese necesario, condiciones de ejecución bajo las que se ejecutó el caso de prueba y las iteraciones donde se registran las clases válidas e inválidas según los datos suministrados, resultado esperado, el resultado obtenido y las observaciones necesarias.

2.5.2 Procedimiento de pruebas.

El diseño de las pruebas y los resultados de las mismas, quedaron establecidos en el documento Diseño de Casos de Prueba. Ver anexo 11.

Para los casos de prueba de las **pruebas funcionales**, la mayor información se obtuvo de la especificación de casos de uso. Para diseñar los casos de pruebas, se tuvieron en cuenta los requerimientos funcionales a implementar, y fundamentalmente las especificaciones de los casos de uso, que constituye la base de éstos.

A continuación se muestran algunas secciones y escenarios de los casos de prueba diseñados:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Agregar canal	EC 1.1: Propiedades del canal	Llenar todas las propiedades del canal.
SC 2: Modificar canal	EC 2.1: Clic derecho sobre el canal, seleccionar propiedades y ejecutar la Sección número 1.	Debe seleccionar propiedades del canal y luego ejecutar todos los escenarios de la sección 1.

SC 3: Eliminar canal	EC 3.1: Clic derecho sobre el canal, seleccionar Eliminar.	Seleccionar el canal a eliminar.
	EC 3.2: Error al intentar eliminar canal.	Al seleccionar un canal para ser eliminado muestra un error diciendo que está asociado a algún sub-canal.

Tabla 3: Administrar Canales.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Agregar punto analógico	EC 1.1: Información general	Debe llenar todos los campos de información para el punto analógico.
	EC 1.2: Alarmas	Debe habilitar y configurar la alarma.
	EC 1.3: Control	Debe llenar todos los campos para configurar el control del punto.
	EC 1.4: Históricos	Debe seleccionar un grupo histórico.
SC 2: Modificar punto analógico	EC 2.1 Clic derecho sobre el punto analógico, seleccionar propiedades y ejecutar la sección número 1.	Debe seleccionar propiedades del punto analógico y luego ejecutar todos los escenarios de la sección 1.
SC 3: Eliminar punto analógico	EC 3.1 Clic derecho sobre el punto analógico, seleccionar eliminar.	Seleccionar el punto analógico a eliminar.

Tabla 4: Administrar Puntos Analógicos.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Habilitar alarma de nivel	EC 1.1: Configurar alarma de nivel.	Debe habilitar y configurar la alarma de nivel.
	EC 1.1: Deshabilitar alarma de nivel.	Marcar deshabilitar la alarma de nivel.

Tabla 5: Configurar Alarma de Nivel.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Agregar dispositivo de control	EC 1.1: Información General.	Debe llenar los campos Nombre, Descripción, Grupo operacional de privilegios.
	EC 1.2: Alarma de falla de comunicación.	Debe marcar Habilitar, seleccionar una Severidad, elegir una Prioridad y un Número de reintentos.
	EC 1.3: Parámetros-Límites.	Se deben seleccionar todos los campos del formulario.
	EC 1.4: Parámetros-Protocolo Modbus.	Se deben seleccionar todos los campos del formulario.
	EC 1.5: Parámetros-Transporte.	Se deben llenar todos los campos del formulario.
SC 2: Modificar dispositivo de control.	EC 2.1: Clic derecho sobre el dispositivo de control, seleccionar propiedades y	Debe seleccionar propiedades del dispositivo de control y luego ejecutar todos los escenarios de la

	ejecutar la Sección número 1.	sección 1.
SC 3: Eliminar dispositivo de control.	EC 3.1: Clic derecho sobre el dispositivo de control, seleccionar Eliminar.	Seleccionar el dispositivo de control a eliminar.
	EC 3.2: Error al intentar eliminar un dispositivo de control.	Al seleccionar un dispositivo de control para ser eliminado muestra un error diciendo que está asociado a algún punto.

Tabla 6: Administrar dispositivos de Control.

2.6 Incorporación de scripts de prueba a la herramienta.

Luego del diseño de los casos de prueba realizados a partir de los casos de uso anteriormente presentados se implementan los scripts siguiendo dichos diseños. Este es el paso más importante en el proceso de automatización de pruebas. Seguidamente se procede con la ejecución de los scripts obteniendo un reporte en formato XML. Para un mejor entendimiento de los resultados se aplica un estilo al reporte y se logra mostrar los datos a través de un archivo en formato HTML.

Para el caso de prueba Agregar Canal se obtuvo el siguiente reporte:

REPORTES DE PRUEBAS AUTOMATIZADAS

Proyecto: Guardian Del Alba

Caso de Prueba: Administrar Canales

Nombre de la Prueba	Nombre del Script	Estado	Error(es)
SC1_Agregar_Canal.py	Escenario 1.1 Propiedades del Canal	0	Fallo el escenario 1.1 Propiedades del Canal
SC2_Modificar_Canal.py	Escenario 2.1 Clic derecho modificar	1	
SC3_Eliminar_Canal.py	Escenario 3.1 Clic derecho eliminar	1	

2.7 Ejemplificación de otros aportes o funcionalidades.

En este epígrafe se tratarán algunos aportes o funcionalidades que maximizan la utilidad y productividad de LDTP. Creación de servidores de pruebas, ejecución remota de los script y el uso del lenguaje XSLT para aplicar un estilo a los resultados de las pruebas.

2.7.1 Creación de servidor de pruebas.

Xvfb es un servidor que se puede ejecutar en máquinas sin dispositivos de salida de video ni dispositivos de entrada (mouse, teclado, etc). Emula un framebuffer con la memoria virtual. El principal uso que se le da es para montar servidores para prueba, pero otros usuarios más nuevos lo usan para otras cosas, como probar clientes con configuraciones de profundidad de color y pantalla inusuales, realizar procesos por lotes con Xvfb como motor de dibujado de fondo, intentar adaptar el servidor a nuevas plataformas .

Este paquete también contiene un script convenientemente llamado xvfb-run que simplifica la ejecución automatizada de los clientes de prueba en un entorno con un servidor virtual. Este script necesita usar el programa xauth.

El servidor de prueba puede iniciarse sin requerir una tarjeta de vídeo configurada usando el servidor Xvfb con el parámetro 1, con la pantalla por default (0) y con resolución de 800×600 a 24 bits, display inicia el escritorio por defecto (1) para iniciar de esta forma startx seria:

Start - 'wich xvfb':1-screen 0 800x600x24 && DISPLAY=:1 x11vnc

2.7.2 Ejecución de los scripts remotamente.

A través de la herramienta para prueba automatizada LDTP, se puede ejecutar los scripts de prueba desde una estación remota. La ejecución del motor LDTP puede correr sobre un servidor de prueba remoto y se puede ejecutar los scripts de prueba desde tu propio ordenador. A continuación se muestran los pasos a seguir.

1- Iniciar manualmente el motor LDTP en la computadora cliente a través de las siguientes opciones.

- $LDTP –s
- $ LDTP **--**script-engine
- $ LDTP -s -p 12345(puerto por defecto 23456)
- $ LDTP --script-engine --port=12345

2- Desde la computadora remota se necesita cambiar LDTP_SERVER_ADDR con la dirección IP del servidor remoto.

- export LDTP_SERVER_ADDR=xx.xx.xx.xx

3- Iniciar la ejecución de los scripts.

2.7.3 Base de datos o colección de casos de pruebas como scripts.

Todos los script que se generen a partir del resultado de los diseños de pruebas serán almacenados de forma organizada en una carpeta con el siguiente orden jerárquico Casos de Uso/Secciones de Prueba/Escenarios de pruebas.

2.7.4 XSLT

Al igual que XML, XSLT es un lenguaje de programación. Forma parte de la trilogía transformadora de XML, compuesta por las CSS (Cascading Style Sheets, hojas de estilo en cascada), que permite dar una apariencia en el navegador determinada a cada una de las etiquetas XML. XSLT (XML Stylesheets Language for Transformation**,** o lenguaje de transformación basado en hojas de estilo).

XSL: FO, (Formatting Objects**,** objetos de formateo) o transformaciones para fotocomposición o, en general, para cualquier cosa que no sea XML, como por ejemplo HTML o PDF.

Los programas XSLT están escritos en XML, y generalmente, se necesita un procesador de hojas de estilo, o stylesheet processor para procesarlas, aplicándolas a un fichero XML. El estilo de programación con las hojas XSLT es totalmente diferente a los otros lenguajes a los que estamos acostumbrados (tales como C++ o Perl). En la práctica eso significa dos cosas:

- ✓ No hay efectos secundarios. Una instrucción debe de hacer lo mismo cualquier que sea el camino de ejecución que llegue hasta ella. O sea, no va a haber variables globales, ni bucles en los que se incremente el valor de una variable, o tenga un test de fin de bucle, ni nada por el estilo. En realidad, esto no es tan grave, y se puede simular usando recursión.
- ✓ La programación está basada en reglas: cuando ocurre algo en la entrada, se hace algo en la salida.

2.7.5 Sumario de resultados de pruebas.

Criterios de comparación	Pruebas Manuales	Pruebas Automatizadas
Tiempo de diseño del caso de prueba.	480 min	480 min
Tiempo de programación de los script de pruebas.	0 min	40 min
Tiempo de ejecución de las pruebas.	60 min	10 min
Recurso humano para la ejecución de las pruebas	1 persona	1 persona
Recursos materiales para la ejecución de las pruebas.	1 computadora	1 computadora
Formato de resultado de las pruebas.	Word, PDF	HTML , PDF

Tabla 7: Comparación entre pruebas manuales y pruebas automatizadas en su primera iteración.

Criterios de comparación	Pruebas Manuales	Pruebas Automatizadas
Tiempo de diseño del caso de prueba.	0 min	0 min
Tiempo de programación de los script de pruebas.	0 min	0 min
Tiempo de ejecución de las pruebas.	60 min	10 min
Recurso humano para la ejecución de las pruebas	1 persona	1 persona

Recursos materiales para la ejecución de las pruebas.	1 computadora	1 computadora
Formato de resultado de las pruebas.	Word, PDF	HTML , PDF

Tabla 8: Comparación de pruebas manuales y pruebas automatizadas en su segunda iteración.

Casos de uso (u)	Pruebas Manuales	Pruebas Automatizadas
1	60 min	10 min
20	1200 min	200 min
50	3000 min	500 min
70	4200 min	700 min
100	6000 min	1000 min

Tabla 9: Comparación entre pruebas manuales y automatizadas de acuerdo a tiempo demorado por un probador en la realización de casos de uso.

De acuerdo a los resultados obtenidos anteriormente podemos decir que:

- El tiempo y los recursos son inversamente proporcionales.

El tiempo que demora un probador manualmente es 6 veces mayor que el tiempo que demora utilizando una herramienta para prueba automatizada. Por tanto, para que demoren el mismo tiempo hay que aumentar para pruebas manuales a 6 la cantidad de computadoras y probadores.

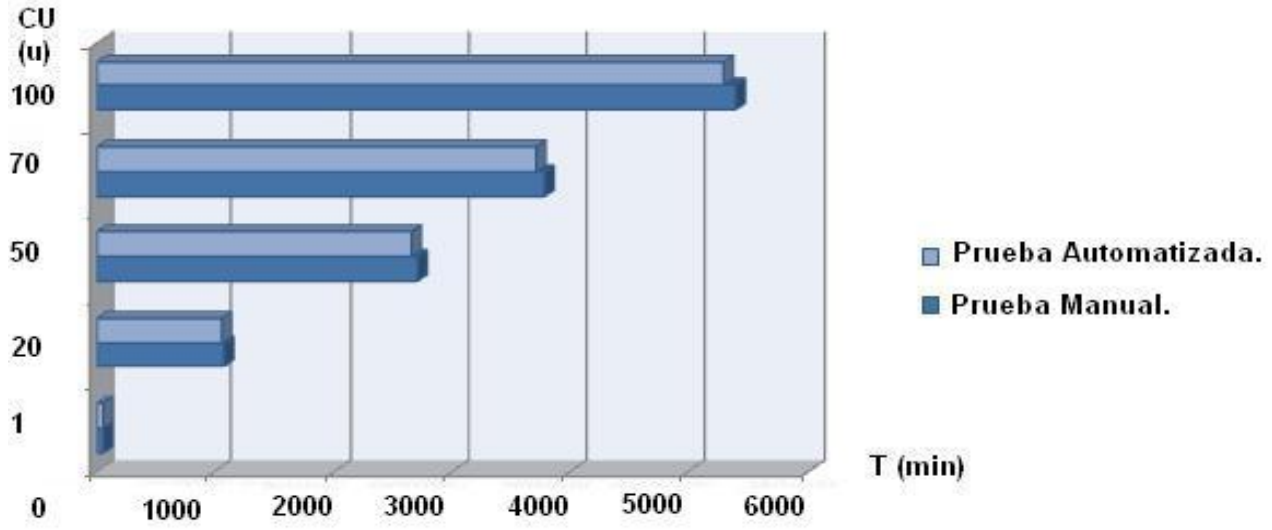
A continuación se muestra una tabla (tipo de prueba-recurso) para la realización de 100 casos de uso con la misma duración:

Tipo de Prueba	Recurso (Probador-Pc)
Pruebas Manuales.	6
Pruebas Automatizadas.	1

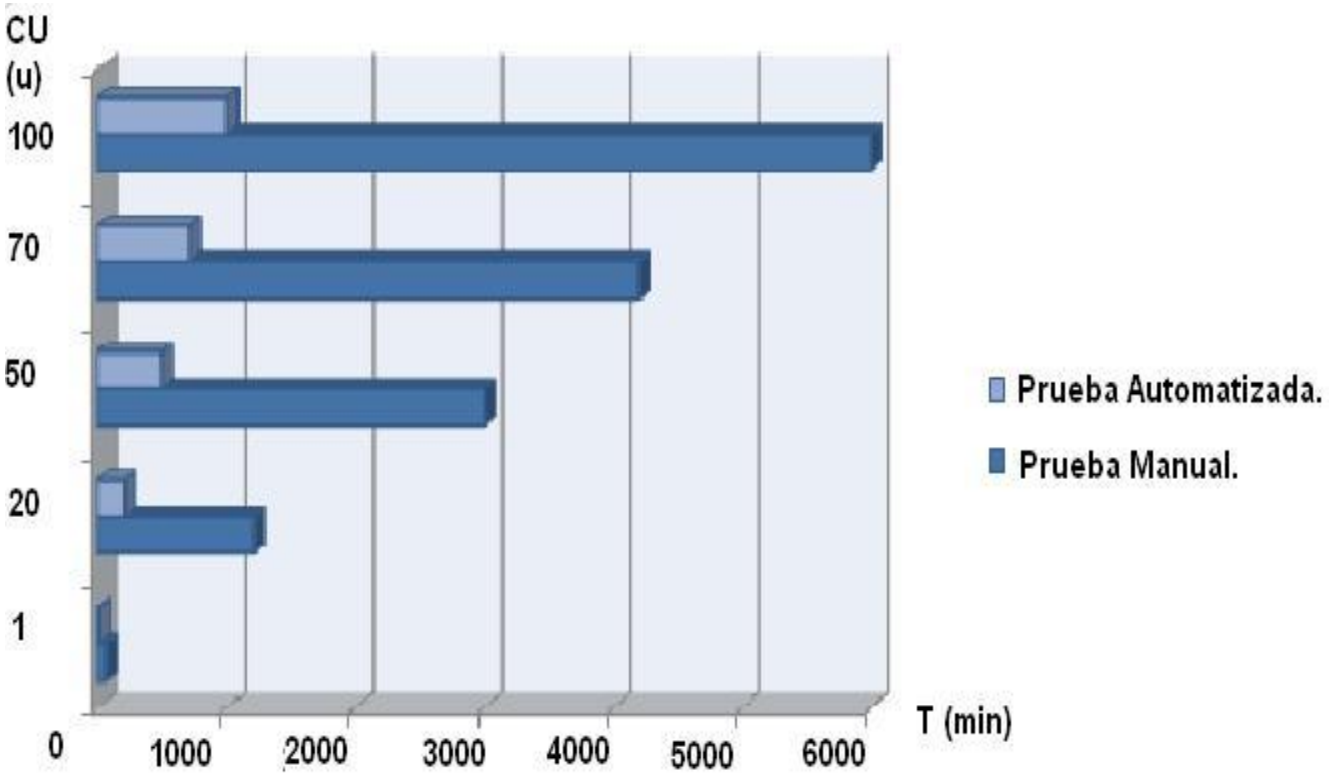
Tabla #10: Tipo de prueba-recurso.

A continuación se muestran un conjunto de gráficas que hacen más visibles los resultados alcanzados.

Para una primera iteración de prueba a 100 casos de uso la diferencia de tiempo entre las pruebas manuales y automatizadas fue de 16 horas de trabajo es decir 2 días para un probador. Para iteraciones posteriores de pruebas a los casos de uso esta diferencia aumentará considerablemente, debido a que en las pruebas automatizadas se ahorrará el tiempo de programación de los scripts.



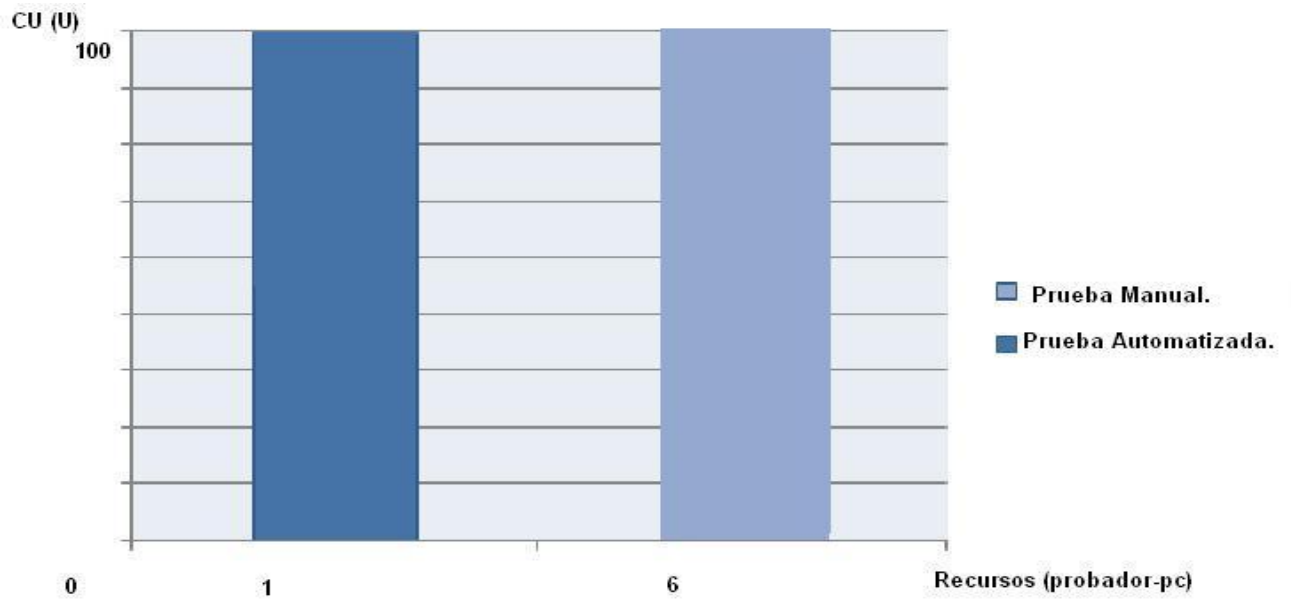
Primera iteración de pruebas.



Segunda Iteración de prueba.

Tiempo de diferencia entre pruebas manuales y automatizadas para una segunda iteración, en términos cuantitativos sería de 10 días de trabajo.

La siguiente gráfica (CU-recurso) muestra la cantidad de recursos empleados para pruebas manuales y automatizadas para la realización de 100 casos de uso siendo el tiempo de demora igual para ambas pruebas.



Recursos utilizados por casos de uso.

Suponiendo que un probador gana x cantidad de dinero en pesos por hora de trabajo, para una segunda iteración de prueba con la utilización de la herramienta para pruebas automatizadas (LDTP) el centro se ahorraría 80 veces dicha cantidad.

Continuidad y mantenibilidad:

Al integrar LDTP a la herramienta de integración del centro los scripts de prueba serán almacenados en una base de datos. Posibilitando la facilidad de mantener los actuales casos de pruebas e incorporar nuevos según los ciclos de desarrollo.

Errores humanos:

Cuando el proceso de pruebas se realiza de forma manual se pueden introducir errores a la hora de interactuar con la aplicación ya que es un proceso tedioso, por lo cual el probador debe estar muy concentrado y aún así no se garantiza que no ocurran errores. Es esta una de las desventajas que tiene la realización del proceso de forma manual.

Facilidad de uso:

Los scripts de prueba se realizan utilizando python y las funciones que brinda LDTP. Esto es muy conveniente ya que muchos de los integrantes del Centro están familiarizados con este lenguaje de programación. Además que a través del manual de usuarios que se propone es muy fácil aprender a trabajar con la herramienta.

Se utilizó LDTP versión 1 para la cual se requiere:

- ✓ GNOME Version - min 2.10.
- ✓ Hablilitar la accesibilidad.
- ✓ At-spi-devel.
- ✓ Gail-devel.
- ✓ Libgail-deve o libgail-gnome.
- ✓ Python-devel.
- ✓ Libxml2-devel.
- ✓ Python 2.5.

2.8 Conclusiones.

Durante el desarrollo del presente capítulo se trata todo lo referente al conjunto de herramientas existentes para automatizar el proceso de de prueba de software, además de exponer las ventajas que brinda utilizar una herramienta para automatizar este proceso. Para la selección de la herramienta a utilizar se define un conjunto de parámetros los cuales se determinan serían necesarios que cumpliera esta, dentro de ellos se encuentra: complejidad de uso, documentación, software libre entre otros.

En este capítulo a partir de los criterios de selección se tomó la decisión de utilizar la herramienta LDTP de la cual se hace un profundo estudio en todos los aspectos, además se muestra un manual de usuario

bien intuitivo con el objetivo de hacer más sencillo el aprendizaje de la herramienta para su posterior utilización en el Centro de Informática Industrial.

Se desarrolló todo el proceso de pruebas de forma manual y automatizada con el objetivo de demostrar prácticamente por qué es más factible probar automáticamente, se muestran todos los artefactos que se generan a partir de la ejecución de este proceso.

Conclusiones Generales.

Con el desarrollo de la presente investigación se obtuvieron significativos resultados así como aportes para el proceso de prueba del Centro de desarrollo de Informática Industrial (CEDIN). Se integró la herramienta para prueba automatizada (LDTP) a una herramienta de integración para ser extensible su aplicación a otros proyectos del centro. Se obtuvieron mejoras en cuanto a tiempo y recurso resolviendo uno de los principales problemas del proceso de prueba del CEDIN.

Recomendaciones.

Como recomendación se propone hacer extensible la herramienta LDTP para QT, además de agregar otras funciones LDTP para diferentes componentes gráficos en aplicaciones de desktop.

Bibliografía.

[KRU95] Kruchten, P. Architectural Blueprints. 2004.. *The "4+1" View Model of Software Architecture.* 2004.

Arechavala, Yolanda González y García, Fernando de Cuadra. 2001. *Calidad del software (I).* País Vasco : Vol: 1 , 2001.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler,M., el al. Grenning, J.,. 2001. *Manifesto for agile software development.* 2001.

Boehm, B. 2006.. *A view of 20th and 21st century software engineering.* New York, NY, USA. : s.n., 2006.

Corporation, [RSC02] Rational Software. 2002. . *Rational Software Corporation.* 2002. .

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000.. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education S.A. : 84-7829-036-2., 2000.

Juristo, N., Moreno, A. y Vegas, S. 2003.. *Limitations of Empirical Testing Technique Knowledge.* New Jersey. : s.n., 2003.

Kruchten, P. 2000.. *The Rational Unified Process: An Introduction.* 2000.

Montero. Citado el: 18 de 02 de 2010.. *Autómatas.* http://www.autómatas.org/redes/scada.htm : s.n., Citado el: 18 de 02 de 2010.

Montero, Dagoberto, Barrantes, David B. y Quirós, Jorge M. 2004.. *Introducción a los sistemas de control supervisor y adquisición de datos (SCADA). .* Costa Rica : s.n., 2004.

Pressman. 2002. *Ingenieria del Software. Un enfoque práctico. .* 2002.

[En línea] 2006. [Citado el: 12 de 02 de 2010.]. *Qscada.* http://qscada.sourceforge.net/. : s.n., [En línea] 2006. [Citado el: 12 de 02 de 2010.].

Shore, J., Warden, S. 2007. *The Art of Agile Development. O'Reilly Media,.* 2007.

Teams, Best Practices for Software Development. 1998. *[RSC98] Rational Software Corporation, Rational Unified Process.* 1998.

[En línea] 2004. [Citado el: 15 de 02 de 2010.]. *visual.* http://visual.sourceforge.net/new/index.php. : s.n., [En línea] 2004. [Citado el: 15 de 02 de 2010.].

Referencias Bibliográficas.

[KRU95] Kruchten, P. Architectural Blueprints. 2004.. *The "4+1" View Model of Software Architecture.* 2004.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler,M., el al. Grenning, J.,. 2001. *Manifesto for agile software development.* 2001.

Boehm, B. 2006.. *A view of 20th and 21st century software engineering.* New York, NY, USA. : s.n., 2006.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000.. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Education S.A. : 84-7829-036-2., 2000.

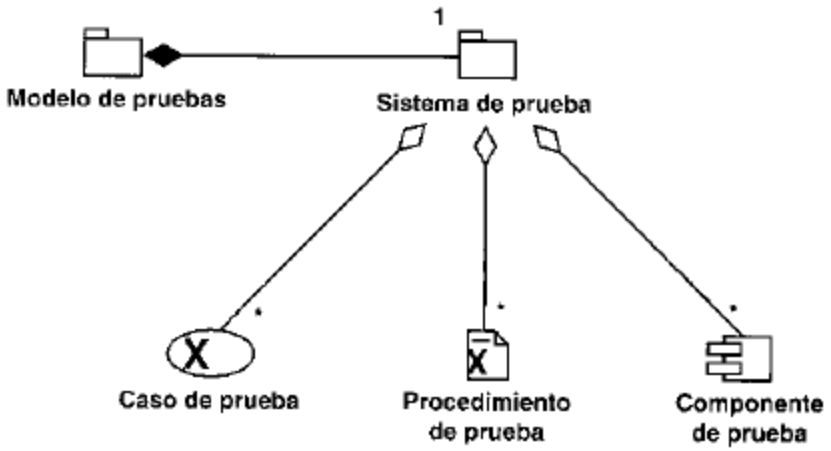
Juristo, N., Moreno, A. y Vegas, S. 2003.. *Limitations of Empirical Testing Technique Knowledge.* New Jersey. : s.n., 2003.

Pressman. 2002. *Ingenieria del Software. Un enfoque práctico. .* 2002.

Shore, J., Warden, S. 2007. *The Art of Agile Development. O'Reilly Media,.* 2007.

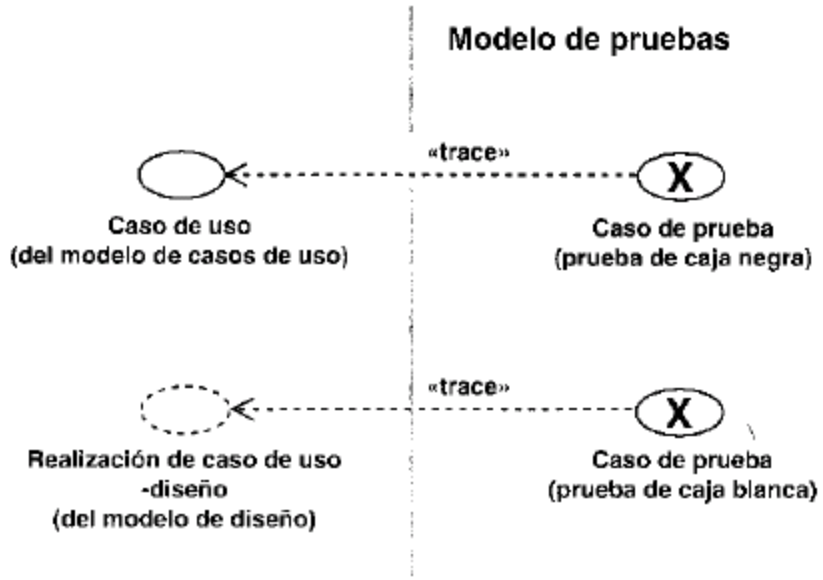
Anexos.

Anexo 1: Modelo de prueba



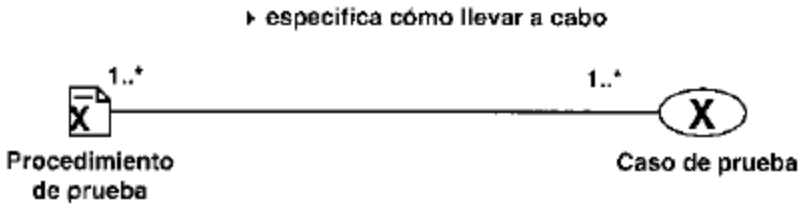
Un modelo de pruebas está compuesto por uno o muchos sistemas de prueba, el que a su vez posee relaciones de agregación de uno a mucho con los casos de prueba, procedimiento de prueba y componente de prueba.

Anexo 2:



El probador para la realización de los casos de prueba se guía por las especificaciones de los casos de uso.

Anexo 3:

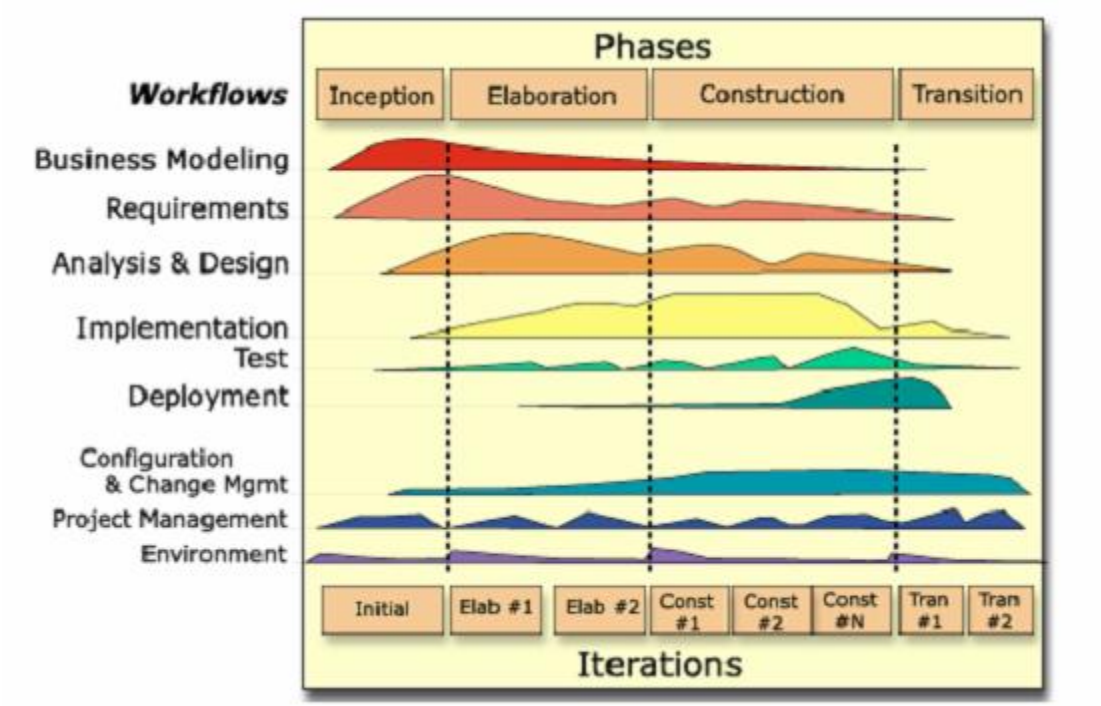


Hay asociaciones muchos-a-muchos entre los procedimientos de prueba y los casos de prueba.

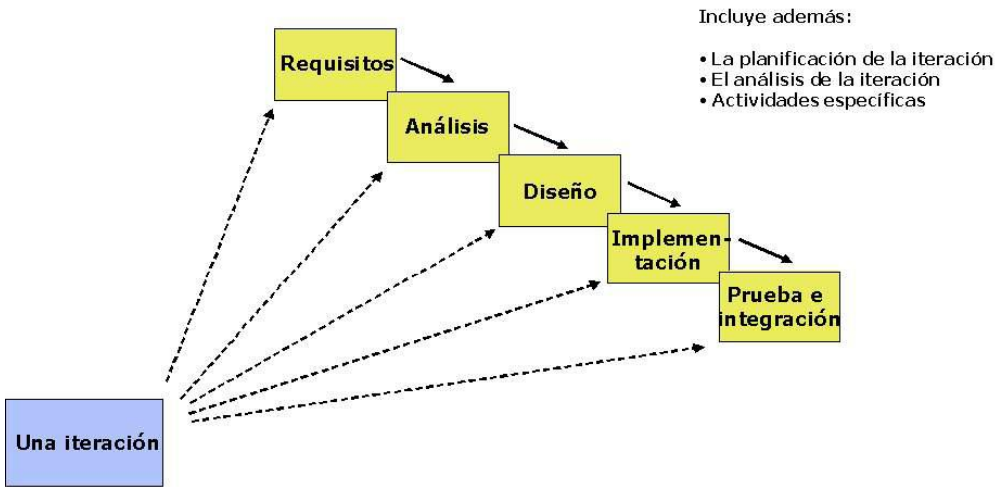
Anexo 4:

Hay asociaciones muchos-a-muchos entre los componentes de prueba y los procedimientos de prueba.

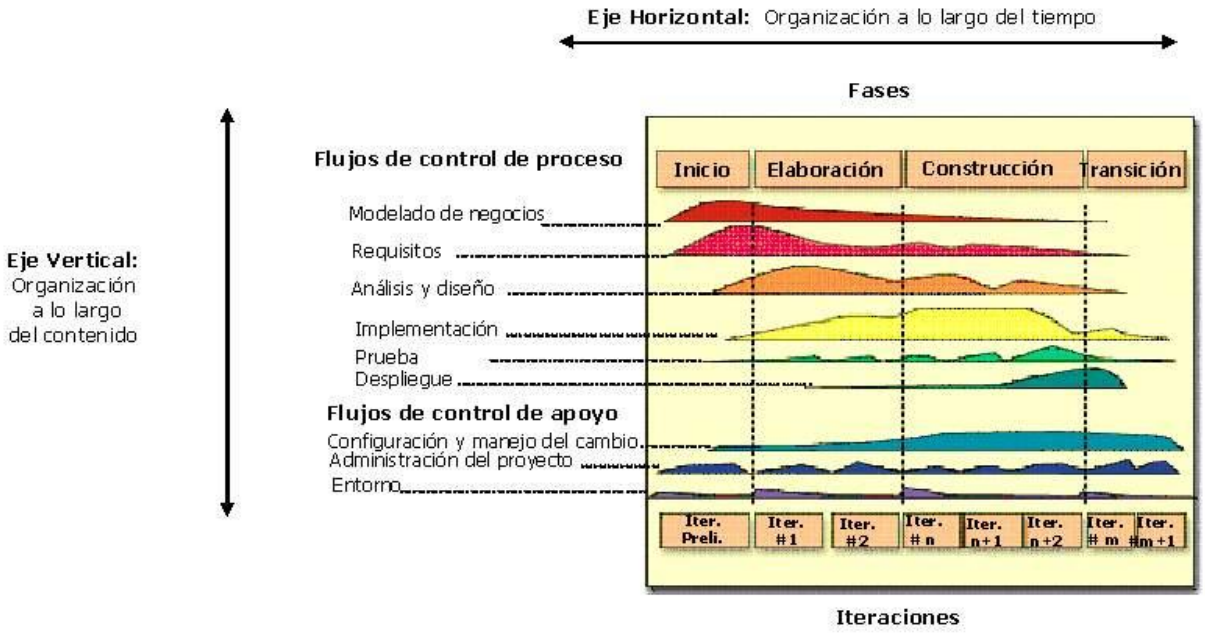
Anexo 5:



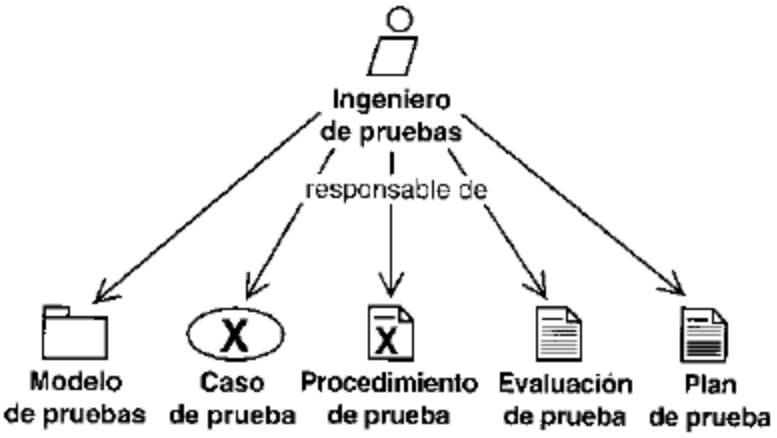
Anexo 6:



Anexo 7:



Anexo 8:



Las responsabilidades de un diseñador de pruebas en el flujo de trabajo de la prueba.

Anexo 9:

Manual de Usuario LDTP.

Este manual tiene como objetivo facilitar el uso y aprendizaje de la herramienta LDTP.

Pasos a seguir para ejecutar una prueba con LDTP.

- ✓ Ejecutar la aplicación a probar.
- ✓ Realizar los scripts de prueba basándose en la API de python-LDTP que se encuentra en la siguiente dirección: http://LDTP.freedesktop.org/user-doc/index.html
- ✓ Ejecutar los scripts a través del comando python. Ejemplo python NombScript.py
- ✓ Aplicar un estilo al reporte xml generado con el paso anterior, utilizando el comando xsltproc. Ejemplo xsltproc -o salida.html Estilo.xsl NombReporte .xml.
- ✓ Se obtiene un archivo html para mostrar los resultados de la prueba de forma más visible.

Ejemplo de una prueba utilizando LDTP a la Calculadora.

- ✓ Ejecutar la Calculadora.
- ✓ Realizar el script con el nombre Prueba.py: este será explicado para su mejor entendimiento.

```
from Ldtp import*   //Importando los módulos de LDTP

startldtplog ('<Ejemplo.xml>') // Generando el reporte XML de salida.


logMsg = 'Probando la Calculadora'

logMsg1= 'Caso de Prueba 1'
logMsg2= 'Caso de Prueba 2'
logMsg3= 'Caso de Prueba 3'
logMsg4= 'Caso de Prueba 4'
```

```
ldtplog(logMsg,  'begin') //Esta función indica el comienzo de una suite de prueba.

#Funcionalidad Sumar
ldtplog(logMsg1, 'teststart') //Generando el tag que indica el inicio de la pruaba.

//El siguiente bloque simula las acciones sobre la calculadora las cuales fueron 2+3=
y la herramienta verifica que el resultado sea 5. Este procedimiento se realizará para
las siguientes funcionalidades.

click('frmCalculator','btnNumeric2')
click('frmCalculator','btnAdd')
click('frmCalculator','btnNumeric3')
click('frmCalculator','btnCalculateresult')

//La siguiente condición verifica que la salida sea la esperada.
if gettextvalue('Calculator','txtResultRegion') ==6 :
  ldtplog('Error al sumar','Error')
else:
  ldtplog('successful','pass')
ldtplog(logMsg1, 'testend')



#Funcionalidad Restar
ldtplog(logMsg2, 'teststart')
click ("frmCalculator", "btnNumeric1")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnSubtract")
click ("frmCalculator", "btnNumeric5")
click ("frmCalculator", "btnCalculateresult")
if gettextvalue('Calculator','txtResultRegion') == 12:
  ldtplog('Error al restar','Error')
else:
  ldtplog('successful','pass')
ldtplog(logMsg2, 'testend')
#Multiplicar
ldtplog(logMsg3, 'teststart')
```

```
click ("frmCalculator", "btnNumeric5")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnMultiply")
click ("frmCalculator", "btnNumeric2")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnCalculateresult")
if gettextvalue('Calculator','txtResultRegion') == 12:
  ldtplog('Error al multiplicar','Error')
else:
  ldtplog('successful','pass')
ldtplog(logMsg3, 'testend')


#Funcionalidad dividir
ldtplog(logMsg4, 'teststart')
click ("frmCalculator", "btnNumeric1")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnDivide")
click ("frmCalculator", "btnNumeric5")
click ("frmCalculator", "btnNumeric0")
click ("frmCalculator", "btnCalculateresult")
if gettextvalue('Calculator','txtResultRegion') == 12:
  ldtplog('Error al dividir','Error')
else:
  ldtplog('successful','pass')
ldtplog(logMsg4, 'testend')


ldtplog(logMsg,'end') //Indica el fin de la suite de pruebas.
stopldtplog() //Indica el fin del archivo XML.
```

- ✓ Para ejecutar el script con el comando python de la siguiente forma: python Prueba.py
- ✓ Aplicando el estilo: xsltproc -o salida.html Estilo.xsl Ejemplo.xml

- ✓ Obtenemos el resultado de la prueba en el archivo salida.html.

Anexo 10:

Las tablas que se muestran a continuación se muestran contienen los juegos de datos que fueron usados para la realización de las pruebas.

1.1. SC 1 Agregar canal

Escenario	Variable 1	Variable 2	Variable 3	Respuesta Esperada
Propiedades del canal	Nombre canal1	Descripción estos es una prueba	Grupo operacional de privilegios NA	El sistema debe agregar el canal de forma correcta con los datos especificado.
	@@#$%^&*%$#@	prueba	NA	El sistema debe mostrar un mensaje de error diciendo que el nombre no es válido
	wi$%#^&*re*$#%	probando	NA	El sistema debe mostrar un mensaje de error diciendo que el nombre no es válido.

1.1. SC 1 Agregar dispositivo de control

Escenario	Variable 1	Variable 2	Variable 3	Respuesta Esperada
Información General.	Nombre pepe	Descripción Esto es una prueba	Grupo operacional de privilegios NA	El sistema deberá crear el dispositivo con los datos pasados.
	@%^$#@*(%#	Prueba	NA	El sistema deberá responder que hay campos que no están llenos deforma correcta.
	kk*/%pe	Prueba	NA	El sistema deberá crear el dispositivo con los datos pasados.

1.1. SC 1 Agregar punto analógico

Escenario	Variable 1	Variable 2	Variable 3	Respuesta Esperada
Información general	Nombre del punto a TAG pepe	Descripción esto es una prueba	Grupo operacional de privilegios NA	El sistema debe agregar el punto analógico.
	@pepe	Esto	NA	El sistema debe mostrar un mensaje de error diciendo que hay campos que están incorrectos.
	@@@@2	prueba	NA	El sistema debe mostrar un mensaje de error diciendo que hay campos que están incorrectos.

1.1. SC 1 Habilitar Alarma de nivel.

Escenario	Variable 1	Variable 2	Variable 3	Respuesta Esperada
Configurar alarma de nivel.	Habilitar Marcar el checkbox	Valor 2.00	Severidad NA	El sistema debe activar la alarma correctamente.
	No marcar el checkbox	100000000	NA	El sistema debe actualizar los valores de los campos dentro del rango establecido.
	Marcar el checkbox	100000000	NA	El sistema debe actualizar los valores de los campos dentro del rango establecido.

Glosario de Término

D

Diseño de Caso de Prueba.

E

Estrategia de prueba: un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba.

H

HMI: Estación Maestra y ordenador con interfaz Hombre-Máquina.

L

LDTP (Linux Desktop Testing Project).

M

MTU: Unidades de Terminal Central.

P

Plan de Prueba: informe en el cual se recogen todas las pruebas necesarias que se llevan a cabo en un proyecto.

Plantilla: forma de dispositivo que proporciona una separación entre la forma o estructura y el contenido.

PLC: Controlador Lógico Programable.

R

RTU: Múltiples Unidades de Terminal Remota.

S

SCADA: Sistemas de Supervisión y Adquisición de Datos.