Universidad de las Ciencias Informáticas

Facultad 6

Sistema para la automatización de pruebas unitarias a componentes en javascript en el contexto de una Línea de Productos de Software

Trabajo de Diploma para optar por el Título de
Ingeniero en Ciencias Informáticas

Autor(es): Ana Lilian López Cordero
Hector Yasmany Beltran Royé

Tutor(es): Ing. Diana Monné Roque
Ing. Frank González Fernández

Ciudad de La Habana, junio del 2011
"Año 53 de la Revolución"

"La responsabilidad nuestra es luchar porque la calidad del producto que aquí se haga, sea de las mejores y la mejor posible..."

Ernesto Che Guevara

Declaramos que somos los únicos autores de la presente tesis titulada: *"Sistema para la automatización de pruebas unitarias a componentes en javascript en el contexto de una Línea de Productos de Software"* y reconocemos, a la Universidad de las Ciencias Informáticas, sus derechos patrimoniales sobre la misma con carácter exclusivo.

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

Autores:

_____ _____

Ana Lilian López Cordero **Hector Yasmany Beltran Royé**

Tutores:

_____ _____

Ing. Diana Monné Roque **Ing. Frank González Fernández**

DATOS DE CONTACTO

Tutor: Ing. Diana Monné Roque
Ingeniero en Ciencias Informáticas
E-mail: dmonne@uci.cu

Tutor: Ing. Frank González Fernández
Ingeniero en Ciencias Informáticas
E-mail: ffernandez@uci.cu

Agradecimientos

A mi papi por los principios inflexibles que rigen su vida.

A mi mami por guiar a sus hijos por caminos profesionales, por su paciencia y cariño.

A mi Celi y mi Carli por siempre estar ahí..

A Maide por enseñarme que dejar de cumplir las reglas no siempre está mal.

A mi Catín por ser ese tesorito y demostrarme el largo camino de la vida.

A Yeni por revisar mi tesis.

A mis amigos por hacerme pasar ratos inolvidables.

A mis tutores por su ayuda incondicional.

A Yotuel mi dúo por soportar mis malcriadeces.

Ya ti mi negrito, por enseñarme a querer de una forma diferente.

… Gracias

Ani

Agradecer a mi madre por ser mi musa, mi meta, mi razón de ser, mi todo.

A mi papá Jose por hacer de mí un hombre de bien.

A mi padre Manuel por sus buenos consejos y guía en mi formación como profesional,

A mis hermanos por hacerme sentir como un ejemplo para ellos.

A mis amigos, los mejores que he tenido y el mejor recuerdo de mis 5 años como universitario.

A mis tutores por su guía y consejos.

y en especial a ana lilian mi dúo por soportarme y por tenerme paciencia.

… Gracias.

Hector

A ustedes, que son la razón de mi ser, mi MAMI y mi PAPI, este es el resultado de su sacrificio.

…a mi CECI y mi TIN, mis hermanitos del alma,

…a mi CATIN mi pequeño revoltoso,

…y a TI mi negrito que te has ganado ese derecho. LOS ADORO.

Ani

Dedico el presente trabajo a mi madre, mis papás Jose y Manuel y a mis hermanos.

Hector

Crear productos de software con calidad constituye el objetivo principal del grupo de desarrollo, por lo que el software es sometido a diferentes pruebas a lo largo de su ciclo de desarrollo. Una de ellas, son las pruebas unitarias, las que se hacen a nivel de desarrollador, que se encargan de probar individualmente cada uno de los componentes de un sistema, con el objetivo de aislar cada parte del programa y mostrar que las partes individuales son correctas.

En el Centro de Tecnología de Gestión de Datos no se cuenta con una serie de pasos bien definidos, para realizar las pruebas unitarias, además de que los desarrolladores las realizan de forma manual, debido a esta necesidad surge la investigación. Finalmente se define un procedimiento para la realización de las pruebas unitarias y se produce una mejora de la interfaz gráfica del JsUnit, además de que se le adicionan nuevas funcionalidades, obteniendo como resultado el sistema DATEST**,** que automatizará dichas pruebas.

Con el fin de confirmar la validez de la investigación realizada se aplicó el procedimiento a varios componentes javascript del proyecto SIGE. La puesta en práctica del procedimiento propuesto, implicó que se realizaran cada una de sus actividades y se plasmaran los resultados en cada una de las planillas involucradas, además de que se examinara la estructura interna del software, lo que disminuyó el número de errores existentes en los componentes, garantizando el concepto de calidad total definido en el Centro de Tecnología y Gestión de Datos. La utilización del sistema DATEST para la automatización de las pruebas unitarias agilizó todo el proceso.

Palabras Claves

Pruebas unitarias, Procedimiento, Componente, Calidad del software, Línea de Productos de Software.

Índice de Ilustraciones

INTRODUCCIÓN

El campo de las Tecnologías de la Información y las Comunicaciones ha alcanzado un auge mundial sin precedentes, trayendo consigo un gran avance en la industria del software. Todas estas innovaciones tecnológicas han llevado a que el tamaño y la complejidad del software aumenten rápidamente, mientras que los clientes y usuarios son cada vez más exigentes, en este sentido, alcanzar la calidad de los productos de software, continúa siendo un desafío para quienes se dedican a esta actividad.

El desarrollo de software es todo un proceso que enmarca una serie de tareas que tienen como propósito establecer un mínimo de calidad, verificando que los entregables al cliente cuenten con los requisitos necesarios que se exigen de acuerdo con los estándares asumidos en el proyecto. Lo más importante para los integrantes del equipo de desarrollo es crear un producto que sea eficaz y eficiente, para lo cual se hace necesario la constante revisión y corrección de los artefactos que se generan en cada una de las fases de desarrollo. Una de estas fases es la implementación, en la que se le da cumplimiento al diseño especificado.

Para garantizar la calidad dentro de esta fase, los desarrolladores realizan pruebas unitarias a cada uno de los componentes resultantes del proceso de desarrollo de software, las mismas no son más que una forma de comprobar el correcto funcionamiento de un módulo de código. Las pruebas unitarias a los componentes de software construidos durante la etapa de implementación cobran gran importancia pues se centran en encontrar y documentar defectos de la aplicación, velar por el cumplimiento de los requisitos del sistema, mejorar la productividad de los desarrolladores y la calidad final del producto. En este contexto, las pruebas unitarias juegan un papel significativo ya que ayudan a verificar el proceso de programación convirtiéndolo en una tarea eficiente, logrando con ello mejores resultados del trabajo. Con la automatización de las pruebas se ahorra tiempo, y recursos en las revisiones y detecciones de errores en el sistema, el equipo de desarrollo puede conducirse a un avance continuo e ininterrumpido en las metas propuestas por el proyecto. Se procura un mínimo de calidad en los componentes que se implementan y se obtiene en gran medida el producto deseado.

La creciente complejidad del desarrollo de software ha llevado a nuevas vías para producir sistemas. Entre las vías sobresalen las Líneas de Productos de Software que buscan obtener beneficios,

que minimizan los costos y el tiempo, para elevar la calidad a partir de las características comunes que comparten sus productos.

Para destacar la importancia que tienen las pruebas unitarias de software en el proceso de desarrollo de software, Rod Johnson planteó: *"Una revolución ha ocurrido en el desarrollo del software en los últimos años. De ser un marginado del desarrollo, despreciado por esos con más cosas importantes e interesantes que hacer, las pruebas – al menos, las pruebas unitarias – se han convertido en el corazón del proceso de desarrollo. Finalmente, en vez de ser visto como un requerimiento tedioso de ciertos procesos (…), ha sido integrado como una manera altamente productiva de trabajo"*. (1)

Cuba, no se encuentra ajena al desarrollo acelerado de la ciencia y las tecnologías de la información. Ello implica que las empresas informáticas enfrenten cada día un reto de brindar una respuesta rápida, eficaz y con calidad a los clientes, que cada vez se vuelven más exigentes. Una muestra de estas entidades productoras de software es la Universidad de las Ciencias Informáticas, que, a pesar de ser un centro destinado a formar profesionales altamente calificados en esta rama, produce software y brinda servicios informáticos.

En la Universidad de las Ciencias Informáticas la producción del software se estructura por centros productivos. En la facultad 6 se encuentra el Centro de Geoinformática y Señales Digitales y el Centro de Tecnología de Gestión de Datos, este último dividido en diferentes departamentos, y a cada uno de ellos se le asignan un conjunto de proyectos para desarrollar, basado en los principios de calidad total y mejora continua. Uno de estos departamentos es "Integración de Soluciones", que está enfocado a las Líneas de Productos de Software, el mismo está integrado por tres grupos de trabajo, el de análisis, arquitectura y desarrollo. En el grupo de desarrollo los desarrolladores realizan de forma manual las pruebas a los artefactos que generan, por lo que no se tienen en cuenta determinados criterios a la hora de probarlos. Emplean mucho tiempo realizando la depuración del código que escriben, por lo engorroso del proceso, de recrear condiciones apropiadas. Además no se cuenta con una planificación adecuada y concreta para realizar las pruebas, por lo que el cronograma puede resultar más largo y demorar la entrega del producto. No realizar con detalles la depuración del código trae consigo que a la hora de liberar los productos se encuentren una serie de no conformidades. Los desarrolladores no garantizan un mínimo de calidad requerida, cuando la revisión está sujeta a mecanismos intuitivos y criterios humanos.

Teniendo en cuenta la situación problémica existente se plantea el siguiente **problema de la investigación**: ¿Cómo contribuir al proceso de pruebas unitarias a componentes en javascript en el contexto de una Línea de Productos de Software?

La investigación tiene como **objeto de estudio**: el proceso de pruebas unitarias a componentes en javascript enmarcado en el **campo de acción:** pruebas unitarias en una Línea de Productos de Software.

El **objetivo general** de la investigación es: definir un procedimiento para realizar pruebas unitarias a componentes javascript en el contexto de una Línea de Productos de Software.

En correspondencia con ello se plantean los siguientes **objetivos específicos**:

- ✓ Caracterizar el proceso de pruebas unitarias a componentes en javascript en el contexto de una Línea de Productos de Software.
- ✓ Definir el procedimiento para la gestión de pruebas unitarias a componentes en javascript.
- ✓ Aplicar el procedimiento mediante el análisis, diseño e implementación y prueba al sistema.

Para dar cumplimiento a los objetivos se desglosan las siguientes **tareas**:

- ✓ Análisis bibliográfico relacionado a pruebas unitarias en una Línea de Productos de Software.
- ✓ Análisis de los enfoques de calidad total en una Línea de Productos de Software.
- ✓ Identificación de las actividades del procedimiento.
- ✓ Descripción de las actividades del procedimiento.
- ✓ Identificación de las funcionalidades del sistema.
- ✓ Especificación del Modelo de Sistema.
- ✓ Especificación del Modelo de Diseño.
- ✓ Implementación del sistema.
- ✓ Ejecución de las pruebas al sistema.
- ✓ Aplicación del procedimiento haciendo uso del sistema.

Aporte de la investigación:

Se obtendrá un “Procedimiento para la gestión de pruebas unitarias a componentes en javascript” y un “Sistema para la automatización de su aplicación” en el contexto de una Línea de Productos de Software. La solución “procedimiento + sistema” facilita la aplicación del enfoque de calidad total definido en el Centro de Tecnología de Gestión de Datos.

El contenido de la investigación está estructurado en 3 capítulos:

Capítulo 1: Fundamentación teórica de las Pruebas Unitarias
En este capítulo se analizan diferentes conceptos de calidad como punto de partida para la fundamentación. Seguidamente se realiza un estudio de las pruebas de software, la automatización de las mismas y como caso particular las pruebas unitarias, y su inserción en una Línea de Productos de Software. Además se realiza un estudio comparativo de las herramientas que automatizan el proceso de pruebas unitarias y se menciona la metodología, las tecnologías y las herramientas utilizadas para el desarrollo del sistema.

Capítulo 2: Procedimiento para Pruebas Unitarias
Las pruebas unitarias que se realizan al software tienen un impacto significativo en la calidad del producto final, por esta razón en el presente capítulo a partir de la fundamentación teórica se ofrece una propuesta de solución, para garantizar la realización de pruebas unitarias a los artefactos producidos por el desarrollador. Como propuesta de solución se ofrece un procedimiento, el que se describe en términos de entrada, actividades a desarrollar y salida. Se describen las actividades propuestas y sus principios de forma general.

Capítulo 3: Aplicación del procedimiento y análisis de resultados.
En este capítulo, se realiza el modelo de dominio para el levantamiento de los principales conceptos a tratar en el sistema. Se le hace una mejora a la interfaz gráfica de la herramienta seleccionada, además se le adicionan nuevas funcionalidades especificándose en los requisitos funcionales y no funcionales de la misma. Para posteriormente aplicar el procedimiento haciendo uso del sistema con su interfaz mejorada y analizar los resultados, cumpliendo todas las actividades descritas en cada una sus fases, demostrando la fiabilidad del mismo.

CAPÍTULO 1:FUNDAMENTACIÓN TEÓRICA DE PRUEBAS UNITARIAS

Introducción

En este capítulo se analizan diferentes conceptos de calidad como punto de partida para la fundamentación. Seguidamente se realiza un estudio de las pruebas de software, la automatización de las mismas y como caso particular las pruebas unitarias, y su inserción en una Línea de Productos de Software. Además se realiza un estudio comparativo de las herramientas que automatizan el proceso de pruebas unitarias y se menciona la metodología, las tecnologías y las herramientas utilizadas para el desarrollo del sistema.

1.1 Calidad del Software

En el mundo globalizado de hoy, la calidad de los productos de software se ha convertido en una necesidad ineludible para permanecer en el mercado, es por ello que la industria del software viene realizando grandes esfuerzos por mejorar la calidad de los productos.

Existen varias definiciones de la calidad del software expresadas por diversos autores destacados en este sector, tales como:

“Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente”. (2)

“La calidad del software es el grado con el que un sistema componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”. (3)

Partiendo de las definiciones antes expuestas se puede concluir que la calidad del software es el cumplimiento de los requisitos establecidos, para la conclusión de un proyecto de forma satisfactoria, de acuerdo a los parámetros y estándares previamente registrados.

1.2 Pruebas de Software

Una de las actividades críticas en el aseguramiento de la calidad de software es la prueba. “La prueba es el flujo de trabajo fundamental cuyo propósito general es comprobar el resultado de la implementación mediante las pruebas de cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema que van a ser entregadas a terceras partes” (4). El nombre “prueba”, además de la actividad de probar, se puede utilizar para designar “un conjunto de casos y procedimientos de prueba”. (5)

En la práctica, las pruebas son las actividades necesarias que se realizan sobre un programa informático, para verificar su correcto funcionamiento, así como la existencia de errores de diversas índoles, por lo que sin dudas corresponden a uno de los flujos ingenieriles más importantes en el ciclo de desarrollo de software.

Bill Hetzel, en “The Complete Guide To Software Testing”, planteó como objetivos de las pruebas de software (6):

- ✓ Planificar las pruebas necesarias en cada iteración.
- ✓ Diseñar e implementar las pruebas creando los casos de prueba que especifican que probar, diseñando los procedimientos de prueba que especifican cómo realizar las pruebas y estableciendo, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- ✓ Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan 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.

Analizando lo antes expuesto se plantea que las pruebas de software ofrecen la posibilidad de controlar que los productos de software cumplan los requisitos mínimos de operatividad, además de garantizar la calidad de los mismos.

1.3 Pruebas automatizadas de software

La fase de pruebas de todo el proceso de producción de software puede tardar un tiempo aún más largo que la fase de producción del mismo. Esto se debe a que los probadores deben explorar todos los ámbitos

posibles y el ángulo del producto de software a fin de detectar errores, defectos en el programa y luego trabajar para corregir en consecuencia. La automatización de las pruebas comprende el uso de estrategias, herramientas y artefactos que aumentan o reducen la interacción manual o humana en tareas repetidoras o redundantes.

1.3.1 Ventajas de las pruebas automatizadas de software

Las pruebas automatizadas permiten a los desarrolladores ser mucho más audaces en el modo en que modifican el software existente. Esto se debe a que aseguran una forma más evolutiva del desarrollo, soportando una entrega incremental de funcionalidades para el cliente, acelera la retroalimentación por parte del usuario y mejora la calidad del producto de software en construcción.

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

Los desarrolladores pueden ejecutar las pruebas cada vez que modifiquen el código de producción y necesiten obtener un voto de confianza de manera rápida. Conciben las pruebas como una red de seguridad que trata de atrapar errores, y les permite trabajar con más rapidez. El desarrollo se hace más productivo, independientemente del esfuerzo adicional implicado en escribir las pruebas.

Sin embargo, escribir un buen código de prueba no es sencillo. Cuando se hace, en muchos casos, existe una fuerte tendencia o tentación a rendirse de seguir escribiéndolo. Es cierto que siempre habrá un costo adicional para crear y mantener una suite de pruebas automatizadas. Muchos desarrolladores experimentados sostienen el criterio, de que es peor emplear y gastar más tiempo tratando de encontrar el motivo por el cual algo está fallando, por lo que automatizar pruebas de software minimizaría el impacto de lo que llamaríamos un error humano y facilitaría la mejora continua, ahorrando tiempo y esfuerzo dentro del desarrollo de software.

1.4 Estrategia de pruebas

Las pruebas se aplican durante todo el ciclo de desarrollo de software para diferentes objetivos y en distintos niveles de trabajo, la estrategia que se ha de seguir a la hora de evaluar dinámicamente un sistema de software debe permitir comenzar por los componentes más simples y más pequeños e ir

avanzando progresivamente hasta probar todo el software en su conjunto. Una decisión estratégica sería seleccionar cuáles son las pruebas a automatizar y qué herramientas utilizar.

1.4.1 Selección de pruebas automatizadas

Existen muchos puntos de vistas por los cuales se puede o no definir tipos de pruebas. Uno es el que divide las pruebas en las siguientes categorías: (7)

Pruebas de funcionalidades cruzadas – verifican varios aspectos del comportamiento del sistema que interrelacionan funcionalidades específicas. Generalmente son manuales y se realizan para criticar y valorar el producto.
Pruebas por funcionalidad – verifican el comportamiento del sistema bajo prueba en respuesta a un estímulo en particular. Prueban el sistema a varios niveles. Estas pruebas son usualmente automatizadas y forman parte del apoyo al desarrollo del producto.

En el caso de las pruebas por funcionalidad verifican el comportamiento directamente observable de una pieza de software en respuesta a un estímulo específico. La funcionalidad puede estar relacionada con el negocio o con requerimientos operacionales, incluyendo el mantenimiento del sistema y escenarios específicos de tolerancia a fallas.

En consecuencia, se pueden subdividir según los propósitos que persiguen en: pruebas de aceptación, pruebas de componentes y pruebas unitarias.

Pruebas de aceptación - verifican el comportamiento completo del sistema o de la aplicación en cuestión. Típicamente corresponden a escenarios de uno o más casos de uso, características o historias de usuario. Pueden ser automatizadas por desarrolladores, pero la característica clave es que el usuario final debe ser capaz de reconocer el comportamiento especificado por la prueba, incluso si estos no pueden leer la representación de la prueba.

Pruebas unitarias - verifican el comportamiento de sólo una clase o método, que es consecuencia de una decisión de diseño. Este comportamiento no es típicamente relacionado a los requerimientos excepto cuando el segmento clave de la lógica de negocio es encapsulado dentro de la clase o del método en

cuestión. Son escritas por desarrolladores para su propio uso. Ayudan a describir qué es lo que "parece estar terminado", resumiendo el comportamiento de la unidad en forma de pruebas.

Pruebas de componentes - verifican componentes que consisten en grupos de clases que colectivamente proveen algún servicio. Estas se encuentran entre las pruebas unitarias y las pruebas del cliente en cuanto al tamaño del sistema bajo prueba que está siendo verificado.

1.4.2 Herramientas para la automatización de pruebas unitarias a javascript

Una herramienta ofrece los servicios necesarios para dar soporte a una tarea determinada. Para la realización de pruebas unitarias a javascript existen una serie de herramientas que facilitan su creación. La selección de la herramienta adecuada es también una decisión estratégica, por lo que se hace necesario el análisis de los siguientes puntos en aras de identificar las restricciones que condicionan la elección de la misma:

- ✓ Entorno de aplicación de la herramienta hardware y software.
- ✓ Plataforma donde debe ejecutarse la herramienta.
- ✓ Factores políticos (obligatoriedad de comprar a determinados fabricantes, etc.).
- ✓ Calidad de la herramienta (documentación, complejidad, frecuencia de fallos, riesgo de que la herramienta provoque fallos en otras partes del entorno, etc.).

Siguiendo los puntos antes expuestos se analizaron las siguientes herramientas teniendo en cuenta que no se pretende enumerar todas ellas exhaustivamente, ni hacer una comparación entre las más importantes, la herramienta universal no existe.

RhinoUnit

Funciona de manera similar a otros marcos de pruebas NUnit, aunque algo modificado para requisitos particulares de javascript. Es sumamente pequeño, pero también sorprendentemente poderoso. Dirigido por un script ANT, y reduce la cantidad de repeticiones de prueba. Se ha intentado ejecutar el código de prueba de trabajo con JsUnit, y se ha encontrado problema al cargar los archivos comunes de javascript.

YUI Test

Es un framework de pruebas basadas en el lado del cliente. En el mismo se derivan algunas características de NUnit y JUnit. Permite una rápida creación de casos de prueba a través de una sintaxis sencilla, posee una avanzada detección de fallos a los métodos que generan errores, agrupa los casos de prueba relacionados en suit de pruebas. No posee un vínculo directo a cualquier marco xUnit. Algunos navegadores móviles como Opera Mini 4, no son compatibles con algunas de sus funcionalidades, las cuales son necesarias para las pruebas asincrónicas.

JsUnit

Es un framework de pruebas unitarias para el lado del cliente (en el navegador) de javascript. Proporciona un servidor de prueba (Java), que permite la ejecución automática de pruebas en todos los navegadores disponibles. Plug-in de Eclipse y obliga a tener una instalación local del marco JsUnit.

Comparación entre las herramientas:

Atendiendo a los criterios enunciados anteriormente se comparan las herramientas en la siguiente tabla:

Herramienta	Plataforma de Ejecución	Entorno de Aplicación	Factores Políticos	Calidad	Total
RhinoUnit	**X**	**X**			**2**
YUI Test	**X**	**X**	**X**		**3**
JsUnit	**X**	**X**	**X**	**X**	**4**

Tabla 1.1 Comparación entre las herramientas

Fuente: Investigación realizada

Analizando los resultados arrojados por la tabla se decide utilizar como motor para la automatización de las pruebas unitarias a javascript, JsUnit, ya que acumula una mayor cantidad de puntos, pues incluye una plataforma para la ejecución de las pruebas unitarias a javascript en varios navegadores y máquinas múltiples ejecutando sistemas operativos diferentes, es una herramienta de código abierto, o sea software

libre, y es con la que mayor documentación se cuenta, pues es la más difundida por los desarrolladores en la web. No presenta frecuencia a fallos. Además se trata de un puerto de JUnit a JavaScript, es el framework Xunit para javascript, conserva los estándares de un típico framework Xunit, escrito 100 % en javascript. Corre en las principales combinaciones plataforma/navegador y en el lado del cliente. Realiza las pruebas unitarias mediante los Tests Functions (funciones de prueba), permite agrupar una serie de páginas de prueba en suite de pruebas, es una herramienta de pruebas a javascript puro.

1.5 Pruebas Unitarias

Rod Johnson define “las pruebas unitarias como el nivel más fino de granularidad en las pruebas, que verifican una simple unidad de funcionalidad y deben probar que cada método de una clase satisface su contrato documentado”. (1)

Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente. Las pruebas unitarias aseguran que un único componente de la aplicación produzca una salida correcta para una determinada entrada. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. Se encargan de un único caso cada vez, lo que significa que un único método puede necesitar varias pruebas unitarias si su funcionamiento varía en función del contexto.

Para este tipo de pruebas se deben tener claros los siguientes aspectos:

- ✓ Los datos de entrada son conocidos por quien diseñe las pruebas y estos deben ser preparados con minuciosidad, ya que el resultado de las pruebas depende de estos.
- ✓ Se debe conocer de antemano qué resultados debe devolver el componente según los datos de entrada utilizados en la prueba.
- ✓ Finalmente se deben comparar los datos obtenidos en la prueba con los datos esperados, si son idénticos se puede decir que el módulo superó la prueba.

Las pruebas unitarias tienen una mayor dependencia de la implementación, y no descubrirán todos los errores del código, estas solo prueban que el segmento de código que se revisa esté lógicamente correcto. No descubrirán errores de integración, y, aunque se haya conseguido tener trazabilidad entre el código y los requisitos, en las pruebas unitarias no se tiene la perspectiva de sistema completo, por lo que "no se puede decir a priori que si la función pasa la prueba unitaria, el requisito de sistema se vaya a satisfacer". (8)

¿Por qué realizar pruebas unitarias?

Se realizan pruebas unitarias porque:

- ✓ Aseguran la calidad del código entregado. Es la mejor forma de detectar errores tempranamente en el desarrollo. No obstante, esto no asegura detectar todos los errores, por tanto las pruebas de integración y aceptación siguen siendo necesarias.
- ✓ Ayudan a definir los requisitos y responsabilidades de cada método en cada clase probada.
- ✓ Simplifican la integración: permitiendo llegar a las pruebas de integración con un alto grado de seguridad de que el código está funcionando correctamente, facilitando dichas pruebas.
- ✓ Permiten encontrar errores en los inicios del desarrollo, teniendo en cuenta que: mientras más temprano se corrijan los errores, menos costará corregirlos.
- ✓ Fomentan el cambio: las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- ✓ Le dan más seguridad al programador: aseguran que una corrección no repercuta en otros módulos, permitiendo al programador centrarse en la corrección del error y no en la repercusión que puede tener esa corrección.
- ✓ Las pruebas funcionales se hacen más sencillas: pues la mayoría de los aspectos individuales de cada unidad ya están probados a través de las pruebas unitarias. De este modo, las pruebas funcionales deben centrarse solo en verificar la correcta cooperación de las distintas unidades y en los funcionamientos generales del programa.

1.6 Métodos de diseño de casos de prueba

Los métodos de diseño de casos de prueba se utilizan con el objetivo de facilitar la búsqueda de errores con el mínimo consumo de tiempo y recursos, garantizando, de esta forma, la obtención de un producto correcto.

Casos de prueba: Es un artefacto que explica la forma de probar el sistema, detallando las entradas, salidas y las condiciones de la prueba que se va a realizar. Para su mejor realización se debe redactar el procedimiento de pruebas y de esta forma ayudar al ingeniero de pruebas a la ejecución de las mismas.

Según Rod Johnson (9), para el caso particular de las pruebas unitarias se pueden mencionar dos formas principales de realizar estas pruebas, Método de Caja Negra y Método de Caja Blanca.

Método de Caja Negra

Método de Caja Negra o Funcional: "Se realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar" (10). También conocidas como pruebas de comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. Se centran principalmente en los requisitos funcionales del software.

Para seleccionar el conjunto de entradas y salidas sobre las cuales trabajar, se debe que tener en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en el sistema, y como consecuencia producen una serie de salidas que revelan la presencia de defectos. Entonces, dado que la prueba exhaustiva es imposible, el objetivo final es encontrar una serie de datos de entrada cuya probabilidad de pertenecer al conjunto de entradas que causan dicho comportamiento erróneo sea lo más alto posible.

En esencia este método permite encontrar (11):

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.

- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.

Lograr una buena cobertura de código con las pruebas de Caja Negra aseguraría que el software hace lo que se espera de él, pero no se aseguraría que haga (además) otras acciones menos aceptables.
Para confeccionar los casos de prueba de Caja Negra existen distintos criterios; algunos de ellos son (11):

- ✓ Técnica de la Partición de Equivalencia: Divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- ✓ Técnica del Análisis de Valores Límites: Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- ✓ Técnica de Grafos de Causa-Efecto: Permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Método de Caja Blanca

"El método de Caja Blanca se basa en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa" (10). El objetivo del método es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa. Con estos casos de prueba se pretende demostrar que las funciones del software son operativas, aceptando correctamente los datos de entrada y produciendo los resultados esperados, donde la estructura interna se comporta de la manera requerida.

Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Este método garantiza que:

- ✓ Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- ✓ Se ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- ✓ Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- ✓ Se ejerciten las estructuras internas de datos para asegurar su validez.

Puede ser impracticable realizar una prueba exhaustiva de todos los caminos de un programa. Por ello se han definido distintas técnicas de pruebas para implementar este método, que permiten decidir qué sentencias o caminos se deben examinar con los casos de prueba (11):

- ✓ Prueba del Camino Básico: Permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa. Constituye la técnica más usada dentro del método de Caja Blanca.
- ✓ 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: 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: Se centra exclusivamente en la validez de las construcciones de bucles.

1.7 Línea de Productos de Software

En el proceso de desarrollo de software el crecimiento de la capacidad tecnológica ha estado siempre unido con el incremento de la complejidad, y ello a su vez ha propiciado el nacimiento de nuevas técnicas para hacer frente a dicha complejidad. Las Líneas de Producto Software (LPS) han aparecido en el proceso de desarrollo de software como una técnica cuyo objetivo es el de poder crear diferentes variantes de software a partir de una infraestructura común, se basan en la reutilización del software y en el desarrollo basado en componentes. Las mismas aumentan la productividad y promocionan al extremo la reutilización. Se definen como:

“Un conjunto de sistemas con uso intensivo de software que comparten un grupo común y regulado de características que satisfacen las necesidades de un mercado o misión particulares, y que son desarrollados a partir de un núcleo común de activos siguiendo un método prescrito”. (12)

Básicamente una LPS está conformada por una entrada compuesta por los activos de software que consisten en una colección de partes de software (requisitos, diseños, componentes, casos de prueba,

entre otros) que se configuran y componen de una manera prescrita para producir los productos de la línea. El control de la línea se compone de modelos de decisión y decisiones de productos. Los modelos de decisiones describen los aspectos variables y opcionales de los productos de la línea. Cada producto de la línea es definido por un conjunto de decisiones (decisiones del producto). El proceso de producción establece los mecanismos o pasos para componer y configurar productos a partir de los activos de entrada. Las decisiones del producto se usan para determinar qué activos de entrada utilizar y cómo configurar los puntos de variación de esos activos. Como salida se obtienen los productos de software que son el conjunto de todos los productos que pueden o son producidos por la LPS.

Los procesos de negocio de una LPS, están compuestos por (Ver ilustración 1.1):

- ✓ **La Ingeniería de Dominio (ID):** captura información y representa el conocimiento sobre un dominio determinado, con el fin de crear activos de software reutilizables en el desarrollo de cualquier nuevo producto de una LPS. La ID tiene como productos definiciones de dominios (descripciones del contexto), modelos del dominio, modelos de requisitos del dominio, modelos arquitectónicos (arquitecturas de dominio), ontologías del dominio, lenguajes del dominio y estándares del dominio. Su actividad principal es la de analizar la familia de productos para determinar los requisitos que son comunes, se encarga de realizar el diseño de la arquitectura de dominio de la LPS la cual tiene componentes comunes a todos los miembros de la familia, componentes opcionales que son requeridos por algunos miembros y componentes variantes de los cuales algunos miembros de la familia emplean distintas versiones. La implementación del dominio consiste en la creación y almacenamiento de los activos de software que se emplearán para producir los productos de software.
- ✓ **La Ingeniería de Aplicaciones:** se encarga del desarrollo de los productos de la LPS a través de la reutilización de activos de software y planes de producción. La arquitectura de dominio es empleada como un modelo de referencia para diseñar los productos de la línea. El repositorio de la misma provee los activos requeridos durante el desarrollo de cada nuevo producto.
- ✓ **La Gestión Técnica:** realiza la recolección de datos, métricas y seguimiento, analiza el cómo hacer/comprar/descubrir/encomendar (aprovisionamiento de activos) los activos necesarios para la LPS, se definen los procesos y alcance. Gestión de la Configuración, planificación técnica, gestión de riesgos técnicos y soporte de herramientas son tareas propuestas en esta área.

- ✓ **La Gestión Organizacional:** relacionada con la organización de la empresa y las actividades que ella debe implantar para asegurar el aprovechamiento eficaz y eficiente del paradigma LPS. Los aspectos organizacionales de las LPS involucran la aplicación de un conjunto de prácticas de gestión como la construcción de casos de negocio, gestión de relaciones con los clientes, desarrollo de una estrategia de adquisición, análisis de mercados, planificación organizacional, gestión de riesgos organizacionales, estructuración de la empresa, proyección de tecnologías y capacitación del personal. Esta área es la que dirige la LPS manteniendo una estrecha relación con las aéreas.

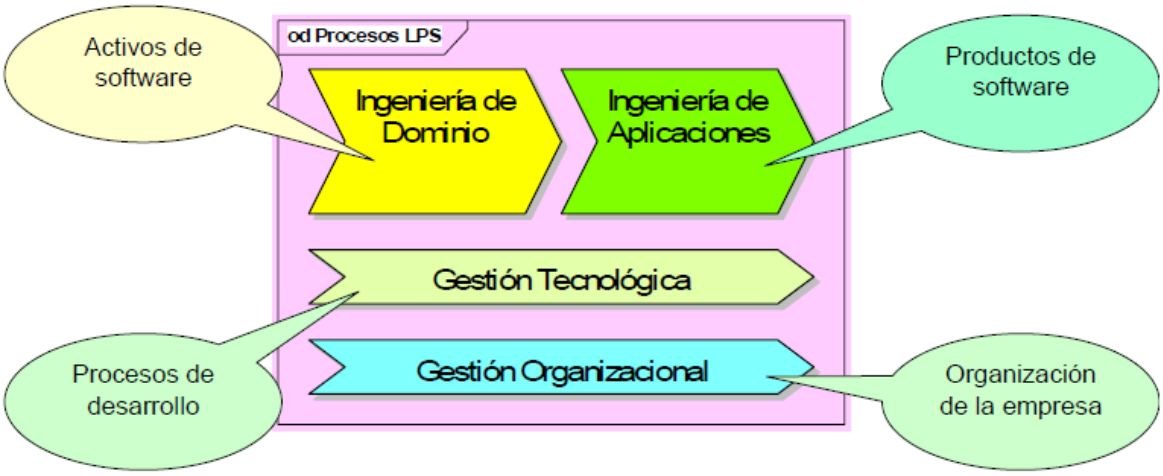

Ilustración 1.1 Proceso de negocio de una LPS

Fuente: Conf: Introducción a los nuevos modelos de Producción de Software. Dr.C Pedro Y. Piñero Pérez

Concluyendo las LPS garantizan reducir tiempo, esfuerzo, costo y complejidad de crear y mantener los productos de la línea mediante la consolidación y reutilización de los activos de entrada. Constituyen la reducción en el número promedio de defectos por producto, reducción en el esfuerzo promedio requerido para desarrollar y mantener los productos, incremento en el número total de productos que pueden ser efectivamente desplegados y mantenidos, mejoras en el valor competitivo del producto y márgenes de mayores ganancias.

1.7.1 Pruebas en una Línea de Productos de Software

"Las pruebas de una Línea de Productos de Software, deben examinar los activos básicos, el software específico de los productos, y las interacciones entre ellos" (8). J.D McGregor en su libro "Bulding

Reusable Test Assetsfor a Product Line" comenta que, aunque lo habitual es que las pruebas de sistema y de integración estén bajo la responsabilidad de quien desarrolla los productos, el equipo responsable de los activos básicos debe de realizar unas pruebas unitarias tan exhaustivas como sea posible. Es necesario también establecer unos mecanismos de gestión de defectos, para resolver los posibles problemas de integración con la plataforma que pudieran existir. (13)

Los artefactos empleados en las pruebas (documentos, conjuntos de datos de prueba, herramientas software) de los activos básicos de la Línea de Productos de Software pueden reutilizarse también en toda la LPS. Lo mismo puede decirse para las pruebas de los productos.

No se puede afirmar de forma general que las pruebas puedan reutilizarse en una LPS, dependerá de las características del dominio, de los recursos disponibles, y de la arquitectura de la LPS. Si podemos aislar partes en la arquitectura, probarlas por separado y luego ensamblarlas sin temor a efectos colaterales, será viable omitir algunas de las pruebas en determinados productos. Si no es así, habrá que repetir todas las pruebas. "En este tipo de desarrollo, se requiere un gran esfuerzo en las pruebas unitarias y de integración" (8). La cuestión fundamental es decidir hasta qué punto es posible probar las diferentes variantes de forma común.

1.7.2 Pruebas Unitarias en una Línea de Productos de Software

En una LPS se propicia una reutilización a alto nivel, desarrollando los elementos comunes que servirán como base para la generación de otros nuevos. La estructura de la prueba debe apoyar la trazabilidad desde el código de prueba para el código que pone a prueba, en desarrollo iterativo, incremental en todo el proceso que corrige sus errores. Para maximizar la trazabilidad, el software de prueba debe reflejar la arquitectura de LPS siempre que sea posible, cuando dos partes de la arquitectura de LPS tienen una relación particular, la prueba unitaria para cada una de las partes también se relaciona, este enfoque reduce el costo de mantener el software de prueba unitaria, ya que es más fácil identificar dónde hacer los cambios, facilitando así la reutilización, por lo que la arquitectura de la LPS puede proporcionar apoyo para la prueba.
A mayor grado de exhaustividad de las pruebas unitarias de los componentes, menor riesgo de que fallen al integrarse en la arquitectura final, facilitando así la reutilización de los componentes. Siempre existirá el

riesgo de fallos en la especificación de las interfaces y de que haya problemas de rendimiento al integrar el hardware y el software en el sistema completo.

En cualquier caso las pruebas unitarias de un componente de software son iguales si, el mismo pertenece a una LPS o a un sistema de software que se desarrolla de forma individual. Estas tienen la peculiaridad en el caso de los "activos básicos" que deben de ser válidas para toda la LPS, lo cual es posible solo si los datos internos de los mismos están encapsulados y sus interfaces son sencillas y están definidas.

1.8 Metodología de desarrollo de software

En la actualidad, desarrollar un software es una tarea complicada, que exige una metodología de desarrollo para simplificar el trabajo y garantizar un producto con la calidad requerida. Una metodología propone un conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto. Existen varias metodologías de desarrollo de software, cada una con sus propias características, aunque objetivamente persigan lo mismo. En dependencia de las particularidades de cada producto de software se escoge la metodología a ser aplicada. Ejemplos de estas son: RUP, XP, Open/UP.

El grupo de arquitectos del departamento LPS "Integración de Soluciones" ha definido las tecnologías, las herramientas, así como metodología de desarrollo a utilizar dentro de la línea.

Proceso Unificado Abierto (*Open Unified Process*, OpenUP) (14)

Open/Up es una versión más ágil de la metodología RUP, que puede ser utilizada en varios tipos de proyectos de software.

Mantiene las características esenciales de RUP, en el cual se incluyen las siguientes:

- ✓ Desarrollo iterativo e incremental.
- ✓ Uso de casos de uso y escenarios.
- ✓ Manejo de riesgos.
- ✓ Diseño basado en la arquitectura.

Su proceso puede ser personalizado y extendido para distintas necesidades, que aparecen a lo largo del ciclo de vida del desarrollo del software, dado que su modelo de desarrollo es iterativo e incremental, es capaz de producir versiones, además, una de sus ventajas es que puede ser acoplado para proyectos pequeños, dado que en su gráfica de roles aparecen 4 personas.

Roles de Open/Up

Analista

El analista es el que representa al cliente y el usuario final, se refiere a la obtención de requerimientos de los interesados, por medio de comprender el problema a resolver, capturando y creando las prioridades de los requisitos.

Arquitecto

El arquitecto es el responsable del diseño de la arquitectura del software, tomando las decisiones técnicas claves, las cuales limitarán el conjunto de diseño y la implementación del proyecto.

Desarrollador

El desarrollador es quien tiene la responsabilidad del desarrollo de una parte del sistema o el sistema completo dependiendo de la magnitud del mismo, se encarga del diseño, ajustándolo a la arquitectura y de la implementación de pruebas unitarias y de integración para los componentes desarrollados.

Líder de Proyecto

El líder de proyecto es quien dirige la planificación del proyecto en colaboración con las partes interesadas y el equipo, coordina las interacciones de los interesados, manteniendo al equipo del proyecto enfocado en los objetivos del mismo.

Fases de Open/Up

1. Concepción: en esta fase se pretende determinar los objetivos y establecer el alcance del proyecto.
2. Elaboración: el propósito de esta fase es establecer la línea base de la arquitectura del sistema y proporcionar una base estable para el desarrollo de la siguiente fase.
3. Construcción: esta fase tiene como propósito completar el desarrollo del sistema basado en la arquitectura definida, enfocándose en el diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo.
4. Transición: en esta fase se asegura que el software esté listo para entregarse a los usuarios.

Flujos de Trabajo de Open/UP (propone 6):

1. Requerimientos: Se realizan entrevistas con el cliente para comprender el problema a resolver y se definen los requisitos.

2. Análisis y Diseño: Se realiza el diseño de los requisitos que serán después implementados.
3. Implementación: Se realiza la implementación del sistema basándose en el diseño realizado.
4. Prueba: Busca los defectos a los largo del ciclo de vida.
5. Gestión del Proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
6. Gestión de Configuración y Cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización y actualización, control de versiones, etcétera.

1.9 Herramientas CASE

Las herramientas Ingeniería de Software Asistida por Ordenador conocida por sus siglas en inglés como Herramientas CASE (Computer Aided Software Engineering), son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Existe una larga lista de Herramientas CASE, Rational Rose y Visual Paradigm son algunas de ellas.

Visual Paradigm 6.1

Es una herramienta de desarrollo que se integra al Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) de Eclipse. Diseñada para desarrollar software con Programación Orientada a Objetos (POO), busca reducir la duración del ciclo de desarrollo. Es un producto distinguido que facilita la organización de los diagramas, su misión es diseñar, integrar y desplegar las aplicaciones de la empresa y sus bases de datos subyacentes. La herramienta ayuda al equipo de desarrollo de software a agilizar el modelado del software, aumentando al máximo y acelerando el trabajo en equipo y las contribuciones individuales. Se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, JBuilder, NetBeans IDE.

Ventajas de Visual Paradigm (15):

- ✓ Entorno de creación de diagramas para UML 2.0.
- ✓ Generación de código (PHP).
- ✓ Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.

- ✓ Capacidades de ingeniería directa e inversa.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- ✓ Disponibilidad de múltiples versiones para cada necesidad.
- ✓ Disponibilidad de integrarse en los principales IDE.
- ✓ Disponibilidad en múltiples plataformas.

Visual Paradigm 6.1 representa la herramienta de modelado más poderosa y de mejor valor en el mercado actual. Combina las funcionalidades de todas las ediciones en una amplia plataforma de modelado visual. Está diseñada para brindar apoyo a arquitectos, desarrolladores, diseñadores UML, analistas de procesos de negocio y modeladores de datos con el fin de agilizar todo el proceso de desarrollo de código del modelo para aplicaciones empresariales complejas.

1.10 Lenguaje de modelado

El lenguaje de modelado es un conjunto estándar de símbolos y de formas que facilita la modelación de un diseño de software.

Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML 2.0)

Es un lenguaje para especificar, construir, visualizar y documentar los artefactos (información que es utilizada o producida mediante un proceso de desarrollo de software) de un sistema de software orientado a objetos, que por su potencialidad se ha convertido en un estándar. Es el lenguaje que propone Open/Up. Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo. (16)

1.11 Entorno de Desarrollo Integrado

El ambiente de desarrollo (Development Environment) es algo imprescindible en la producción de software. Es donde se definen el conjunto de herramientas y tecnologías, versiones a usar y su integración, que intervienen en un proceso de desarrollo de software.

IDE NetBeans 6.9

El NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Es una base modular y extensible usada como una estructura de integración para crear grandes aplicaciones de escritorio. Proporciona herramientas para la programación en el lenguaje PHP e incluye resaltado de sintaxis, auto-completado de código, templates, documentación de PHP integrada, entre otras funcionalidades. Provee soporte para el framework Symphony y cuenta con un módulo de subversion para facilitar la manipulación de las versiones del código durante la implementación. Es un producto libre y gratuito sin restricciones de uso.

1.12 Tecnologías del lado del cliente

JavaScript

Es un lenguaje de programación interpretado, utilizado en la realización de páginas Web. Una de sus principales características es que es Orientado a Objetos (OO). La verdadera fuerza de JavaScript es la compatibilidad con los distintos navegadores, incluso con los más anticuados. Además de que es interpretado por todos los navegadores modernos. (17)

1.13 Frameworks

Un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Proporciona una estructura que fuerza al desarrollo de código más legible. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio y permite separar en capas la aplicación.

ExtJS 2.2

Es un framework de presentación completamente Orientado a Objetos de javascript, es multiplataforma y hace uso de la tecnología AJAX. Este framework brinda múltiples posibilidades para el trabajo con las validaciones y manejo de errores en el cliente. Además permite la personalización de temas de estilos y provee el trabajo con una amplia configuración e intenso trabajo con las hojas de estilo CSS. Basa toda su funcionalidad en javascript a través de librerías YUI, jQuery, o haciendo uso de la librería nativa, así en tiempo de ejecución carga y crea todos los objetos HTML a través del uso intenso de DOM. Cuenta con dos licencias, una comercial y otra, código abierto. (18)

1.14 Lenguaje de Programación

PHP5

PHP (Hypertext Pre-processores) es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el lado del servidor. Puede ser desplegado en la mayoría de los servidores Web y en casi todos los sistemas operativos y plataformas sin costo alguno. Este lenguaje posee amplias ventajas como (19):

- ✓ Es un lenguaje multiplataforma.
- ✓ Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL.
- ✓ Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados extensiones).
- ✓ Posee una amplia documentación en su página oficial (www.php.net), entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- ✓ Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- ✓ Permite las técnicas de Programación Orientada a Objetos.
- ✓ Contiene una biblioteca nativa de funciones sumamente amplia e incluida.
- ✓ No requiere definición de tipos de variables.
- ✓ Tiene manejo de excepciones.

1.15 Servidor Web

Un servidor web básicamente, sirve contenido estático a un navegador, carga un archivo y lo sirve a través de la red al navegador de un usuario. Este intercambio es mediado por el navegador y el servidor que hablan el uno con el otro mediante HTTP, conocido por sus siglas en inglés hypertext transfer protocol.

Apache 2.0

Un servidor web Apache es un servidor HTTP de código abierto. Es un programa que se ejecuta continuamente desde una computadora (que será el llamado servidor) e interpreta las peticiones HTTP que recibe por parte de un cliente (un navegador de Internet) y las satisface.

Características principales de Apache 2.0:

- ✓ Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- ✓ Los módulos de Apache pueden escribirse para que se comporten como filtros que actúan sobre el flujo de contenidos tal y como salen del servidor o como son recibidos por el servidor. Posee mensajes de error en diferentes idiomas, tiene un soporte mejorado para las plataformas que no son tipo Unix y cuenta con la infraestructura necesaria para servir distintos protocolos.
- ✓ Apache trabaja con gran cantidad de Perl, PHP y otros lenguajes de script. También trabaja con Java y páginas javascript puro, teniendo todo el soporte que se necesita para tener páginas dinámicas.
- ✓ Servidor altamente configurable de diseño modular, siendo muy sencillo ampliar sus capacidades.
- ✓ Permite la creación de ficheros de log a medida del administrador, de este modo se puede tener un mayor control sobre lo que sucede en el servidor.
- ✓ Apache es una tecnología gratuita, de código fuente abierto.

1.16 Navegador Web

Un navegador o navegador web (del inglés, web browser) es un programa que permite visualizar la información que contiene una página web (ya esté alojada en un servidor dentro de la World Wide Web o en uno local). El navegador interpreta el código, HTML generalmente, en el que está escrita la página web

y lo presenta en pantalla permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces o hipervínculos. La funcionalidad básica de un navegador web es permitir la visualización de documentos de texto, posiblemente con recursos multimedia incrustados.

Mozilla Firefox 4.0

Mozilla Firefox es un navegador de software libre. Entre sus características se encuentran que presenta una forma rápida y eficiente de navegar por la web, que le permite abrir varias páginas en una misma ventana mediante el empleo de pestañas separadas. Mozilla Firefox además protege al usuario de la publicidad de ventanas emergentes no solicitadas. (20)

Firefox utiliza además a Firebug que es una extensión creada y diseñada especialmente para desarrolladores y programadores web. Es un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM), editar, monitorizar y depurar el código fuente, CSS, HTML y javascript de una página web de manera instantánea y online.

Conclusiones del capítulo

En este capítulo se abordaron varios temas que constituyen el fundamento teórico y tecnológico para la confección del procedimiento y para la automatización de pruebas unitarias a componentes en javascript en el contexto de una LPS. Además se realizó una pequeña descripción de la metodología y las tecnologías y herramientas que servirán para el desarrollo del sistema.

Se utiliza como metodología de desarrollo de software Open/Up, como herramienta CASE Visual Paradigm en su versión 6.1, lenguaje de modelado UML en su versión 2.0, lenguaje de programación PHP5, framework de desarrollo ExtJS 2.2, como entorno de desarrollo el IDE NetBeans en su versión 6.9, servidor web Apache en su versión 2.0,como navegador web Moxilla Firefox en su versión 4.0, tecnología del lado del cliente JavaScript y como herramienta para la automatización de las pruebas unitarias JsUnit.

CAPÍTULO 2: PROCEDIMIENTO PARA PRUEBAS UNITARIAS

Introducción

Las pruebas unitarias que se realizan al software, tienen un impacto significativo en la calidad del producto final, por esta razón, en el presente capítulo a partir de la fundamentación teórica, se ofrece una solución, para garantizar la realización de pruebas unitarias a los artefactos producidos por el desarrollador. Como propuesta de solución se ofrece un procedimiento, el que se describe en términos de entrada, actividades a desarrollar y salida. Se describen las actividades propuestas y sus principios de forma general.

2.1 Procedimiento para la realización de pruebas unitarias en el contexto de una LPS

El proceso de desarrollo de software tiene como objetivo satisfacer una necesidad planteada por el cliente. Para asegurar que se han alcanzado los niveles de calidad acordados es necesario realizar pruebas al producto de software a medida que se va construyendo. Las pruebas unitarias en una LPS de una organización deben examinar los activos de software básico. A diferencia de un solo proyecto de desarrollo de software, las responsabilidades de las pruebas pueden ser distribuidas en las distintas partes de la LPS, las mismas representa una actividad y un productor de artefactos, cuyos esfuerzos están reutilizados a través de una multitud de productos. En la siguiente ilustración (Ver ilustración 2.1) se resalta en rojo en qué lugar se encuentran las pruebas unitarias dentro del modelo de producción y organización de los procesos de una LPS de una organización.

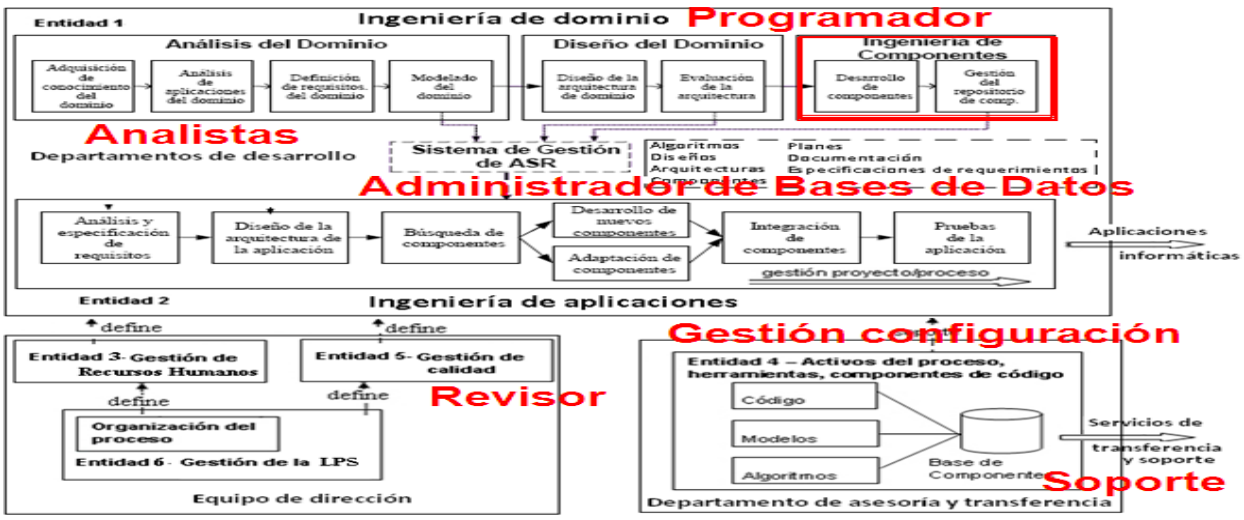

Ilustración 2.1 Modelo de producción y organización de una LPS
Fuente: Conf: Introducción a los nuevos modelos de Producción de Software. Dr.C Pedro Y. Piñero Pérez

Descripción del procedimiento

El procedimiento para realizar pruebas unitarias en el contexto de una LPS, definirá de forma detallada una secuencia de pasos que guiarán todo el desarrollo de las pruebas. Está estructurado en 3 fases, donde se describen las actividades a realizar y la documentación de entrada y salida que las conforman.

Alcance

El procedimiento va dirigido a realizar pruebas unitarias en el contexto de una LPS.

Objetivos

- ✓ Identificar las actividades para realizar pruebas unitarias en el contexto de una LPS.
- ✓ Describir las actividades para realizar pruebas unitarias en el contexto de una LPS.

Las pruebas unitarias desarrolladas en este procedimiento tienen como objetivo aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el fragmento de código debe satisfacer. El objetivo principal sería producir las piezas de código de la forma más eficiente y eficaz posible generando pruebas unitarias para las mismas que aseguren su correcto comportamiento.

Precondiciones

Para hacer uso del procedimiento se hace necesario contar previamente:

- ✓ Descripción de la especificación de Requisitos.
- ✓ Descripción de los Casos de Usos.
- ✓ Modelo de diseño.
    - Descripción de Clases del Diseño.
    - Descripción de los Casos de Uso del Diseño.

2.1.2 Fases del procedimiento

El procedimiento está estructurado en 3 fases:

- ✓ Fase de Análisis de Pruebas Unitarias.
- ✓ Fase de Diseño de Pruebas Unitarias.
- ✓ Fase de Ejecución de Pruebas Unitarias.

En los siguientes epígrafes que se muestran se describirá de forma detallada cada una de las fases definidas en el procedimiento.

2.1.2.1 Fase de Análisis de Pruebas Unitarias

En esta fase se analizará lo establecido en el contrato, que define las relaciones entre las clases y los clientes como un acuerdo formal, que expresa los derechos y obligaciones de cada parte, "estableciendo precondiciones y postcondiciones, la precondiciones comprende al cliente, donde define las condiciones bajo las cuales es legítima la llamada a una rutina, siendo una obligación del cliente y un beneficio para el proveedor, y la postcondición comprende a la clase, define las condiciones que debe asegurar la rutina al retornar, es un beneficio para el cliente y una obligación para el proveedor, sirviendo de guía para definir los casos de pruebas unitarias que se le realizarán al activo de software" (21). Además se definirán los trabajadores y los recursos necesarios para realizar este procedimiento. Recogiéndose todo en el Plan de Inicial de Pruebas Unitarias, en el cual se seleccionan y documentan una serie de elementos que aseguran el punto de partida de la siguiente fase, en aras de garantizar una mejor calidad del producto de software.

Fase de Análisis de Pruebas Unitarias		
Entradas	**Actividades**	**Salidas**
_Descripción de especificación de Requisitos. _ Descripción de los Casos de Usos. _Modelo de diseño. • Descripción de Clases del Diseño. • Descripción de los	_Definir la estrategia a seguir. • Flujo de trabajo. • Identificar los trabajadores a realizar las pruebas unitarias. • Seleccionar la herramienta.	_Plan de Inicial de Pruebas Unitarias.

Casos de Uso del Diseño.		

Observación: Fase de estricto cumplimiento.

Tabla 2.1 Fase Inicial de Pruebas Unitarias
Fuente: Investigación realizada

Estrategia a seguir

Para llevar a cabo el proceso de pruebas unitarias en el contexto de una LPS, se trazó el flujo de trabajo a seguir, en aras de organizar todo el procedimiento, donde se definen una serie de pasos bien planificados que detallarán la forma en que será probado el activo durante la realización de las pruebas unitarias, así como la secuencia de actividades de forma general a desarrollar. Además se definen los trabajadores responsables que participarán y las herramientas a utilizar.

1. **Flujo de Trabajo**

El flujo de trabajo se inicia cuando el desarrollador con un nivel básico en pruebas, analiza los artefactos de entrada y comienza a diseñar los casos de prueba unitaria, para implementarlos y posteriormente pasar a la ejecución de los mismos, haciendo uso de la herramienta seleccionada. En la medida que se detecten errores, estos serán registrados y corregidos, para posteriormente subir junto con el activo la documentación de las pruebas unitarias realizada, al repositorio de activos de la LPS.

Ilustración 2.2 Flujo de trabajo
Fuente: Investigación realizada

2. **Trabajadores**

Para determinar los trabajadores y sus responsabilidades, tómese en cuenta los criterios siguientes:

"Prueba unitaria es la más micro escala de las pruebas; para probar funciones particulares o módulos de código. Típicamente hechas por desarrolladores y no por probadores. (...)". (22)

"Mientras otras formas de probar pueden ser realizadas por probadores especializados, las pruebas unitarias deben ser escritas por el desarrollador (o una pareja de estos) que escriba cada clase, junto al código de la clase en sí. Un par de programadores es particularmente más efectivo, porque dos desarrolladores pueden pensar en un rango más amplio de escenarios de pruebas que uno". (1)

Teniendo en cuenta lo antes planteado y que se debe contar con un conocimiento básico de pruebas, se define para la aplicación del procedimiento de pruebas unitarias en el contexto de una LPS, dos trabajadores fundamentales:

Trabajadores	**Responsabilidades**
Desarrollador	Encargado de diseñar, implementar y ejecutar los casos de pruebas unitarias, así como registrar y corregir los errores.
Jefe de proyecto o el designado por el Grupo de Calidad	Supervisa que se hallan realizado las pruebas unitarias a cada activo, a través de la documentación registrada por el desarrollador, en caso de encontrar errores informar al desarrollador para que haga las correcciones pertinentes.

Tabla 2.2 Trabajadores y sus responsabilidades

Fuente: Investigación realizada

A continuación se muestra la relación de los trabajadores con las actividades más importantes a desarrollar (Ver ilustración 2.3).

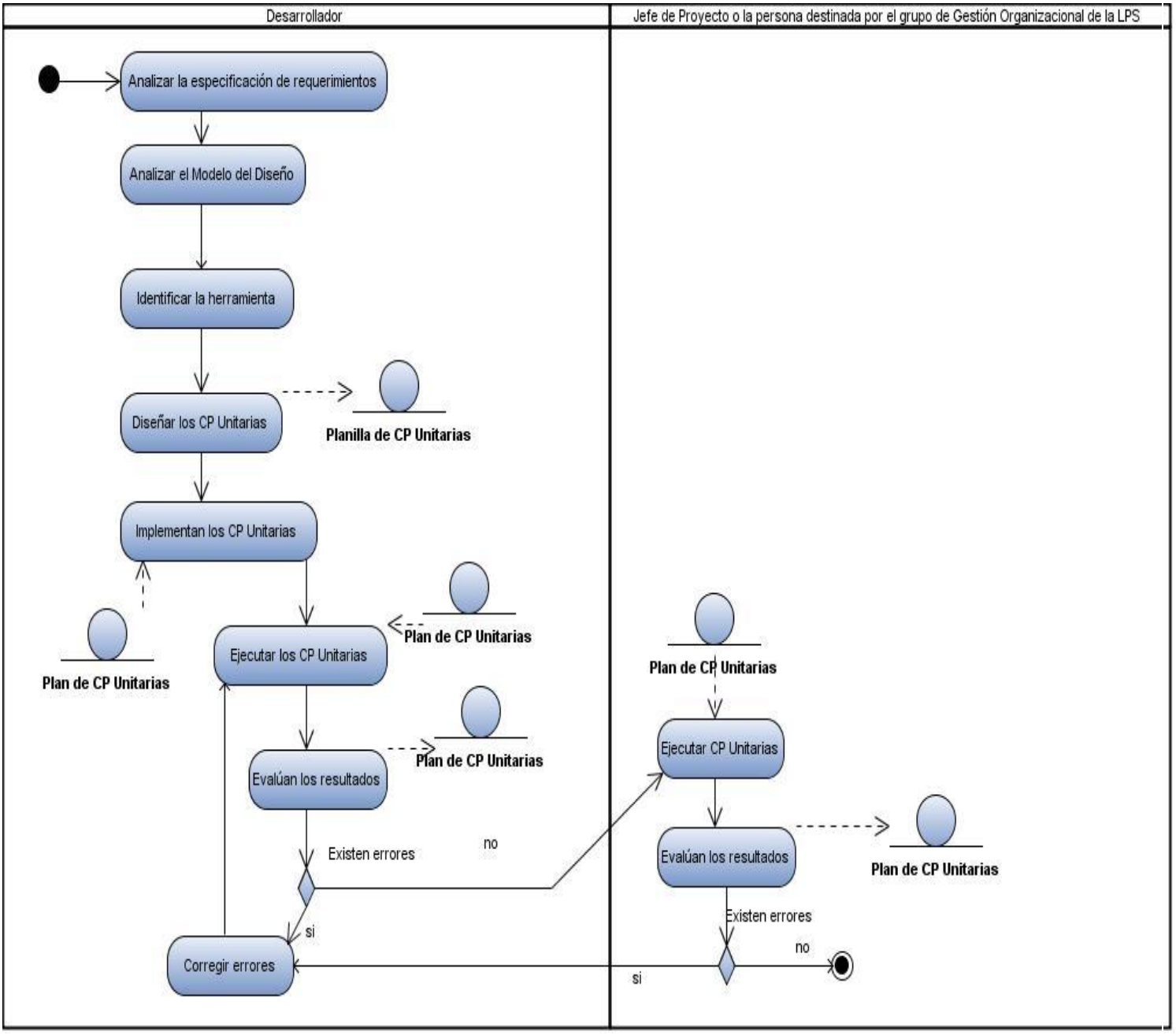

Ilustración 2.3 Relación de los trabajadores con las principales actividades
Fuente: Investigación realizada

3. Selección de la Herramienta

Para la selección de la herramienta, se debe hacer un análisis de una serie de elementos, en aras de seleccionar la que mayor beneficio aporte. La siguiente tabla que se muestra, servirá como guía para la selección. En la parte superior de la tabla se exponen una serie de elementos que se deben tener en cuenta.

Herramienta	Plataforma de Ejecución	Entorno de Aplicación	Factores Políticos	Calidad	Total
[1]					

Tabla 2.3 Selección de la herramienta
Fuente: Investigación realizada

Descripción de cada uno de los campos de la tabla:

- ✓ Herramienta: nombre de la herramienta.
- ✓ Plataforma de Ejecución: sí se cuenta con la plataforma donde se ejecutará la herramienta, 1 punto.
- ✓ Entorno de aplicación: sí se cuenta con el entorno de aplicación de la herramienta, 1 punto.
- ✓ Factores políticos: sí debido a la obligatoriedad de comprar a determinados fabricantes, se puede utilizarla 1 punto.
- ✓ Calidad: esta columna será una sumatoria, entiéndase, sí se cuenta con una adecuada documentación 1 punto, del caso contrario 0 puntos, si tiene frecuencia a fallos -1 punto, del caso contrario 0 puntos, riesgos de que la herramienta provoque fallos en otras partes del entorno -1 punto, del caso contrario 0 puntos.
- ✓ Total: suma horizontal de los puntos.

Planilla Inicial de Pruebas Unitarias

El documento más importante de la fase inicial es el Plan Inicial de Pruebas Unitarias, ya que en él se recoge la estrategia a seguir a lo largo del procedimiento, entiéndase el flujo de trabajo, los trabajadores con sus responsabilidades, así como la herramienta que automatizará todo el proceso de pruebas unitarias.

Ver Anexo 1: Planilla del Plan Inicial de Pruebas Unitarias.

2.1.2.2 Fase de Diseño de Pruebas Unitarias

La Fase de Diseño comprende la descripción de los casos de prueba unitarias, donde se describe como diseñar dichos casos de prueba para poder ejecutarlos, y así comprobar el correcto funcionamiento de cada una de las funcionalidades o métodos y el cumplimiento de la lógica de negocio.

Fase de Diseño de Pruebas Unitarias		
Entradas	**Actividades**	**Salidas**
_ Plan Inicial de Pruebas Unitarias. __Descripción de especificación de Requisitos. _ Modelo de diseño. • Descripción de Clases del Diseño. • Descripción de los Casos de Uso del Diseño. _ Código Fuente	_Diseñar los casos de prueba unitaria. _Definir el método a utilizar. _Implementar los casos de pruebas unitarias.	_ Plan de Casos de Prueba Unitarias. _Script de Prueba.
Observaciones: Se deben diseñar los casos de prueba unitaria.		

Tabla 2.4 Fase de Diseño de Pruebas Unitarias
Fuente: Investigación realizada

1. **Descripción de los Casos de Prueba Unitarias**

En esta actividad se define el nivel más bajo de cada una de las funcionalidades a probar, se realiza una descripción detallada de cada uno de los casos de prueba a realizar y se propone la Planilla de Casos de Pruebas Unitarias, para que sea utilizada en el desarrollo del procedimiento.
Ver Anexo No2.Planilla de Casos de Pruebas Unitarias

Planilla de Casos de Pruebas Unitarias

La planilla tiene en su encabezado el nombre del proyecto y del componente que se va a probar, y de este último su descripción general y las condiciones que deben cumplirse para su ejecución, así como la

versión en la que se encuentra la prueba unitaria. Luego se detalla en una tabla el nombre de las funcionalidades que se van a probar del activo, así como una pequeña descripción de las mismas.

Funcionalidad	Descripción de la funcionalidad
< Nombre de la funcionalidad>	*<Descripción de la Funcionalidad.>*

Tabla 2.5 Funcionalidades a probar
Fuente: Investigación realizada

En la siguiente tabla se detallan los casos de prueba para las funcionalidades descritas en la tabla anterior. Se recogen los resultados obtenidos después de aplicar aquellos métodos que ofrece la herramienta seleccionada para comprobar el correcto funcionamiento de las funcionalidades.

Descripción de Caso de Prueba Unitaria					
Iteración:< iteración que se ejecuta >					
Descripción de la Prueba Unitaria:< descripción breve de la prueba >					
Responsable:< nombre del responsable >					
Funcionalidad	**Método utilizado**	**Recibe**	**Respuesta esperado del método**	**Respuesta real de la prueba**	**Observaciones**
<Nombre de la funcionalidad >	*< se escribe el nombre del método a utilizar>*	*< parámetros que recibe la funcionalidad >*	*<se escribe la respuesta esperado del método >*	*< se escribe la respuesta real de la prueba >*	*< claras y precisas>*

[2]					
Observaciones: Si no coinciden los últimos dos campos, existen errores hay que corregir.					

Tabla 2.6 Descripción de la Prueba Unitaria
Fuente: Investigación realizada

Descripción de cada uno de los campos de la tabla:

- ✓ Iteración: iteración en la que se encuentra la ejecución de la prueba unitaria.
- ✓ Descripción de la Prueba Unitaria: descripción breve de la prueba unitaria a ejecutar.
- ✓ Responsable: nombre del trabajador encargado de realizar la prueba unitaria.
- ✓ Funcionalidad: nombre de la funcionalidad a probar.
- ✓ Método utilizado: nombre del método a utilizar, de los que propone la herramienta seleccionada para hacer la prueba unitaria.
- ✓ Recibe: parámetros que recibe la funcionalidad a probar.
- ✓ Respuesta esperada del método: respuesta esperada del método.
- ✓ Respuesta real de la prueba: resultado de la prueba una vez ya ejecutada.
- ✓ Observaciones: claras y precisas.

2. Implementación de los Casos de Pruebas

Una vez diseñados los casos de prueba unitaria, es posible implementar cada prueba, teniendo en cuenta que cubran una serie de criterios de cobertura, que adicionalmente permitan la identificación de los casos de prueba redundantes. La cobertura de las pruebas unitarias está dada por la medida de cuanta parte del código es probada. Significa inspeccionar si cada línea de código bajo prueba es ejecutada al menos una vez en alguna prueba. Es importante conocer si las pruebas brindan buena cobertura. Es inútil -o al menos, no se podrían aprovechar al máximo todas las bondades de las pruebas, si se tiene una suite de pruebas que no prueban todo el código.

Una suite de pruebas debe contar siempre con instrumentos que permitan conocer si las pruebas han olvidado algo por probar. Así, por lo menos se conocerá donde hay mayor probabilidad de que los

usuarios o probadores encuentren errores. Entre los tipos básicos de cobertura que se deben lograr con las pruebas están:

- ✓ **Cobertura de sentencias**

Comprueba el número de sentencias ejecutables que se han ejecutado.

- ✓ **Cobertura de decisiones**

Comprueba el número de decisiones ejecutadas, considerando que se ha ejecutado una decisión cuando se han recorrido todas sus posibles ramas (la que la hace *true* y la que la hace *false*, pero también todas las posibles ramas de un *switch*).

- ✓ **Cobertura de condiciones**

Comprueba el número de condiciones ejecutadas, entendiendo que se ha ejecutado una condición cuando se han ejecutado todas sus posibles ramas.

- ✓ **Cubrimiento de bucles**

Comprueba el número de bucles que han sido ejecutados cero veces (excepto para bucles *do while*), una vez y más de una vez.

- ✓ **Cobertura de operadores relacionales**

Comprueba si se han ejecutado los valores límite en los operadores relacionales (>, <, >=, <=), ya que se asume la hipótesis de que estas situaciones son propensas a errores.

- ✓ **Cobertura de tablas**

Comprueba si se ha hecho referencia a todos los elementos de los arreglos.

2.1.2.3 Fase de Ejecución de Pruebas Unitarias

En esta tercera y última fase se realiza la ejecución de los casos de pruebas ya diseñados e implementados y se evalúa el resultado, con la ayuda de la herramienta seleccionada que automatizará la prueba.

Fase de Ejecución de Pruebas Unitarias		
Entradas	**Actividades**	**Salidas**
_Plan de los Casos de Pruebas Unitarias. _Código fuente.	_Ejecutar los casos de pruebas unitarias. _Evaluar los resultados.	_ Plan de Casos de Pruebas Unitarias. (actualizado)

_ Descripción de especificación de Requisitos. _Script de Prueba **Observaciones:**	_Registrar los resultados en la Planilla de Casos de Pruebas Unitarias.	

Tabla 2.7 Fase de Ejecución de Pruebas unitarias
Fuente: Investigación realizada

1. **Ejecución de las Pruebas Unitarias**

En esta fase el desarrollador ejecuta las pruebas una vez ya implementadas. Hace uso de la herramienta seleccionada y los casos de prueba diseñados por él anteriormente, teniendo en cuenta no sobrepasar la fecha límite de ejecución. Para comenzar se recomienda realizar una validación e inspección de los casos de prueba diseñados, verificando que se han contemplado todas las pruebas necesarias, que no existen pruebas duplicadas o implícitas en otras. Luego se ejecutan las pruebas, los resultados se van registrando, para que en caso de encontrar fallos puedan posteriormente ser corregidos, y se vuelven a ejecutar hasta no encontrar errores, para que al final se tenga la certeza de que la funcionalidad está correcta. Se actualiza la Planilla de Casos de Pruebas Unitarias, y de este modo se tiene un inventario de todos los casos de pruebas ejecutados, las iteraciones de los mismos y las incidencias asociadas,

Para esta tarea, en general, se recomienda mantener un repositorio de pruebas implementadas, con el fin de mantener una estructura de almacenamiento que permita su reutilización.

Evaluación de los resultados

Una vez ejecutadas las pruebas unitarias, el siguiente paso es evaluar el resultado. El objetivo de esta evaluación, es ver la magnitud de los errores encontrados, para llegar a conclusiones y recomendaciones tras la ejecución de las pruebas unitarias, así como la toma de decisiones acerca del número de iteraciones a realizar, y ver cuán tan cerca está la funcionalidad probada de lo establecido en el contrato. En fin se evalúan los resultados de los esfuerzos de prueba, tales como la cobertura del caso de prueba, de código y el estado de los defectos.

Para mostrar la evaluación de la ejecución de las pruebas unitarias realizadas se detalla a continuación una tabla en la que se recogen todos los posibles errores encontrados cuando se ejecutaron las pruebas unitarias:

Reporte de errores			
Errores encontrados: *<cantidad de errores encontrados>*			
Funcionalidad: *<nombre de la funcionalidad probada>*			
Responsable: *<nombre del responsable>*			
Método	**Error Encontrado**	**Clasificación**	**Observaciones**
<tipo de sentencia>	*<descripción del error encontrado>*	*<prioridad de impacto >*	*<clara y precisa >*

Tabla 2.8 Reporte de Errores
Fuente: Investigación realizada

Para la evaluación del error se definirá su prioridad de acuerdo al impacto de los mismos sobre el componente:

- ✓ Crítico (debe ser corregido inmediatamente)
- ✓ Alto
- ✓ Normal
- ✓ Bajo
- ✓ Satisfactoria (en caso de no encontrar errores).

La evaluación del error se incluirá dentro del Plan de Casos de Prueba de Unidad.

Conclusiones del capítulo

En este capítulo se realizó la descripción del procedimiento de pruebas unitarias en el contexto de una LPS. A lo largo del procedimiento se describieron las actividades que se definieron en cada una de sus fases, para desarrollar un proceso correcto de pruebas unitarias. Se espera que en los resultados de su

aplicación se demuestre su fiabilidad, partiendo de que no existe una iniciativa precedente a la que se expone en esta investigación para la aplicación de las pruebas unitarias en el contexto de una LPS.

CAPÍTULO 3: APLICACIÓN DEL PROCEDIMIENTO Y ANÁLISIS DE RESULTADOS

Introducción

En este capítulo, se realiza el modelo de dominio para el levantamiento de los principales conceptos a tratar en el sistema. Se le hace una mejora a la interfaz gráfica de la herramienta seleccionada, además se le adicionan nuevas funcionalidades especificándose en los requisitos funcionales y no funcionales de la misma. Para posteriormente aplicar el procedimiento haciendo uso del sistema con su interfaz mejorada y analizar los resultados, cumpliendo todas las actividades descritas en cada una sus fases, demostrando la fiabilidad del mismo.

3.1 Propuesta de Solución

Como valor añadido al procedimiento se propone el desarrollo del sistema que utilizará como motor el JsUnit, al que se le realizó una mejora de su interfaz gráfica y se le adicionaron nuevas funcionalidades. Dicho sistema servirá de apoyo al procedimiento descrito en el capítulo anterior, pues automatiza las pruebas unitarias a componentes en javascript en el contexto de una LPS, ya que permite la ejecución de los casos de prueba ya implementados, detectando posibles errores, contribuyendo así, al correcto funcionamiento de la funcionalidad probada, garantizando el concepto de calidad total definido en DATEC.

3.2 Modelo del Dominio

Un modelo del dominio es una representación visual de las clases conceptuales u objetos del mundo real en un área de interés. Utilizando la notación UML se representa con un conjunto de diagramas de clases en los que no se define ninguna operación. Pueden mostrar objetos del dominio o clases conceptuales, asociaciones entre las clases conceptuales y sus atributos. Es una representación de las clases conceptuales del mundo real significativas en un dominio del problema, no de componentes del software, aunque algunos objetos del modelo pueden terminar siéndolo. (23)

3.2.1 Diagrama conceptual del Modelo del Dominio

Durante el análisis del sistema no se observan los procesos claramente definidos por lo que se realiza el modelo de dominio, el cual tiene como objetivo fundamental comprender y describir los

conceptos más importantes dentro del contexto del sistema, es una representación visual del entorno real del proyecto.

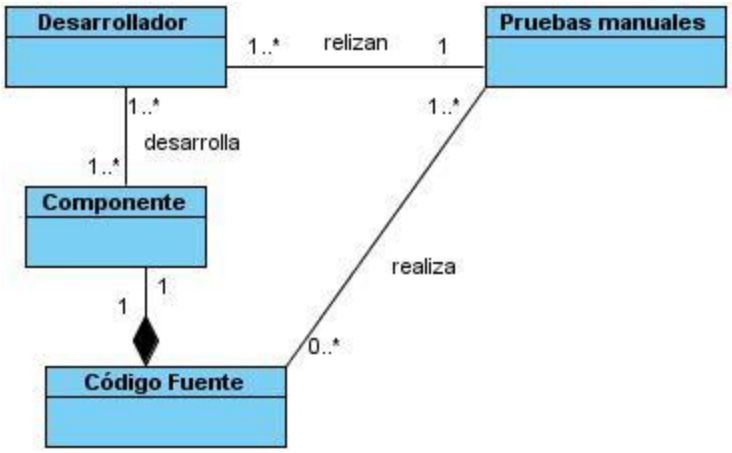

Ilustración 3.1 Modelo del Dominio
Fuente: Investigación realizada

3.2.2 Definición de clases del Modelo del Dominio

Desarrollador: Escribe, depura y mantiene el código fuente.

Componente: "Una pieza [de software] funcional que es liberada independientemente [de otras] y que proporciona acceso a sus servicios a través de sus interfaces". (24)

Código Fuente: Conjunto de líneas de texto que constituyen las instrucciones que debe seguir la computadora para ejecutar dicho programa, escritos en un lenguaje de programación específico.

Pruebas Manuales: involucra las operaciones del sistema bajo condiciones controladas, evaluando los resultados, de forma manual en aras de encontrar posibles errores y fallos.

3.3 Especificación de Requisitos de Software

Los requisitos de software describen los servicios que ha de ofrecer el sistema y las restricciones asociadas a su funcionamiento, no son más que propiedades o restricciones determinadas de forma precisa que deben satisfacerse. El Instituto de Ingenieros Electricistas y Electrónicos (IEEE) define un requisito como:

1. Una condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo.

2. Una condición o capacidad que debe tener un sistema o un componente de un sistema para satisfacer un contrato, una norma, una especificación u otro documento formal.
3. Una representación en forma de documento de una condición o capacidad como las expresadas en 1) o en 2).

En este flujo de trabajo se establece que debe hacer exactamente el sistema que se construya. Los requisitos han de ser: claros y concretos, evitando imprecisiones y ambigüedades; concisos, completos y consistentes.
Los requisitos pueden clasificarse, por lo general se dividen en dos grupos: requisitos funcionales y requisitos no funcionales.

3.3.1 Requisitos Funcionales

Los requisitos funcionales (RF) son capacidades o condiciones que el sistema debe cumplir. Además, se mantienen invariables sin importar con que propiedades o cualidades se relacionen. Definen las funciones que el sistema será capaz de realizar. Los RF añadidos a la herramienta son los siguientes:

RF1: Gestionar Caso de Prueba.
El sistema brinda la posibilidad de insertar, buscar, listar, modificar y eliminar casos de prueba.

RF1.1 Insertar Caso de Prueba.
El sistema brinda la posibilidad de insertar casos de prueba.
Entrada: Dirección de la ubicación del caso de prueba.
Salida: Caso de prueba insertado.

RF1.2 Buscar y Visualizar Caso de Prueba.
El sistema brinda la posibilidad de buscar y visualizar el Caso de Prueba.
Entrada: Dirección de la ubicación del caso de prueba.
Salida: Visualización del Caso de Prueba.

RF1.3 Modificar Caso de Prueba.
El sistema brinda la posibilidad de modificar el caso de prueba.
Entrada: Dirección de la ubicación del caso de prueba.
Salida: Caso de prueba modificado.

RF1.4 Listar Casos de Prueba.

El sistema brinda la posibilidad de listar los casos de prueba.

Salida: Lista con todos los casos de prueba almacenados en el sistema.

RF1.5 Eliminar Caso de Prueba.

El sistema brinda la posibilidad de eliminar casos de prueba.

Entrada: Dirección de la ubicación del caso de prueba.

Salida: Caso de prueba eliminado.

RF2: Ejecutar Caso de Prueba.

El sistema permite la ejecución de casos de prueba.

Entrada: Dirección de la ubicación del caso de prueba.

Salida: Caso de prueba ejecutado.

3.3.2 Requisitos no Funcionales

Los requisitos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Estas propiedades o cualidades se refieren a las características que hacen al producto atractivo, usable, rápido o confiable. Para la herramienta propuesta se definen los siguientes RNF:

Usabilidad

RNF 1: El sistema podrá ser utilizado por cualquier persona que posea conocimientos básicos en el manejo de la computadora y de un ambiente Web en sentido general.

Portabilidad

RNF 2: El sistema debe ser multiplataforma.

Interfaz de Usuario

RNF 3*:* El sistema deberá poseer una interfaz web sencilla, amigable, lo más atractiva, clara e intuitiva posible para el usuario.

RNF 4: Las interfaces de usuario a implementar deben ser diseñadas para su utilización en un ambiente web.

RNF 5: La aplicación deberá ser accesible remotamente a través de la web utilizando navegadores estándares como Mozilla Firefox.

Restricciones en el diseño y la implementación

RNF 6: Se hace uso de la herramienta JsUnit.

RNF 7: Se utiliza el framework Ext Js versión 2.2 para el desarrollo de la interfaz.

RNF 8: Utilizar como servidor Web Apache.

Hardware

RNF 9: Procesador Pentium II o superior, 256 MB de memoria RAM o superior.

Software

RNF 10: En el lado del cliente debe existir un navegador Mozilla Firefox. En el lado del servidor debe estar instalado el servidor web Apache.

Requerimientos Legales

RNF 11: Se utilizan herramientas de software libre y herramientas de las cuales se posee licencia para su uso y distribución.

3.4 Modelo de Casos de Uso del Sistema

El modelo de casos de uso del sistema representa gráficamente la relación entre los actores y los casos de uso (CU). Los actores son terceros fuera del sistema que interactúan con él (puede ser cualquier persona, individuo, grupo, entidad, organización, máquina o sistema de información externo; con los que el sistema interactúa) y los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores.

3.4.1 Actor del Sistema

Actor	Descripción
Desarrollador	Persona que interactúa con el sistema para realizar las pruebas unitarias a su componente.

Tabla 3.1 Actor del Sistema

Fuente: Investigación realizada

3.4.2 Casos de Uso del Sistema

Descripción textual del caso de uso Ejecutar Caso de Prueba

Caso de uso	Ejecutar Caso de Prueba
Actores	Desarrollador
Resumen	El caso de uso se inicia cuando el desarrollador inserta el caso de prueba en el sistema y lo ejecuta finalizando así el caso de uso.
Precondiciones	Prueba implementada al componente. Componente a probar.
Referencias	RF2
Prioridad	Crítica
Flujo Normal de Eventos	
Sección "Ejecutar Casos de Prueba"	
Acción del Actor	**Respuesta del Sistema**
1. Una vez que el desarrollador implementa el caso de prueba, coloca la dirección física del caso de prueba y da clic en el botón cargar.	2. El sistema carga el caso de prueba y lo muestra.
3. El desarrollador da clic en el botón guardar.	4. El sistema guarda el caso de prueba en la herramienta y actualiza la lista de casos de prueba.
5. El desarrollador da clic en el botón ejecutar.	6. El sistema ejecuta el caso de prueba y muestra en el panel inferior los errores con su descripción, la cantidad de corridas, las fallas en caso de encontrar alguna, y un log con el historial registrando todas las actividades realizadas con el sistema en la ejecución de los casos de prueba, terminando así el caso de uso.
Flujos Alternos	

Flujo Alterno al paso 1 ”No encontrar el Caso de Prueba”	
	2.1 El sistema muestra un mensaje diciendo que “verifique si la dirección es correcta” o “verifique si la página es un caso de prueba”.
Flujo Alterno al paso 5 ”Detener ejecución del CP”	
5.1 Si el desarrollador no desea ejecutar el caso de prueba da clic en el botón detener.	5.2 El sistema detiene el caso de prueba, y se finaliza así el caso de uso.
Poscondiciones	Caso de Prueba ejecutado.

Tabla 3.1 Descripción textual del CU Ejecutar Casos de Prueba

Fuente: Investigación realizada

Descripción textual del caso de uso Gestionar Caso de Prueba

Caso de uso	Gestionar Casos de Prueba
Actores	Desarrollador
Resumen	El caso de uso se inicia cuando el desarrollador decide insertar, eliminar, modificar, buscar y visualizar, o listar casos de prueba. Finaliza una vez que se haya ejecutado una de estas acciones.
Precondiciones	Prueba implementada al componente. Componente a probar.
Referencias	RF1
Prioridad	Crítica
Flujo Normal de Eventos	
1. Si el desarrollador solicita la interfaz correspondiente a la Gestión de los Casos de Prueba para insertar, eliminar,	2. El sistema muestra la interfaz correspondiente: a. Si desea insertar un caso de

modificar, buscar y visualizar, o listar.	prueba ir a la sección "Insertar Caso de Prueba". b. Si desea buscar o visualizar un caso de prueba ir a la sección "Buscar o Visualizar Caso de Prueba". c. Si desea eliminar un caso de prueba ir a la sección "Eliminar Caso de Prueba".
Sección "Insertar Casos de Prueba"	
Acción del Actor	**Respuesta del Sistema**
1. Una vez que el desarrollador implementa el caso de prueba coloca la dirección física del caso de prueba en el sistema y da clic en el botón cargar.	2. El sistema carga el caso de prueba y lo visualiza.
3. El desarrollador da clic en el botón cargar.	4. El sistema inserta el caso y actualiza la lista de casos de prueba.
Sección "Buscar y Visualizar Casos de Prueba"	
1. El desarrollador da clic en uno de los casos de prueba almacenados en el sistema.	2. El sistema visualiza los detalles del caso de prueba seleccionado.
Sección "Modificar Casos de Prueba"	
1. El desarrollador selecciona el caso de prueba a modificar.	2. El sistema visualiza el caso de prueba seleccionado.
3. El desarrollador realiza las modificaciones pertinentes y da clic en la acción guardar.	4. El sistema modifica el caso de prueba, guarda los cambios y actualiza la lista de casos de prueba.
Sección "Eliminar Casos de Prueba"	
1. El desarrollador selecciona el caso de	

prueba a eliminar.	
2. El desarrollador da clic en el botón eliminar.	3. El sistema elimina el caso de prueba seleccionado.
Flujos Alternos	
Flujo alterno a la sección "Insertar Casos de Prueba" al paso 2	
	2.1 El sistema muestra un mensaje diciendo que "verifique si la dirección es correcta" o "verifique si la página es un caso de prueba".

Tabla 3.2 Descripción textual del CU Gestionar Casos de Prueba
Fuente: investigación realizada

3.4.3 Diagrama de Casos de Uso del Sistema

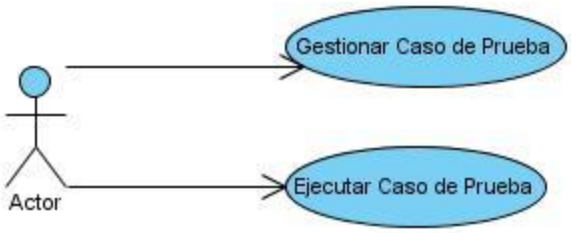

Ilustración 3.2 Diagrama de CUS
Fuente: Investigación realizada

3.5 Diagrama de Clases del Diseño

Un diagrama de clase del diseño es un diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos, operaciones y las relaciones existentes entre ellos. Contienen la siguiente información:

- ✓ Clases asociadas
- ✓ Métodos
- ✓ Navegabilidad
- ✓ Dependencias

A continuación se muestra el diagrama de clases del diseño correspondiente:

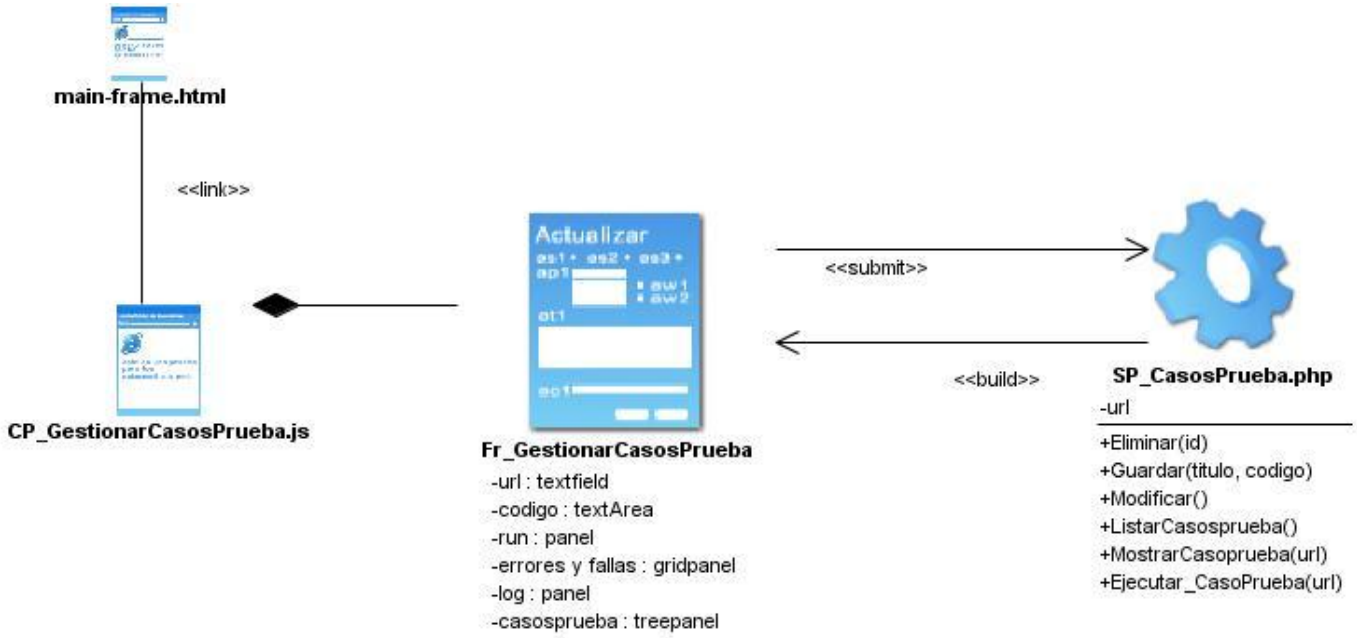

Ilustración 3.3 Diagrama de Clases del Diseño
Fuente: Investigación realizada

3.6 Diagrama de Componentes

Un diagrama de componentes es modelado por el arquitecto del sistema y representa gráficamente como un sistema de software es dividido en componentes mostrando las dependencias existentes entre estos. El diagrama contiene componentes, interfaces, relaciones entre ellos y puede contener paquetes utilizados para agrupar elementos del modelo; mostrando las dependencias lógicas entre los componentes del software.

A continuación se muestra el diagrama de componentes correspondiente:

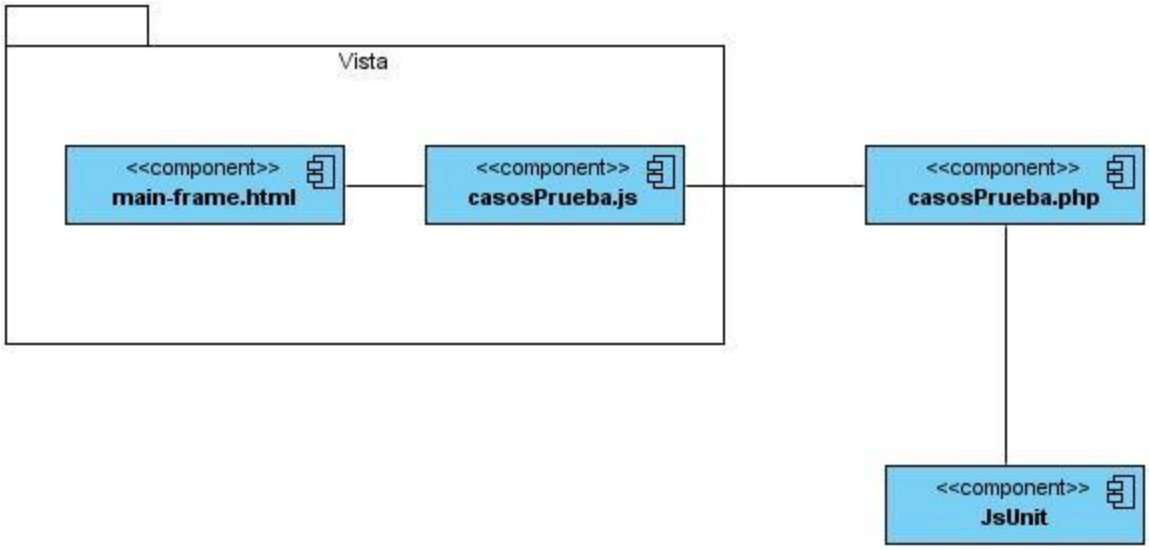

Ilustración 3.4 Diagrama de Componentes
Fuente: Investigación realizada

En el diagrama de componentes anterior se puede observar las dependencias que poseen al motor JsUnit el resto de los componentes, las cuales se describen brevemente a continuación:

- ✓ El componente JsUnit interactúa con el componente casosPrueba.php para comunicarse con la vista principal.
- ✓ El componente casosPrueba.php es quien controla la comunicación del componente JsUnit con el componente vista principal.
- ✓ El componente vista es la interfaz que se comunica con el componente casosPrueba.php para poder comunicarse con el componente JsUnit.

3.7 Descripción del sistema

El sistema DATEST, permite automatizar las pruebas unitarias al código javascript. Cuenta con una interfaz amigable y de fácil comprensión para el usuario. La misma está estructurada en tres partes fundamentales, el panel izquierdo superior (I) donde se muestra un árbol con los casos de pruebas almacenados en la herramienta, una vez que ya fueron ejecutados, para que sirva de apoyo a los demás desarrolladores que vayan a realizar sus pruebas unitarias. El panel derecho superior (II) donde se introduce la url del caso de prueba del componente a probar, además existe un área donde se muestra al

usuario el caso de prueba a realizar y cuenta con una serie de botones que permiten ejecutar, detener el caso de prueba una vez que se esté ejecutando, y guardar que ofrece la posibilidad al desarrollador de almacenar su caso de prueba unitaria en el sistema, para que sirva de apoyo a otros desarrolladores, permitiendo así la reutilización. El panel derecho inferior (III) donde se muestra los posibles errores y fallas una vez ejecutada la prueba. (Ver Ilustración No.3.5)

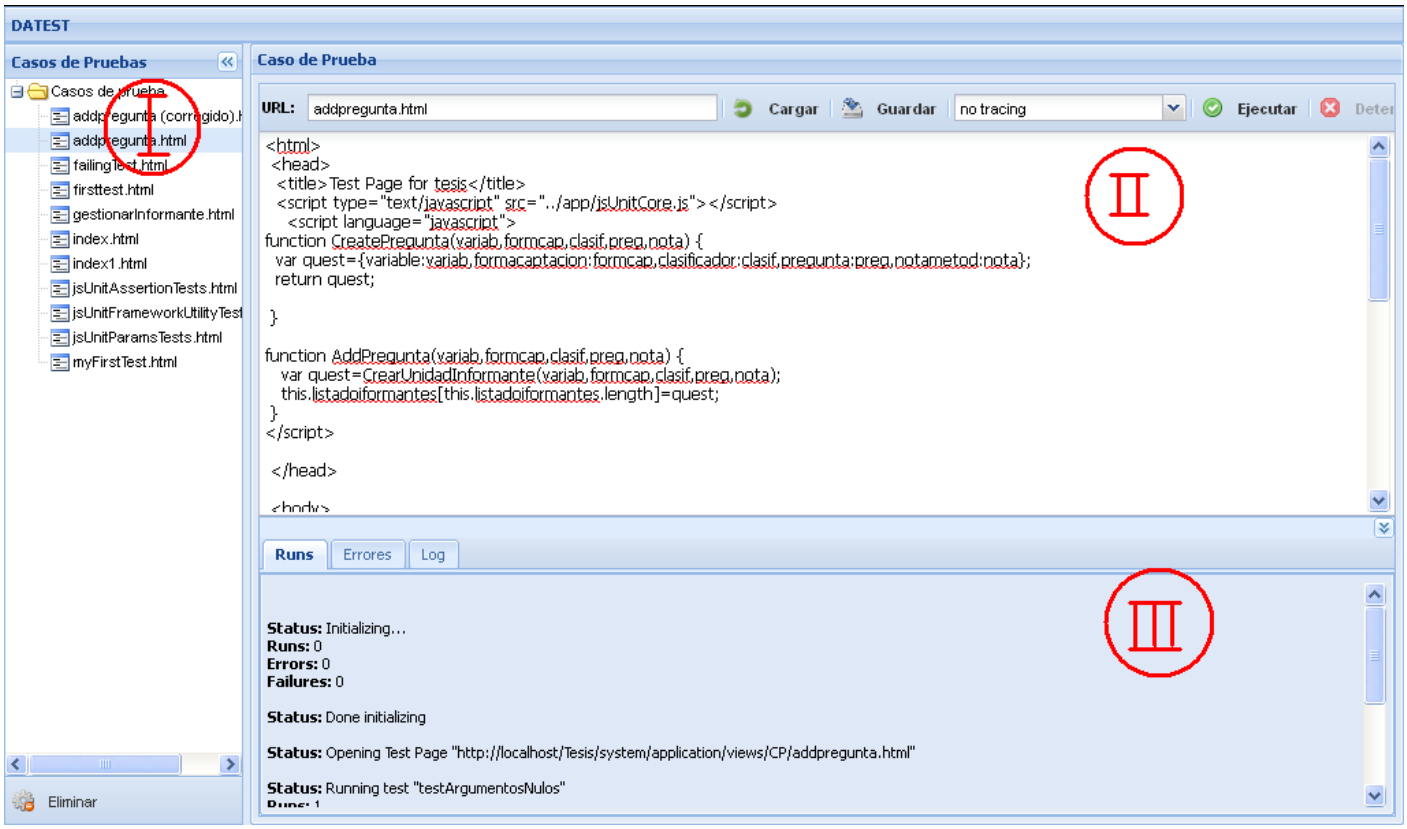

Ilustración 3.5 Interfaz Gráfica de DATEST
Fuente: Investigación realizada

DATEST realiza las pruebas unitarias mediante los Tests Functions (funciones de prueba), las que se encuentran implementadas en la clase jsUnitCore.js. Todos estas funciones de prueba o assert, tienen su contraparte o sea lo contrario a cada uno y se le pasan tres parámetros (mensaje, valor esperado, valor real). Dichos assert se describirán en la tabla que se muestra a continuación:

Método	Descripción
function assertTrue()	Verifica que el resultado de una funcionalidad sea true. ¨Lanza una jsUnit.Failure si el valor resultante no es true¨,¨Lanza un JsUnitInvalidAssertionArgument si los valores dados no son boolean o si la cantidad de argumentos pasados es incorrecta¨
function assertFalse()	Verifica que el resultado de una funcionalidad sea false. ¨Lanza una jsUnit.Failure si el valor resultante no es false¨,¨Lanza un JsUnitInvalidAssertionArgument si los valores dados no son boolean o si la cantidad de argumentos pasados es incorrecta¨
function assertEquals()	Verifica que dos valores son iguales utilizando el operador (===).¨Lanza una jsUnit.Failure si el los valores no son iguales¨,¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨
function assertNotEquals()	Verifica que dos valores no son iguales utilizando el operador (===).¨Lanza una jsUnit.Failure si el los valores son iguales¨,¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨
function assertNull()	Verifica que un valor es nulo.¨lanza una jsUnit.Failure si el valor no es nulo¨,¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨
function assertUndefined()	Verifica que un valor es indefinido¨lanza una jsUnit.Failure si el valor no es indefinido¨, ¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨.

function assertNaN()	Verifica que un valor No es un Número (Not a Number).¨Lanza una jsUnit.Failure si el valor es un número¨,¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨.
function assertObjectEquals()	Verifica que un objeto es igual a otro utilizando el operador (===) para sus primitivas y sus contrapartes incluso descendiendo a las colecciones de objetos y llamando este método para cada elemento. ¨Lanza una jsUnit.Failure si el valor actual no es igual al esperado¨, ¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨.
function assertArrayEquals()	Verifica que dos arreglos son iguales, primero verificando que son arreglos y luego verificando sus elementos aplicando assertObjectEquals ().¨Lanza una jsUnit.Failure si el valor actual no es igual al esperado¨, ¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨.
function assertHTMLEquals()	Comprueba si un valor es el mismo que una cadena HTML mediante la estandarización de los dos y para comparar el resultado de la igualdad. La estandarización se realiza con carácter temporal, la creación de un DIV, estableciendo el innerHTML del DIV a la cadena, y pidiendo la parte posterior innerHTML.¨Lanza una jsUnit.Failure si el valor estandarizado actual no es igual al valor estandarizado esperado¨, ¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨.
function assertContains()	Verifica si una colección contiene un valor. ¨Lanza una jsUnit.Failure si el valor no se encuentra dentro de la colección¨, ¨Lanza un JsUnitInvalidAssertionArgument si la cantidad de argumentos pasados es incorrecta¨.
function assertArrayEqualsIgnoring Order()	Verifica si dos arreglos son iguales sin importar el orden.

Tabla 3.3 Métodos para las Pruebas Unitarias

Fuente: Herramienta JsUnit

La mejor forma de entender el funcionamiento de cada uno de estos métodos es utilizándolos.

3.9 Aplicación del procedimiento

Para la aplicación del procedimiento se escogieron varios componentes que forman parte del proyecto Sistema Integrado para la Gestión Estadística (SIGE), desarrollado en conjunto con la Organización Nacional de Estadística (ONE) de forma tal que contempla el marco metodológico y operativo de esta. Como sistema está integrado por módulos altamente especializados y cohesivos que permiten la gestión de los centros informantes, la elaboración del Sistema Estadístico Nacional (SEN) para realizar la captación de los datos estadísticos, la digitalización y validación de la información recolectada, y su posterior consulta y análisis.

Uno de los componentes al que se le realizaron las pruebas unitarias es Gestionar Unidades de Observación el cual cuenta con los siguientes requisitos funcionales (Fuente: Especificación de Requisitos de software PATSI-SIGE):

RF4: Gestionar Unidades de Observación.
Descripción: El sistema debe permitir la gestión de las unidades de observación en sus diferentes estados. Para este requisito, Provincia, Municipio, Registro, Código, Nombre y Fecha de la unidad de observación constituyen campos obligatorios.

RF4.1 Agregar unidad de observación.
Descripción: El sistema debe permitir agregar una unidad de observación.
Entradas: Provincia (Menú desplegable), Municipio (Menú desplegable), Registro (Menú desplegable), Código (Alfanumérico de 5 a 15 caracteres (Sólo acepta cadenas de números de 0-9)), Nombre (Alfanumérico de 1 a 255 caracteres), Fecha (Campo fecha. Formato: YYYY/MM/DD), Nota Metodológica (Alfanumérico de longitud ilimitada), Teléfono (Alfanumérico de 1 a 15 caracteres (Sólo acepta cadenas de números de 0-9)), Correo (Alfanumérico de 5 a 100 caracteres), Dirección (Alfanumérico de 1 a 512 caracteres) y Estado (Numérico de 1 a 5 dígitos).

Objetivo de la aplicación del procedimiento

Se le realizarán pruebas unitarias automatizadas a cada una de las funcionalidades del componente haciendo uso del sistema DATEST**,** y contando con el procedimiento descrito en el capítulo 2, que guiará todo el proceso de pruebas unitarias. En aras de verificar que las funcionalidades del componente Gestionar Unidades de Observación, están funcionando correctamente. Se comparan los resultados obtenidos respecto a los resultados que se esperan de los métodos y en caso de que existan errores se documentan y se informan para su posterior corrección.

3.9.1 Aplicación de las fases del procedimiento.

Para comenzar a implementar el procedimiento se debe contar con las precondiciones que el mismo exige para poder ser implementado, dichas precondiciones son:

- ✓ Descripción de la especificación de requisitos (se encuentra documentado en el Portafolio de Proyecto de SIGE).
- ✓ Descripción de los casos de usos.
- ✓ Modelo de Diseño.
    - Descripción de Clases del Diseño.
    - Descripción de los Casos de Uso del Diseño.

Una vez que ya se cuente con las precondiciones que exige el procedimiento, se hace necesario realizar un estudio de donde se va aplicar. El procedimiento para hacer pruebas unitarias descrito en el capítulo 2, se aplica al componente Gestionar Unidades de Observación, el mismo es implementado por desarrolladores del departamento “Integración de Soluciones”, perteneciente al Centro de Tecnología de Gestión de Datos.

Para el desarrollo de dicho componente se utilizó como metodología de desarrollo Open/Up, por lo que se hace necesario ver en que flujo de trabajo de la metodología se encuentran cada una de las fases descritas en el procedimiento de pruebas unitarias, dicha relación se representa en la siguiente ilustración (Ver ilustración No 3.6)

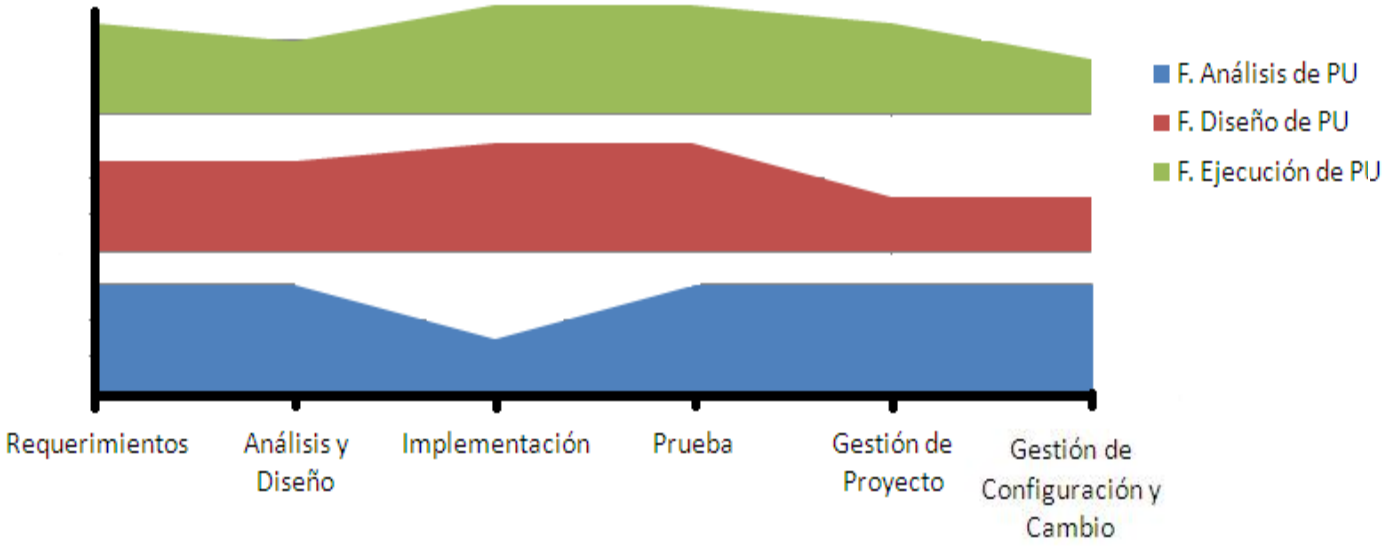

Ilustración 3.6 Correspondencia entre las Fases y los Flujos de Trabajo
Fuente: Investigación realizada.

Una vez estudiado como implementar el procedimiento se procede a la aplicación de cada una de sus fases:

Aplicación de la Fase de Análisis de Pruebas Unitarias

Uno de los componentes a los que se le aplicó el procedimiento fue Gestionar Unidades de Observación, el mismo cuenta con todos los artefactos necesarios que servirán de entrada para esta fase inicial, entiéndase por estos artefactos:

- ✓ Descripción de especificación de Requisitos.
- ✓ Descripción de los Casos de Usos.
- ✓ Modelo de diseño.
    - Descripción de Clases del Diseño.
    - Descripción de los Casos de Uso del Diseño.

Dichos artefactos fueron estudiados para posteriormente dar cumplimiento a cada una de las actividades descritas en esta fase inicial, para finalmente contar con el Plan Inicial de Pruebas Unitarias donde quedan recogidas las principales actividades desarrolladas, que servirán de entrada para la posterior fase.

Estrategia a seguir

Como parte de la estrategia a seguir debe detallarse:

- Objetivo

Realizar las pruebas unitarias a cada una de las funcionalidades del componente javascript Gestionar Unidades de Observación, para verificar su correcto funcionamiento, y garantizar el concepto de calidad total definido en DATEC.

- Flujo de trabajo

El flujo de trabajo a seguir es estudiar la documentación para dar cumplimiento a cada una de las actividades definidas en cada fase. De forma general las actividades son: describir los casos de prueba Unitaria, ejecutarlos y documentar los resultados.

- Trabajadores

Trabajador	Responsabilidades
Desarrollador	▪ Analiza la documentación ▪ Diseña los casos de prueba unitaria ▪ Ejecuta los casos de prueba unitaria ▪ Evalúa los resultados ▪ Corrige errores ▪ Documenta los resultados y los sube para el repositorio el activo con su documentación de pruebas unitarias
Jefe de proyecto o designado por el Grupo de Calidad de la LPS	▪ Chequea la calidad de las pruebas unitarias realizadas

Tabla 3.4 Trabajadores y sus responsabilidades

Fuente: Investigación realizada

- Herramienta

El sistema seleccionado es DATEST, por los beneficios que aporta, descritos anteriormente.

Aplicación de la Fase de Diseño de Pruebas Unitarias

En la Fase de Diseño de Pruebas Unitarias se realiza un estudio de los artefactos de entrada, que son:

- ✓ Plan Inicial de Pruebas Unitarias.
- ✓ Descripción de especificación de Requisitos.
- ✓ Código Fuente.
- ✓ Descripción de los Casos de Usos.
- ✓ Modelo de diseño.
    - Descripción de Clases del Diseño.
    - Descripción de los Casos de Uso del Diseño.

Una vez analizado dichos artefactos se diseñan los casos de prueba unitaria a cada una de las funcionalidades correspondientes del componente Gestionar Unidades de Observación y posteriormente se implementan.

- Descripción de los Casos de Prueba Unitaria.

Para esta actividad se procede a llenar los datos que solicita la Planilla de Casos de Pruebas Unitarias. Inicialmente se documenta la tabla de las funcionalidades a probar con sus características:

Funcionalidad	Descripción de la funcionalidad
CrearUnidadInformante()	La funcionalidad debe permitir crear una unidad informante mediante los parámetros requeridos.
BuscarUnidadInformante()	La funcionalidad debe devolver, dado un código, los datos del informante con el mismo.
addUnidadInformante()	La funcionalidad debe almacenar los datos de un informante una vez creado.

ObtenerListado()	La funcionalidad de tipo auxiliar debe devolver el listado de informantes previamente almacenados.

Tabla 3.5 Funcionalidades a probar

Fuente: Investigación realizada

Una vez analizadas las funcionalidades a probar se describen los casos de prueba unitaria:

Descripción de Caso de Prueba Unitaria					
Iteración:1.0					
Descripción de la Prueba Unitarias: Asegurar la correcta validación de datos de entrada					
Responsable: Hector Beltrán Roye					
Funcionalidad	**Método utilizado**	**Recibe**	**Respuesta esperada del método**	**Respuesta real de la prueba**	**Observaciones**
CrearUnidadInformante()	AssertNull ()	CrearUnidadInformante() ("habana",3,12,"hector","smp",null, "hybeltran",1255411244)	null	object	Corregir
CrearUnidadInformante()	AssertObjectEquals()	SeleccionarCentroInformante ("a","3",12,"hector","smp","25/5/11", "hyb","25/5/11")	number	string	Corregir

CrearUnidadInformante()	AssertObjectEquals()	("a","3",12,"hector","smp","25/5/11", "hyb","25/5/11")	string	number	Corregir
CrearUnidadInformante()	AssertObjectEquals()	CrearUnidadInformante() ("habana",3,12, "hector","smp",2011,"hybeltran",1255411244)	object	object	
CrearUnidadInformante()	AssertNull()	CrearUnidadInformante() ("habana", 3,12,"hector","smp",2011,"hybeltran",JSUNIT_UNDEFINED_VALUE)	null	NaN	Corregir
BuscarUnidadInformante()	AssertNull()	BucarUnidadInformante(Null)	null	object	Corregir
BuscarUnidadInformante()	AssertNull()	BucarUnidadInformante(JSUNIT_UNDEFINED_VALUE)	null	object	Corregir
BuscarUnidadInformante()	AssertObjectEquals()	BucarUnidadInformante(12)	object	object	
Observaciones: Si no coinciden los últimos dos campos, existen errores hay que corregir.					

Tabla 3.6 Diseño de los Casos de Prueba Unitaria
Fuente: Investigación realizada

- Implementación de los casos de prueba unitaria

El desarrollador implementa de forma automática siguiendo algunos criterios de cobertura de sentencias los casos de prueba unitaria ya descritos en el Plan de Casos de Pruebas Unitarias para su posterior ejecución. A continuación se muestra el caso de prueba unitaria implementado para cada una de las funcionalidades del componente Gestionar Unidades de Observación. (Ver Ilustración No.3.7)

```
function testArgumentosNulos() {assertNull("Un argumento de tipo null, debe resultar en null",
    CrearUnidadInformante("habana", 3,12,"hector","smp",null,"hybeltran",1255411244));

}

function testValoresDiferentes(){
    var objt={provincia: "a", region: 3, codigo:12,nombre:"hector",direccion:"smp",fecha:"25/5/11",correo:"hyb",
        telefono:8372894};

    assertObjectEquals("Strings encontrados en lugar de numeros",objt ,
    CrearUnidadInformante("a", "3",12,"hector","smp","25/5/11","hyb",8372894));

    assertObjectEquals("Numeros encontrados en lugar de Strings",objt,CrearUnidadInformante("a", 3,12,"hector",
    "smp",25511,"hyb",8372894));
}

function CasoValido(){
    var obj={provincia:"habana",region:3,codigo:12,nombre:"hector",direccion:"smp",fecha:2011,
        correo:"hybeltran",telefono:1255411244};
    assertObjectEquals("En caso de valores validos",obj,CrearUnidadInformante("habana", 3,12,
    "hector","smp",2011,"hybeltran",1255411244));
}
function testValoresIndefinidos() {
    assertNull("Un valor indefinido debe resultar en  null", CrearUnidadInformante("habana",
    3,12,"hector","smp",2011,"hybeltran",JSUNIT_UNDEFINED_VALUE));
}
```

```
function testArgumentosNulos2() {assertNull("Un argumento de tipo null, debe resultar en null",
    BuscarUnidadInformante(null));


}
function testValoresIndefinidos2() {
    assertNull("Un valor indefinido debe resultar en  null",  BuscarUnidadInformante(JSUNIT_UNDEFINED_VALUE));
}

function CasoValido2(){
    var obj={provincia:"habana",region:3,codigo:12,nombre:"hector",direccion:"smp",fecha:2011,
        correo:"hybeltran",telefono:1255411244};
    assertObjectEquals("En caso de valores validos",obj,BuscarUnidadInformante(12));
}
function CasoNoencontrado2(){
    assertObjectEquals("En caso de no encontrarse",-1,BuscarUnidadInformante(13));
}
```

Ilustración 3.7 Caso de Prueba Implementado
Fuente: Investigación realizada

Aplicación de la Fase de Ejecución de Pruebas Unitarias

En la Fase de Ejecución de Pruebas Unitarias se analizan todas las entradas, que son:

- ✓ Plan de los Casos de Pruebas Unitarias.
- ✓ Código Fuente.
- ✓ Descripción de especificación de Requisitos.
- ✓ Script de prueba.

Posteriormente se ejecutan los casos de prueba con el sistema y se evalúan los resultados.

- Evaluación de resultados

Se ejecutaron las pruebas unitarias automatizadas haciendo uso del sistema **DATEST**, probándose todas las funcionalidades del componente, se clasificaron los errores de acuerdo a su impacto y posteriormente fueron corregidos. Todos los resultados son registrados en el Plan de Casos de Pruebas Unitarias que se subirá con el activo al repositorio. En la siguiente tabla se muestra el resultado de la ejecución de las pruebas unitarias del componente Gestionar Unidades de Observación:

Reporte de Errores			
Errores encontrados: 3			
Funcionalidad: CrearUnidadInformante ()			
Responsable: Hector Beltrán Roye			
Método	**Error Encontrado**	**Clasificación**	**Observaciones**
AssertObjectEquals	Faltan las validaciones en cuanto a la especificidad de los datos de entrada.	crítico	Corregir inmediatamente
AssertNull()	Faltan las validaciones en cuanto a datos de entrada de valor indefinido.	alto	Corregir

AssertNull()	Faltan validaciones en caso de que existan campos vacíos	crítico	Corregir inmediatamente

Tabla 3.7 Evaluación de los resultados

Fuente: Investigación realizada

Se corrigen los errores encontrados y se vuelve a ejecutar la prueba, hasta no encontrar errores. En la siguiente ejecución de la prueba no se encontraron errores por lo tanto todas las funcionalidades probadas funcionan correctamente.

Como se pudo apreciar los errores encontrados fueron:

- ✓ Algunas funcionalidades no validaban los datos de entrada, lo que produce la entrada de datos incorrectos.
- ✓ Faltan validaciones que especifiquen que todos los campos todos deben llenarse, o sea que no se dejen campos vacíos.

Finalmente se probaron 6 funcionalidades correspondientes a 2 componentes del proyecto SIGE, uno de ellos el que se presenta en la investigación, para que sirva de apoyo en la aplicación del procedimiento. Se detectaron 3 funcionalidades con problemas, lo que representa un 50% del total de funcionalidades probadas.

3.10 Análisis de resultados

Una vez aplicado el procedimiento se evalúa de forma satisfactoria, ya que se realizaron las pruebas unitarias y se llenaron cada una de las plantillas requeridas, posibilitando que cada caso de prueba creado se realizara con el mayor rigor y la eficiencia necesaria. Una vez ejecutadas las pruebas arrojaron varios errores, que fueron corregidos; las funciones lanzaron las excepciones correspondientes en cada caso, mostrando su buen funcionamiento. Los resultados arrojados permiten afirmar de forma general:

- ✓ Que el procedimiento para realizar pruebas unitarias es útil, correcto, completo y efectivo para desarrollar dichas pruebas.

- ✓ Se considera que el procedimiento tiene un orden lógico y las actividades están estructuradas correctamente, y son fácil de llevar a cabo.
- ✓ Las actividades descritas en el procedimiento son suficientes para su buen desarrollo siempre y cuando haya disposición de llevarlas a cabo.
- ✓ Se evidencia la necesidad de implantar el mismo en el Centro de Tecnología y Gestión de Datos.
- ✓ Se considera que con la implantación de este procedimiento se mejorará la calidad de los productos y servicios brindados por el Centro de Tecnología de Gestión de Datos.

Conclusiones del capítulo

En el presente capítulo se construyó un sistema, donde se le cambió la interfaz gráfica del JsUnit, y se le agregaron nuevas funcionalidades. Se recogieron los artefactos más importantes en el análisis, diseño e implementación de la misma, como la especificación de requisitos, el diagrama de casos de usos, el diagrama de componentes, el diagrama de clases del diseño, obteniendo como resultado el sistema DATEST.

Se aplicó el procedimiento a varios componentes del proyecto SIGE, cumpliendo cada una de las actividades descritas en el procedimiento propuesto y se plasmaron los resultados en cada una de las planillas involucradas. La puesta en práctica del procedimiento, implicó que se examinara la estructura interna del software, lo que disminuye el número de errores existentes en las funcionalidades probadas, garantizando mayor calidad y confiabilidad. La utilización del sistema para la automatización agilizó todo el proceso de pruebas unitarias.

CONCLUSIONES

Luego de la investigación realizada sobre las pruebas unitarias y con el fin de fortalecer el concepto de calidad total definido en DATEC, así como el cumplimiento del objetivo general y de las tareas trazadas, se concluye que:

- ✓ Se desarrolló un estudio previo sobre los temas relacionados con las pruebas unitarias a componentes javascript en el contexto de una LPS.
- ✓ Se propuso un procedimiento que cuenta con un grupo de actividades que permiten realizar las pruebas unitarias de una forma planificada.
- ✓ Se propuso como sistema para la automatización de las pruebas DATEST, que fue implementado utilizando herramientas, lenguajes y tecnologías propuestas por el grupo de arquitectos del departamento “Integración de Soluciones”, en su mayoría distribuidas bajo licencias de software libre en correspondencia con las políticas de la universidad y del país.
- ✓ El uso del procedimiento como guía y del sistema para su automatización, ayudarán en el proceso de revisión de los artefactos creados por los desarrolladores.

De esta forma se cumplen los objetivos trazados, que a mediano plazo garantiza la calidad dentro del departamento “Integración de Soluciones”.

RECOMENDACIONES

Una vez cumplido con el objetivo general propuesto, se recomienda:

- ✓ Aplicar el procedimiento propuesto en otros proyectos de la LPS debido al resultado satisfactorio obtenido durante su aplicación.

REFERENCIAS BIBLIOGRÁFICAS

1. **R. JOHNSON, J. HOELLER.** *Expert One-on-One™ J2EE™ Development without EJB™.* 2004.

2. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico. 3ra Edición.* 1992.

3. **Yohandris Pupo Blez, LesdeySaavedra López.** *Diseño y aplicación de pruebas al producto Registro Cubano de Discapacitados.Trabajo de Diploma para optar por el título en Ciencias Informáticas.* 2007.

4. **Jacobson, I.** *Proceso unificado del desarrollo del software.* 2003.

5. **J. López, Oracle.** *Fundamentos para el desarrollo de aplicaciones Web.* Buenos Aires-Argentina : s.n.

6. **Hetzel, Bill.** *The Complete Guide To Software Testing.* 1998.

7. **Duharte, Franklin Rivero.** *Pruebas Unitarias de Software en la Plataforma J2EE.Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.* 2008.

8. **Julio Mellado Torío, Ing. Telecomunicación.** *Estrategia de pruebas de Líneas de Productos de Sistemas de Tiempo Real especificados con diagramas de estado jerárquico.Tesis Doctorial.UNIVERSIDAD POLITÉCNICA DE MADRID.* 2004.

9. **R.JOHNSON.** *Expert One-on-One J2EE Design and Development. Indianapolis.* 2003.

10. **Juristo Natalia Moreno, Ana M. Vegas Siria.** *Técnicas de evaluación del software. [En línea] 22 de noviembre de 2010. http://eva.uci.cu.* 2010.

11. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico. 5ta Edición.* 2002.

12. **Paul C. Clements, Lawrence G. Jones, Linda M. Northrop, John D. McGregor.** *Project management in a software product line organization IEEE Software. [En línea] 22 de noviembre del 2010. http://dx.doi.org/10.1109/MS.2005.133..* 2005.

13. **McGregor, John D.** *"Building Reusable Test Assets for a Product Line"tutorial. First Software Product Line Conference. Pittsburgh. Software Engineering Institute.Carnegie Mellon University .* 2000.

14. **Desarrollo, Sitio de Metodología de.** *Ayuda extendida de OpenUp. [En línea] 20 de noviembre de 2010. http://10.34.20.5:5800/OpenUP.* 2010.

15. **Caballero, Ismael.** *Una herramienta CASE: Visual Paradigm. [En línea] 24 de noviembre de 2010. http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/LabTr1_VP.pdf.* 2010.

16. **Xavier Ferré Grau, María Isabe Sánchez Segura.** *Tutorial UML, Desarrollo orientado a objetos con UML. [En línea] 25 de noviembre del 2010. http://www.clikear.com.* 2004.

17. **JavaScript.com.** *The Definitive JavaScript Resource: JavaScript Tutorials, Free JavaScripts, Source Code and Other Scripting Resources. [En línea] 30 de noviembre del 2010.* 2010.

18. **Extjs.com.** *Introducing Ext JS 2.2. Ext Management Team. Ext JS- A foundation you can build on. [En línea] 30 de noviembre del 2010. http://www.extjs.com.*

19. **PHP.net.** *The PHP Group. News Archive - 2007. [En línea] 20 de noviembre del 2010. http://www.php.net/archive/2007.php. .*

20. *El Mundo Informático. [En línea] 26 de 11 de 2010. http://jorgesaavedra.wordpress.com.* 2010.

21. **Meyer, Bertrand.** *Costrucción del software orientado a objetos. Madrid : Prentice Hall.* 1999.

22. **LEWIS, W. E.** *Software Testing and Continuous Quality Improvement . AUERBACH PUBLICATIONS.* 2005.

23. **Larman, Craig.** *UML y Patrones .* 2002.

24. **Montilva, Jonás A.** *Desarrollo de Software Basado en Componentes. [En línea] 25 de febrero de 2011.http://ulaweb.adm.ula.ve/DSIA/presentaciones/2a%20Charla%20DSIA%20Componentes.ppt>.* 2011.

Bibliografía

Caballero, Ismael. *Una herramienta CASE: Visual Paradigm. [En línea] 24 de noviembre de 2010. http://alarcos.inf-cr.uclm.es/per/fgarcia/isoftware/doc/LabTr1_VP.pdf.* 2010.

Desarrollo, Sitio de Metodología de. *Ayuda extendida de OpenUp. [En línea] 20 de noviembre de 2010. http://10.34.20.5:5800/OpenUP.* 2010.

Duharte, Franklin Rivero. *Pruebas Unitarias de Software en la Plataforma J2EE.Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.* 2008.

El Mundo Informático. [En línea] 26 de 11 de 2010. http://jorgesaavedra.wordpress.com. 2010.

Extjs.com. *Introducing Ext JS 2.2. Ext Management Team. Ext JS- A foundation you can build on. [En línea] 30 de noviembre del 2010. http://www.extjs.com.*

Hetzel, Bill. *The Complete Guide To Software Testing.* 1998.

J. López, Oracle. *Fundamentos para el desarrollo de aplicaciones Web.* Buenos Aires-Argentina : s.n.

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

JavaScript.com. *The Definitive JavaScript Resource: JavaScript Tutorials, Free JavaScripts, Source Code and Other Scripting Resources. [En línea] 30 de noviembre del 2010.* 2010.

Julio Mellado Torío, Ing. Telecomunicación. *Estrategia de pruebas de Líneas de Productos de Sistemas de Tiempo Real especificados con diagramas de estado jerárquico.Tesis Doctorial.UNIVERSIDAD POLITÉCNICA DE MADRID.* 2004.

Juristo Natalia Moreno, Ana M. Vegas Siria. *Técnicas de evaluación del software. [En línea] 22 de noviembre de 2010. http://eva.uci.cu.* 2010.

Larman, Craig. *UML y Patrones .* 2002.

LEWIS, W. E. *Software Testing and Continuous Quality Improvement . AUERBACH PUBLICATIONS.* 2005.

McGregor, John D. *"Building Reusable Test Assets for a Product Line"tutorial. First Software Product Line Conference. Pittsburgh. Software Engineering Institute.Carnegie Mellon University .* 2000.

Meyer, Bertrand. *Costrucción del software orientado a objetos. Madrid : Prentice Hall.* 1999.

Montilva, Jonás A. *Desarrollo de Software Basado en Componentes. [En línea] 25 de febrero de 2011. http://ulaweb.adm.ula.ve/DSIA/presentaciones/2a%20Charla%20DSIA%20Componentes.ppt>.* 2011.

P Clements, L Northrop. 2001. *Software Product Lines: Practices and Patterns.* Addison Wesley, Massachussets : s.n., 2001.

Paul C. Clements, Lawrence G. Jones, Linda M. Northrop, John D. McGregor. *Project management in a software product line organization IEEE Software. [En línea] 22 de noviembre del 2010. http://dx.doi.org/10.1109/MS.2005.133..* 2005.

PHP.net. *The PHP Group. News Archive - 2007. [En línea] 20 de noviembre del 2010. http://www.php.net/archive/2007.php. .*

Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico. 3ra Edición.* 1992.

Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico. 5ta Edición.* 2002.

R.JOHNSON. *Expert One-on-One J2EE Design and Development. Indianapolis.* 2003.

R. JOHNSON, J. HOELLER. *Expert One-on-One™ J2EE™ Development without EJB™.* 2004.

Xavier Ferré Grau, María Isabe Sánchez Segura. *Tutorial UML, Desarrollo orientado a objetos con UML. [En línea] 25 de noviembre del 2010. http://www.clikear.com.* 2004

Yohandris Pupo Blez, LesdeySaavedra López. *Diseño y aplicación de pruebas al producto Registro Cubano de Discapacitados.Trabajo de Diploma para optar por el título en Ciencias Informáticas.* 2007.

ANEXOS

Anexo 1: Planilla del Plan Inicial de Pruebas Unitarias

<Nombre del Proyecto>
Módulo: <Nombre del Componente>
<Versión del proyecto>

Plan Inicial de Pruebas Unitarias
Funcionalidades basadas en Componentes

Control de versiones:

Fecha	Versión	Descripción	Autor
<dd/mmm/yy>	<x.x>	<detalles>	<nombre>

Reglas de seguridad

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimiento público su contenido, excepto para cumplir el propósito para el cual se ha generado.
Estas reglas son aplicables a las **N** páginas de este documento.

1. **Descripción General.**

<Resumen del Componente>

2. **Condiciones de Ejecución.**

<Precondiciones del Componente>

3. **Trabajadores y sus responsabilidades**

<Cada trabajador con su responsabilidad >

Trabajadores	Responsabilidades
< Nombre del trabajador>	*< responsabilidad específica.>*
[2]	

4. Definición de la Estrategia de Prueba Unitaria.

[De la estrategia de Prueba Unitaria debe detallarse]

4.1 Objetivo.

[Objetivo a alcanzar]

4.2 Flujo de Trabajo.

[Especifica el hilo de actividades seguir para desarrollar las pruebas unitarias]

4.3 Descripción de la herramienta.

[Específica y describe que herramientas utilizar para desarrollar las pruebas unitarias]

Anexo 2: Planilla del Plan de Casos de Pruebas Unitarias

<Nombre del Proyecto>
Módulo: <Nombre del Componente>
<Versión del proyecto>

Plan de Casos de Pruebas Unitarias
Funcionalidades basadas en Componentes

Control de versiones:

Fecha	Versión	Descripción	Autor
<dd/mmm/yy>	<x.x>	<detalles>	<nombre>

Reglas de seguridad

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimiento público su contenido, excepto para cumplir el propósito para el cual se ha generado. Estas reglas son aplicables a las **N** páginas de este documento.

1. **Descripción General.**

<Resumen del Componente>

2. **Condiciones de Ejecución.**

<Precondiciones del Componente>

3. **Funcionalidades a probar del Componente.**

<Para cada funcionalidad del Componente >

Funcionalidad	Descripción de la funcionalidad
< Nombre de la funcionalidad>	*<Descripción de la Funcionalidad.>*

4. Descripción del Caso de Prueba Unitarias.

Descripción de Caso de Prueba Unitario					
Iteración:< Iteración que se ejecuta >					
Descripción de la Prueba Unitarias:< Descripción de la prueba >					
Responsable:< Nombre del responsable >					
Funcionalidad	**Método utilizado**	**Recibe**	**Respuesta esperado del método**	**Respuesta real de la prueba**	**Observaciones**
<Nombre de la funcionalidad >	*< se escribe el nombre del método a utilizar>*	*< parámetros que recibe la funcionalidad >*	*<se escribe la respuesta esperado del método >*	*< se escribe la respuesta real de la prueba >*	*< claras y precisas>*
Observaciones: Si no coinciden los últimos dos campos, existen errores hay que corregir.					

5. Reporte de Errores

Reporte de Errores			
Errores encontrados: *<Cantidad de errores encontrados>*			
Funcionalidad: *<Nombre de la funcionalidad probada>*			
Responsable: *<Nombre del responsable>*			
Método	**Error Encontrado**	**Clasificación**	**Observaciones**
<Nombre de la sección>	*<descripción de la falla encontrada>*	*< prioridad de impacto >*	*<clara y precisa >*
[2]			

Anexo 3: Aval del Trabajo de Diploma

AVAL a Trabajos de Diplomas

DATEC
CENTRO DE TECNOLOGÍAS DE GESTIÓN DE DATOS

Web: https://portal.datec.prod.uci.cu/ **email:** datec@uci.cu **telef.:** 837 3712, 837 3709

La Habana, 15 de junio de 2011
"Año 53 de la Revolución"

De: Ing. Armando Robert Lobo
Líder del proyecto Sistema de Información de Gobierno (SiGOB),
Arquitecto de Sistemas del Departamento de Integración de Soluciones (DATEC)

A: Miembros del Tribunal.

Por medio del presente documento certifico que la investigación realizada por los estudiantes Hector Yasmany Beltran Royé y Ana Lilian López Cordero, tutorada por Ing. Frank González Fernández e Ing. Diana Monne Roque tributa al proyecto donde se desarrolla. A continuación se exponen los elementos que amparan la afirmación anterior:

1. La solución software para la realización de pruebas unitarias a Java Script beneficia el proceso de desarrollo de software directamente al promover la calidad de los resultados de conjunto con el procedimiento propuesto durante la construcción de componentes en el grupo de desarrollo.
2. Se implementaron los requisitos previstos en el alcance del trabajo de diploma, incluidos los artefactos de ingeniería requeridos por DATEC.
3. Se realizaron con el rigor requerido las pruebas funcionales que verifican la corrección de la solución, requisito para que la misma forme parte de los activos del Departamento de Integración de Soluciones de DATEC.

Finalmente, se solicita adjuntar este documento al expediente de Tesis de los estudiantes. Para que así conste, firmo la presente, en un ejemplar, en la Universidad de las Ciencias Informáticas a los 15 días del mes de junio del año 2011.

Ing. Armando Robert Lobo
Líder del proyecto

Facultad 6 - Universidad de las Ciencias Informáticas (UCI)
Km. 2½ Carretera a San Antonio de los Baños. Reparto Torrens. Boyeros, Ciudad de la Habana. Cuba

UCI Universidad de las Ciencias Informáticas
Decanato Facultad 6

GLOSARIO DE TÉRMINOS

Procedimiento: consiste de una serie de pasos bien definidos que permitirán y facilitarán la realización de un trabajo de la manera más correcta y exitosa posible.

Componente: Es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio.

Xunit: Un conjunto estandarizado de conceptos independientes del lenguaje para escribir y ejecutar pruebas unitarias.

Artefacto: es una información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos: un modelo, una descripción o un software.