Universidad de las Ciencias Informáticas

Facultad 4

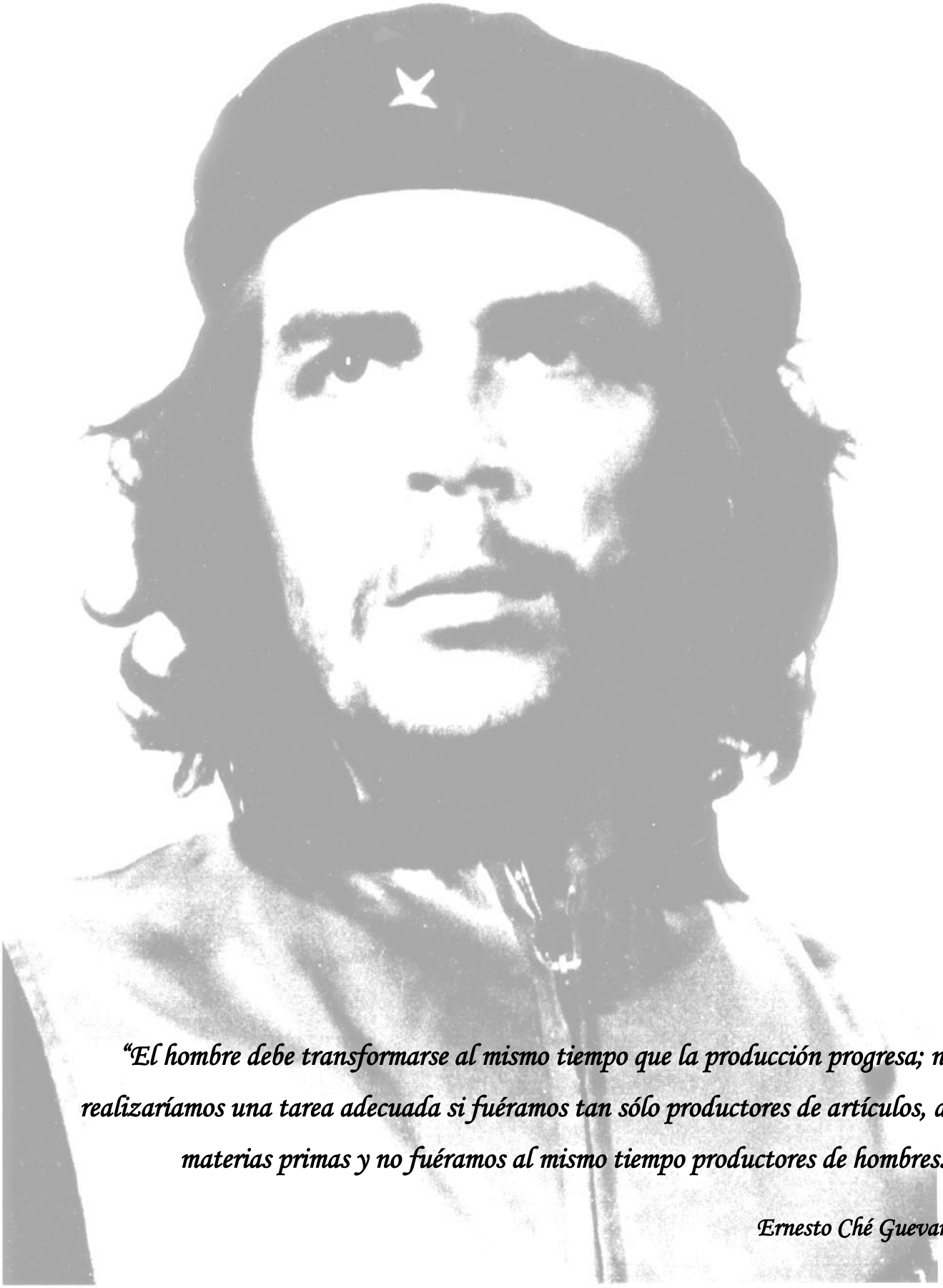
Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: *"Procedimiento para la realización de pruebas de unidad dentro del proyecto Sistema Único de Aduanas".*

Autor: Elizabeth Quintas Sánchez

Tutor: Ing. Maurice Cabrejas Martínez

Junio, 2010.



"El hombre debe transformarse al mismo tiempo que la producción progresa; no realizaríamos una tarea adecuada si fuéramos tan sólo productores de artículos, de materias primas y no fuéramos al mismo tiempo productores de hombres."

Ernesto Ché Guevara

DECLARACIÓN DE AUTORÍA

DECLARACIÓN DE AUTORÍA.

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2010.

Elizabeth Quintas Sánchez **Ing. Maurice Cabrejas Martínez**

_____ _____

Firma del Autor Firma del Tutor

AGRADECIMIENTOS

Agradezco a la Revolución y en especial a Fidel por darme la oportunidad de formarme como ingeniera.

A mis padres, guardianes de mi bienestar y desarrollo profesional. Los que me han dado el mayor impulso para salir adelante. Los adoro.

A mi hermano, gracias por su ayuda y apoyo.

A toda mi familia, por preocuparse por mí.

A mi novio por cuidarme y brindarme todo su amor y cariño. Y sobre todo por estar ahí siempre. Te quiero mucho.

A todos los profesores que en estos cinco años me brindaron los conocimientos y consejos necesarios para crecer profesionalmente.

A todos mis compañeros y amigos por su paciencia, ayuda y dedicación en los momentos más difíciles.

A mi tutor por guiarme en este trabajo.

A todos los que contribuyeron amable y desinteresadamente con la realización de este trabajo, con su tiempo, conocimiento y experiencia.

DEDICATORIA

A mis padres *pues todo lo que soy y seré en la vida se lo agradezco a ellos.*

Porque en cada tropiezo, ante cada momento difícil de mi vida siempre tuve de ellos un gesto de amor y dedicación.

Siempre me apoyaron para que estudiara y me superara y me enseñaron que con esfuerzo todo se puede alcanzar.

Gracias por haber sido mi motor impulsor ante todos mis años de estudios.

A mi novio *por lo especial y maravilloso que ha sabido ser conmigo, por brindarme tanto amor y comprensión.*

Gracias por ser paciente conmigo y ayudarme tanto en todo.

RESUMEN

Resumen

El factor fundamental para el éxito en la producción de software es la calidad y para ello es necesario tener en cuenta una serie de aspectos para que la misma sea óptima. La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software, que permitan uniformar la filosofía de trabajo en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad tanto para la labor de desarrollo como para el control de la calidad del software.

Antes de que el software se le entregue al usuario final es necesario realizar pruebas con el objetivo de detectar errores de la aplicación y la documentación; este proceso resulta de gran importancia ya que da una medida de la calidad del producto siempre que se lleve a cabo de forma apropiada.

En la Universidad de Ciencias Informáticas (UCI), específicamente en la facultad 4, se ha venido desarrollando el proyecto Sistema Único de Aduanas (SUA), el presente trabajo se centra en la aplicación de un procedimiento para realizar pruebas de unidad en dicho proyecto con el objetivo de lograr el nivel de calidad requerido y poder registrar la documentación con los resultados de cada una de las pruebas de unidad realizadas al producto. Para ello se utiliza la metodología RUP, Proceso Unificado de Desarrollo de Software, para lograr una mejor organización en el trabajo, y el framework Symfony para realizar las pruebas unitarias automáticamente.

Palabras Claves

Pruebas de unidad, procedimiento de pruebas, calidad de software.

TABLA DE CONTENIDO

TABLA DE CONTENIDO

TABLA DE CONTENIDO

INTRODUCCIÓN

Introducción

Hoy en día la humanidad está inmersa en un proceso de constante evolución tecnológica, trayendo como consecuencia que muchos países elaboren productos de software cada vez más competitivos para su inserción en el mercado. Nuestro país no está exento a estos cambios, por esta razón ha definido como estrategia la creación de la Universidad de las Ciencias Informáticas (UCI), centro en el cual se forman futuros ingenieros especializados en la industria del software y en la producción del mismo. Para lograr dicho objetivo es necesario aplicar adecuadamente los procedimientos, herramientas y estándares que permitan desarrollar, con la calidad requerida, el proceso de software y de esta manera obtener un producto de gran aceptación para el cliente.

En la UCI se están desarrollando una serie de proyectos de gran envergadura, entre ellos el Sistema Único de Aduanas (SUA), que tiene como objetivo el desarrollo de aplicaciones para contribuir con la informatización de la Aduana General de la República de Cuba.

Una de las fases más importantes del ciclo de vida antes de entregar un software para su explotación es la fase de pruebas. Uno de los problemas con los que suelen encontrarse los nuevos programadores es la enorme cantidad de tiempo y esfuerzo que requiere esta fase. Se estima que la mitad del esfuerzo de desarrollo de un programa (tanto en tiempo como en gastos) se consume en esta fase [1], de ahí la importancia de ésta.

La calidad del software es medible y varía de un programa a otro, puede medirse luego de haberse elaborado el producto, pero esto puede resultar muy costoso, por lo que es imprescindible realizar pruebas de calidad durante todas las etapas del ciclo de vida del software.

Uno de los pasos para garantizar que el producto final tenga la menor cantidad de errores posibles es el desarrollo de un correcto procedimiento de pruebas de unidad. En el proyecto SUA no existe una planificación concreta y eficiente para realizar este tipo de pruebas, debido a la importancia que representa es conveniente la creación de un procedimiento dirigido a realizar pruebas unitarias para lograr una mayor calidad del producto final.

Basándose en lo anterior surge el siguiente **problema a resolver**: ¿Qué procedimiento seguir para realizar las pruebas de unidad dentro del proyecto Sistema Único de Aduanas?

Para dar respuesta al problema planteado se definió como **objetivo general** elaborar un procedimiento que permita planificar, realizar, medir, controlar y documentar la ejecución de pruebas de unidad al proyecto Sistema Único de Aduanas.

El **objeto de estudio** está enmarcado en el análisis del proceso de aplicación de pruebas de unidad.

El **campo de acción** está definido por el proceso de pruebas de unidad dentro del Sistema Único de Aduanas.

Los **posibles resultados** serían los flujos de trabajo para la planificación, ejecución, medición, control y documentación de pruebas de unidad.

Para dar cumplimiento al objetivo de este trabajo se proponen las siguientes **tareas de investigación**:

- Revisión de la documentación existente acerca de las pruebas de unidad.
- Revisión de la documentación referente a los métodos de planificación, medición y control de pruebas.
- Revisión de la información existente sobre los métodos de documentación de pruebas.
- Definición de un procedimiento para las pruebas de unidad a aplicar.
- Definición de los métodos de planificación, medición y control de las pruebas a aplicar.
- Definición de un método de documentación de pruebas.
- Definición de una herramienta para la automatización de las pruebas de unidad.

Métodos científicos de investigación empleados:

Métodos teóricos:

Analítico-sintético: Este método permite comprender cómo se realizan las pruebas de unidad en el proyecto Sistema Único de Aduanas, así como realizar un análisis profundo de los procedimientos, sintetizar los conocimientos y de esta forma descubrir todas las características generales. Mediante el estudio de las pruebas de unidad se documentan y definen en el presente trabajo los principales métodos de planificación, medición y control de las pruebas a aplicar.

Modelación (Modelo teórico): Permite representar las relaciones entre los roles y las principales actividades a realizar.

El presente trabajo se encuentra estructurado por tres capítulos y anexos, en los cuales se exponen todo el trabajo investigativo y práctico realizado:

Capítulo 1. Fundamentación Teórica:

En este capítulo se aborda fundamentalmente el estado actual de las actividades relacionadas con las pruebas. Se realiza además un estudio sobre la metodología RUP y conceptos fundamentales del proceso de pruebas de software, así como sus objetivos, principios y estrategia. La calidad del software, los niveles de prueba, los tipos de pruebas por niveles y las técnicas también son conceptos que se manejan en este capítulo.

Capítulo 2. Desarrollo del procedimiento:

Se explica el procedimiento para la realización de pruebas de unidad, definiéndose 3 fases, así como las actividades a desarrollar en cada una de ellas. Además se plantea una propuesta de la plantilla donde se documentarán los casos de prueba de unidad.

Capítulo 3: Aplicación del procedimiento:

El procedimiento desarrollado se aplica en el Subsistema Mecanismo de Control de Acceso y Autenticación perteneciente al proyecto Sistema Único de Aduanas, con la realización de pruebas de unidad a 3 de sus módulos. Se realiza la evaluación de los resultados a partir de la ejecución de las pruebas de unidad y se exponen los errores encontrados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

Las pruebas son de suma importancia para el software y tienen sus implicaciones en la calidad de éste, citando a Deutsch:

“El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos pueden estar especificados de forma errónea o imperfecta, así como en posteriores pasos de diseño y desarrollo. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad” [2].

Las pruebas del software son un elemento crítico para la garantía de la calidad del mismo y representan una revisión final de las especificaciones del diseño y de la codificación. La creciente percepción del software como un elemento del sistema y la importancia de los costes asociados a un fallo del propio sistema, están motivando a la creación de pruebas minuciosas y bien planificadas.

Con el objetivo de llevar a cabo un procedimiento apropiado referente a las pruebas de unidad, en el presente capítulo se hace un análisis de los conceptos de prueba definidos por varios autores, se dan a conocer sus objetivos y principios que la rigen, así como varios conceptos de calidad de software y su importancia. Se detallan las técnicas, niveles y los tipos de prueba que se realizan para métodos convencionales.

1.1 Estado Actual

Actualmente la industria del software presenta un gran crecimiento y debe asumir las altas exigencias de los clientes y usuarios respecto a las prestaciones y calidad de sus productos. Muchas organizaciones desconocen los aspectos fundamentales del proceso de pruebas, este desconocimiento lleva con frecuencia a subestimar la complejidad de las actividades involucradas en este proceso, lo cual cobra un alto precio en los proyectos.

Existen empresas que se encargan de brindar servicios especializados de pruebas de software, algunas de ellas son [3]:

- La compañía inQA.labs es la líder mundial en soluciones de automatización de pruebas y gestión de los procesos de calidad de software, es una empresa de capital español fundada en Barcelona en 1999 especializada en servicios de prueba de software y consultoría de gestión de calidad de software. Sus servicios ayudan a las empresas a aumentar la rentabilidad de sus proyectos de software, ofreciéndoles la posibilidad de desarrollar productos de mayor calidad.

Apoyándose en una sofisticada infraestructura y un experimentado equipo de profesionales, la compañía reproduce fielmente cualquier entorno del mercado para realizar outsourcing de test, mejorando sustancialmente los procesos relacionados con la calidad de software.

-El Centro de Ensayos de Software (CES) desarrolla su actividad desde el año 2004, es un emprendimiento conjunto entre la Cámara Uruguaya de Tecnologías de la Información (CUTI), el Instituto de Computación (In.Co.) de la Facultad de Ingeniería de la Universidad de la República (UdelaR) de Uruguay, financiado en sus comienzos por la Unión Europea. Tiene por objetivo brindar servicios de pruebas independientes de productos, consultoría y capacitación de pruebas, con el fin de mejorar su capacidad productiva en cuanto a calidad, diversidad de plataformas e innovación de sus productos.

-Software Quality Systems (SQS) es la compañía líder en servicios de Consultoría de Calidad de Software y Pruebas en España. Ofrece servicios de validación y verificación independiente, es experta en diseño e implantación de entornos de pruebas, en revisiones formales de documentación y código así como otros servicios que permiten a sus clientes un mayor control sobre el proceso, una identificación temprana de errores y una reducción drástica de los costes para subsanar estos errores.

1.2 Conceptos de Calidad

1.2.1 Calidad de Software

Hoy en día las empresas buscan no sólo software que funcionen sino que tengan alta calidad. Aunque en muchas ocasiones no se tome muy en serio, esta disciplina tiene gran importancia tanto para el cliente como para los desarrolladores, ya que un software de baja calidad puede prolongar el proceso de desarrollo del producto y hacerlo más engorroso, lo que conlleva en muchos casos al fracaso del proyecto.

La calidad del software se define como:

Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente [2].

La calidad del software siguiendo la norma ISO 9126 describe seis características compuestas: Funcionalidad, Fiabilidad, Facilidad de uso, Eficiencia, Mantenimiento y Movilidad. Así, cuando se adquiere un software se desea que éste funcione siempre y bajo diferentes condiciones, incluso difíciles de cumplir (fiabilidad), que realice las funcionalidades que dice tener y que por ello se ha adquirido (funcionalidad), que se puedan ejecutar las funcionalidades de una forma fácil (facilidad de uso), que lo haga lo más rápido posible y con el mínimo consumo de recursos (eficiencia), que

cuando las circunstancias lo pidan pueda modificarse fácilmente (mantenimiento) y, por último, que pueda transferirse de un entorno a otro (movilidad o portabilidad).

1.2.2 Gestión de la calidad de Software

Se conoce como Gestión de Calidad a los "Aspectos de la función de gestión que determinan y aplican la política de la calidad, los objetivos y las responsabilidades y que lo realiza con medios tales como la planificación de la calidad, el control de la calidad, la garantía de calidad y la mejora de la calidad" [4].

La Gestión de la Calidad de Software es según ISO 9000 el conjunto de actividades de la función general de la dirección que determina la calidad, los objetivos y las responsabilidades y se implanta por medios tales como la planificación de la calidad, el control de la calidad, el aseguramiento (garantía) de la calidad y la mejora de la calidad, en el marco del sistema de calidad [5].

1.2.3 Aseguramiento de la calidad del software

El aseguramiento de la calidad es un modelo planificado y sistemático para proporcionar una adecuada confianza en que un producto es conforme con los requerimientos técnicos establecidos.

Es el "Conjunto de actividades sistemáticas necesarias para aportar la confianza en que el software satisfaga los requisitos dados de calidad". La IEEE[1] lo define como "conjunto de actividades para evaluar el proceso mediante el cual se desarrolla el software" [6].

Entre las principales técnicas para el aseguramiento de la calidad, según el estándar IEEE 1074, están las siguientes:

- Métricas del software para controlar el proyecto.

- Verificación y validación del software (incluyendo pruebas y revisiones) en todas las fases del ciclo de vida.

- Gestión de la configuración del software. [7]

1.2.4 Control de la calidad del software

El control de calidad es una serie de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso del software, para asegurar que cada producto cumpla con los requisitos que le han sido asignados.

"Técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en dos objetivos fundamentales: mantener bajo control un proceso y eliminar las causas de defectos en las diferentes fases del ciclo de vida" [6].

[1] IEEE: Siglas de Instituto de Ingenieros Eléctricos y Electrónicos.

La organización IEEE lo define como "el proceso de verificar el propio trabajo o el de un compañero" [6].

1.3 Pruebas del Software

Una de las actividades críticas en el aseguramiento de la calidad es la prueba, la cual para Myers [8], es el “proceso de ejecutar un programa con el fin de encontrar errores”. El nombre “prueba”, además de la actividad de probar, se puede utilizar para designar “un conjunto de casos y procedimientos de prueba” [9].

“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” [10]

Kaner, Falk y Nguyen sugieren los siguientes atributos de una buena prueba [2]:

- Una buena prueba no debe ser redundante. El tiempo y los recursos para las pruebas son limitados. No hay motivo para realizar una prueba que tiene el mismo propósito que otra. Todas las pruebas deberían tener un propósito diferente (incluso si es sutilmente diferente).

- Una buena prueba debería ser la mejor de la cosecha. En un grupo de pruebas que tienen propósito similar, las limitaciones de tiempo y recursos pueden abogar por la ejecución de sólo un subconjunto de estas pruebas. En tales casos, se debería emplear la prueba que tenga la más alta probabilidad de descubrir una clase entera de errores.

- Una buena prueba no debería ser ni demasiado sencilla ni demasiado compleja. Aunque es posible a veces combinar una serie de pruebas en un caso de prueba, los posibles efectos secundarios de este enfoque pueden enmascarar errores. En general, cada prueba debería realizarse separadamente.

El responsable de la prueba debe entender el software e intentar desarrollar una imagen mental de cómo podrá fallar el mismo.

Durante todo el proceso de desarrollo de un software las pruebas son una parte fundamental del mismo, pues a partir de éstas es posible controlar que los productos cumplan requisitos mínimos de operabilidad además de garantizar la calidad de los mismos.

1.4 ¿Qué es probar?

Como parte del proceso de desarrollo del software la fase de pruebas añade valor al producto que se maneja: todos los programas tienen errores y la fase de pruebas los descubre; ese es el valor que añade. El objetivo específico de la fase de pruebas es encontrar cuantos más errores, mucho mejor.

Es frecuente encontrarse con el error de afirmar que el objetivo de esta fase es convencerse de que el programa funcione bien. En realidad ese es el objetivo propio de las fases anteriores (¿quién va a pasar a la sección de pruebas un producto que sospecha que está mal?). Cumplido ese objetivo, lo mejor posible, se pasa a realizar las pruebas. Esto no consta para reconocer que el objetivo último de todo el proceso de fabricación de programas, sea hacer programas que funcionen bien; pero cada fase tiene su objetivo específico y el de las pruebas es destapar errores. Probar un programa es ejercitarlo con la peor intención a fin de encontrarle fallos. Para que el proceso de pruebas sea totalmente efectivo es necesario tener en cuenta cuáles son los objetivos de las mismas, los cuales se detallan a continuación.

1.5 Objetivos de las pruebas

Glen Myers establece varias normas que pueden servir acertadamente como objetivos de las pruebas [8]:

- La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.

- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.

- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

El objetivo es diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y de esfuerzo.

En fin, es muy importante el cumplimiento de estos objetivos ya que dicho proceso posibilita no sólo la detección de errores de la aplicación, verificando el cumplimiento de los requisitos funcionales y no funcionales, errores ortográficos, abreviaturas, la funcionalidad de vínculos, botones, la ayuda, sino que también verifica la correspondencia de la documentación con la aplicación, dígase especificaciones de casos de uso del sistema, manuales de usuario y de instalación. Además, se comprueba la lógica interna del programa, pues un buen resultado de la implementación no sólo depende de la codificación sino también de las pruebas.

La realización de un proceso adecuado facilita que no aumenten los costos y tiempo del proyecto, que disminuyan los riesgos asociados y además que se eleve el nivel de calidad del producto, de esta forma, una vez que el mismo llegue a manos del cliente va a tener la menor cantidad de errores, obteniendo como resultado la satisfacción del usuario final.

Una vez conocidos los objetivos de las pruebas se deben seguir una serie de principios para que el proceso tenga calidad, éstos se explican a continuación.

1.6 Principios de las pruebas

Un ingeniero del software debe entender los principios básicos que guían las pruebas del software. Davis sugiere un conjunto de principios de prueba [2]:

- *A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.* El objetivo de las pruebas del software es descubrir errores. Se entiende que los defectos más graves (desde el punto de vista del cliente) son aquellos que impiden al programa cumplir sus requisitos.
- *Las pruebas deberían planificarse mucho antes de que empiecen.* La planificación de las pruebas puede empezar tan pronto como esté completo el modelo de requisitos. La definición detallada de los casos de prueba puede empezar tan pronto como el modelo de diseño se haya consolidado. Por tanto, se pueden planificar y diseñar todas las pruebas antes de generar ningún código.
- *El principio de Pareto es aplicable a la prueba del software.* Dicho de manera sencilla, el principio de Pareto implica que al 80 por 100 de todos los errores descubiertos durante las pruebas se les puede hacer un seguimiento hasta un 20 por 100 de todos los módulos del programa. El problema, por supuesto, es aislar estos módulos sospechosos y probarlos concienzudamente.
- *Las pruebas deberían empezar por <<lo pequeño>> y progresar hacia <<lo grande>>.* Las primeras pruebas planeadas y ejecutadas se centran generalmente en módulos individuales del programa. A medida que avanzan las pruebas, desplazan su punto de mira en un intento de encontrar errores en grupos integrados de módulos y finalmente en el sistema entero.
- *No son posibles las pruebas exhaustivas.* El número de permutaciones de caminos para incluso un programa de tamaño moderado es excepcionalmente grande. Por este motivo, es imposible ejecutar todas las combinaciones de caminos durante las pruebas. Es posible, sin embargo, cubrir adecuadamente la lógica del programa y asegurarse de que se han aplicado todas las condiciones en el diseño a nivel de componente.
- *Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.* Por <<más eficaces>> se refiere a pruebas con la más alta probabilidad de encontrar errores (es el objetivo principal de las pruebas). El ingeniero del software que creó el sistema no es el más adecuado para llevar a cabo las pruebas para el software.

Se puede decir que los principios constituyen para las pruebas, requerimientos a seguir para certificar su éxito: detectar la mayor cantidad de errores.

1.7 Estrategia de las pruebas

Para aplicar pruebas al software se deben seguir un conjunto de estrategias para lograr que éstas se hagan en el menor tiempo posible y con la calidad requerida, además de lograr que arrojen los resultados esperados.

Una estrategia de prueba del software debe ser suficientemente flexible para promover la creatividad y adaptabilidad necesarias para adecuar la prueba a todos los grandes sistemas basados en software. Al mismo tiempo la estrategia debe ser lo suficientemente rígida para promover un seguimiento razonable de la planificación y la gestión a medida que progresa el proyecto.

Las pruebas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón, se debe definir en el proceso de la ingeniería del software una plantilla para las pruebas del software: un conjunto de pasos en los que podamos situar los métodos específicos de diseño de casos de prueba.

Se han propuesto varias estrategias de prueba del software con las siguientes características generales definidas por Pressman [2]:

- Las pruebas comienzan a nivel de módulo y trabajan (hacia fuera), hacia la integración de todo el sistema basado en computadora.

- Según el momento, son apropiadas diferentes técnicas de prueba.

- La prueba la lleva a cabo el responsable del desarrollo del software y (para grandes proyectos) un grupo independiente de pruebas.

- La prueba y la depuración son actividades diferentes, pero la depuración se debe incluir en cualquier estrategia de prueba.

Se puede concluir que en un proyecto la prueba, en ocasiones, requiere mayor esfuerzo que cualquier otra actividad de la Ingeniería de Software; si se efectúa sin un plan estratégico el tiempo se desaprovecha y el esfuerzo es consumido innecesariamente, además en el peor de los casos, los errores inadvertidos quedarán sin detectar, por tanto, puede ser muy conveniente establecer una estrategia sistemática para probar el software.

1.8 Técnicas de diseño de casos de prueba

Las técnicas 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.

Entre las técnicas más importantes para elaborar los casos de prueba de software se encuentran la prueba de Caja Blanca y la prueba de Caja Negra, las cuales se verán en detalle a continuación.

1.8.1 Técnica de Caja Blanca

Esta técnica se basa en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.

Permite examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que [11]:

- Se ejercitan todos los caminos independientes de cada módulo.
- Se ejercitan todas las decisiones lógicas.
- Se ejecutan todos los bucles.
- Se ejecutan las estructuras de datos internas.

A este tipo de técnica se le conoce también como Técnica de Caja Transparente o de Cristal. Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento.

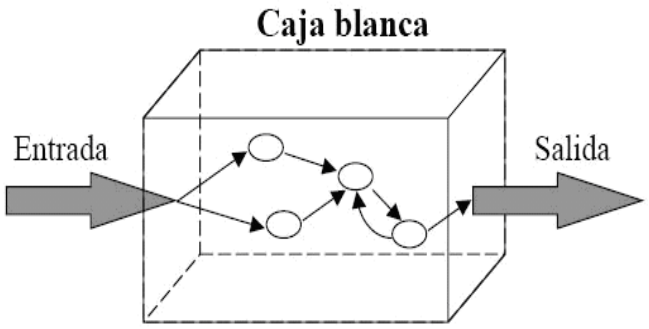


Figura 1: Pruebas de Caja Blanca

El objetivo de la técnica 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.

1.8.2 Técnica de Caja Negra

Las pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa, centrándose en los requisitos funcionales del software [11].

Los casos de prueba de la caja negra pretenden demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Los casos de prueba deben satisfacer los siguientes criterios:

- Reducir, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales.
- Que digan algo sobre la presencia o ausencia de clases de errores.

A menudo los desarrolladores de software experimentados dicen que la prueba nunca termina, simplemente se transfiere del ingeniero de software al cliente. Cada vez que el cliente usa el programa lleva a cabo una prueba. Aplicando el diseño de casos de prueba, el ingeniero del software puede conseguir una prueba más completa y descubrir y corregir así el mayor número de errores antes de que comiencen las pruebas del cliente.

1.9 Niveles de pruebas del Software

Una definición que se puede dar de las pruebas es la siguiente: “Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto”[8].

La prueba es un elemento crítico para la calidad del software. La importancia de los costos asociados a los errores promueve la definición y aplicación de un proceso de pruebas minuciosas y bien planificadas. Las pruebas permiten validar y verificar el software, entendiendo como validación del software el proceso que determina si el software satisface los requisitos y verificación como el

proceso que determina si los productos de una fase satisfacen las condiciones de dicha fase. La prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo.

Se distinguen los siguientes niveles de pruebas [12]:

1.9.1 Prueba de desarrollador

Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas pruebas han sido consideradas sólo para la prueba de unidad, aunque en la actualidad en algunos casos pueden ejecutar pruebas de integración. Se recomienda que estas pruebas cubran más que las pruebas de unidad.

1.9.2 Prueba independiente

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

1.9.3 Prueba de Unidad

Es la prueba enfocada a los elementos testeables más pequeños del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos y que ellos funcionen como se espera.

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.

Lo importante en este tipo de pruebas es que se deben tener claros los siguientes aspectos:

- Los datos de entrada son conocidos por el Tester o Diseñador de Pruebas y éstos deben ser preparados con minuciosidad, ya que el resultado de las pruebas depende de estos.
- Se debe conocer qué componentes interactúan en cada caso de prueba.

- 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, pasando a la siguiente.

Estas pruebas no descubrirán todos los errores del código, éstas sólo prueban que el segmento de código que se revisa esté lógicamente correcto. No descubrirán errores de integración.

Estas pruebas aisladas proporcionan algunas ventajas básicas:

- **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.
- **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un alto grado de seguridad de que el código está funcionando correctamente, de esta manera se facilitan las pruebas de integración.
- **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- **Separación de la interfaz y la implementación**: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de éstas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- **Se da más seguridad al programador**: Normalmente, la persona que ha programado un módulo no es la misma que la que tiene que corregir sus errores. Esto crea una sensación de inseguridad al programador, ya que a la hora de corregir un error no tiene la certeza de que su corrección no va a afectar a otros módulos que desconoce. Las pruebas unitarias aseguran que una corrección no repercute 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 sólo en verificar la correcta cooperación de las distintas unidades y en los funcionamientos generales del programa.

La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo.

Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada. Se examinan las estructuras de datos

locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento.

Se ejercitan todos los caminos independientes (caminos básicos) de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y finalmente, se prueban todos los caminos de manejo de errores.

1.9.4 Prueba de Integración

Aún cuando los módulos de un programa funcionen bien por separado, es necesario probarlos conjuntamente: un módulo puede tener un efecto adverso o inadvertido sobre otro módulo; las subfunciones, cuando se combinan, pueden no producir la función principal deseada; la imprecisión aceptada individuamente puede crecer hasta niveles inaceptables al combinar los módulos; los datos pueden perderse o malinterpretarse entre interfaces, entre otros.

Por lo tanto, es necesario probar el software ensamblando todos los módulos probados previamente. Este es el objetivo de la pruebas de integración.

Es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Estas pruebas descubren errores o son incompletos en las especificaciones de las interfaces de los paquetes. Esta prueba debe ser responsabilidad de desarrolladores y de independientes, sin solaparse las pruebas.

Es el proceso de combinar y probar múltiples componentes juntos. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

1.9.5 Prueba de sistema

Este tipo de pruebas tiene como propósito ejercitar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema (hardware, otro software, entre otros) y que realizan las funciones adecuadas.

Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados.

En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos bien formados de comportamiento de caso de uso son implementados.

Concretamente se debe comprobar:

- Que se cumplen los requisitos funcionales establecidos.

- El funcionamiento y rendimiento de las interfaces hardware, software y de usuario.

- La adecuación de la documentación de usuario.

- El rendimiento y respuesta en condiciones límite y de sobrecarga.

Para la generación de casos de prueba de sistema se utilizan técnicas de caja negra. Este tipo de pruebas se suelen hacer inicialmente en el entorno del desarrollador, denominadas Pruebas Alfa y seguidamente en el entrono del cliente, denominadas Pruebas Beta.

Tipos de Pruebas del Sistema:

- Prueba de Recuperación: Es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleve a cabo apropiadamente.

- Prueba de Seguridad**:** Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de acceso impropios.

- Prueba de Resistencia**:** Están diseñadas para enfrentar a los programas con situaciones anormales.

- Prueba de Rendimiento**:** Está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

De forma general, las pruebas de sistema verifican que el sistema completo satisfaga sus requerimientos funcionales y operacionales. El sistema tiene que demostrar ser tanto funcionalmente correcto como robusto, antes de que sea entregado para las pruebas de aceptación. El equipo de pruebas del sistema está compuesto por los desarrolladores y guiados por una persona especialista en la aplicación desarrollada.

1.9.6 Prueba de aceptación

El usuario debe ser el que realice las pruebas, ayudado por personas del equipo de pruebas, siendo deseable, que sea el mismo usuario quien aporte los casos de prueba.

Estas pruebas se caracterizan por:

- Participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas.

- Están enfocadas a probar los requisitos de usuario, o mejor dicho, a demostrar que no se cumplen los requisitos, los criterios de aceptación o el contrato.

- Corresponden a la fase final del proceso de desarrollo de software.

Se puede resumir que todos los niveles de prueba son importantes para verificar que el software se ha desarrollado correctamente. Son básicamente pruebas funcionales sobre el sistema completo y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable de cara al cliente; se realizan una vez pasadas todas las pruebas de integración por parte del desarrollador.

1.10 Tipos de prueba del Software

Las pruebas son diseñadas para encontrar el mayor número de errores. Cada tipo de prueba tiene un objetivo específico y una técnica que lo soporte. A continuación se muestran los tipos de pruebas basado en dimensiones de calidad.

1.10.1 Prueba de Funcionalidad

La prueba de funcionalidad es la que se encarga de determinar los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a datos [13].

Tipos de Pruebas de Funcionalidad

- **Función:** Pruebas fijando su atención en la validación de las funciones, métodos, servicios y casos de uso.
- **Seguridad:** Asegurar que los datos o el sistema solamente es accedido por los actores deseados.
- **Volumen:** Enfocada verificando las habilidades de los programas para manejar grandes cantidades de datos, tanto como entrada, salida o residente en la BD.

1.10.2 Prueba de Usabilidad

La prueba de usabilidad determina la calidad de la experiencia de un usuario en la forma en la que éste interactúa con el sistema, se considera la facilidad de uso y el grado de satisfacción del usuario.

- **Usabilidad:** Prueba enfocada a factores humanos, estéticos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea, documentación de usuarios y materiales de entrenamiento.

1.10.3 Prueba de Fiabilidad

La prueba de fiabilidad determina la capacidad del programa para soportar entradas incorrectas.

Tipos de Pruebas de Fiabilidad

- **Integridad:** Enfocada a la valoración de la robustez (resistencia a fallos).

- **Estructura:** Enfocada a la valoración de adherencia a su diseño y formación. Este tipo de prueba es hecha a las aplicaciones Web asegurando que todos los enlaces están conectados, el contenido deseado es mostrado y no hay contenido huérfano.
- **Stress:** Enfocada a evaluar cómo el sistema responde bajo condiciones anormales. (Insuficiente memoria, servicios y hardware no disponible, además de recursos compartidos no disponibles).

1.10.4 Prueba de Rendimiento

Las pruebas de rendimiento determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones, entre otros.

Tipos de Pruebas de Rendimiento

- **Benchmark:** es un tipo de prueba que compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.
- **Contención:** Enfocada a la validación de las habilidades del elemento a probar para manejar aceptablemente la demanda de múltiples actores sobre un mismo recurso (registro de recursos, memoria, entre otros.)
- **Carga:** Usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante. La variación en carga es simular la carga de trabajo promedio y con picos que ocurren dentro de tolerancias operacionales normales.
- **Performance profile:** Enfocadas a monitorear el tiempo en flujo de ejecución, acceso a datos, en llamada a funciones y sistema para identificar y direccional los cuellos de botellas y los procesos ineficientes.

1.10.5 Prueba de Soportabilidad

La prueba de soportabilidad es la que determina hasta donde puede soportar el programa determinadas condiciones extremas.

Tipos de Pruebas de Soportabilidad

- **Configuración:** Enfocada a asegurar que funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema.
- **Instalación:** Enfocada a asegurar la instalación en diferentes configuraciones de hardware y software bajo diferentes condiciones (insuficiente espacio en disco, entre otros.)

1.11 Metodología de Desarrollo de Software

Con la evolución de la informática se hace necesario seguir ciertas pautas predefinidas para desarrollar un software de calidad, para esto se debe utilizar una metodología que sea lo suficientemente precisa como para que todo el mundo la pueda seguir y sea de utilidad como pauta común, pero también debe ser adaptable para poder aplicarse en distintos proyectos [14].

El uso de las metodologías permite definir una guía para desarrollar un proceso de software con un alto nivel de calidad. Toda metodología debe de ser adaptada al contexto del proyecto (tiempo de desarrollo, tiempo del sistema, recursos humanos y técnicos). A lo largo de los años se han propuesto una serie de metodologías pero en este capítulo se abordará la metodología RUP, por ser la utilizada en el proceso de desarrollo de software implantado en el proyecto Sistema Único de Aduanas (SUA).

1.11.1 Conceptos del Proceso Unificado de Desarrollo de Software (RUP)

El Proceso Unificado Racional es un proceso de desarrollo de software, el cual constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El ciclo de vida RUP es una implementación del desarrollo en espiral y organiza las tareas en fases e iteraciones.

En su modelación se definen como elementos:

Trabajadores (quién): Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.

Actividades (cómo): Tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.

Artefactos (qué): Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

Flujo de Actividades (cuándo): Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

1.11.2 Flujo de trabajo de Pruebas en la metodología RUP

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. A continuación se analizará el flujo de trabajo de pruebas según esta metodología. En la siguiente figura se muestra cómo el flujo de trabajo de prueba aparece en todas las fases.

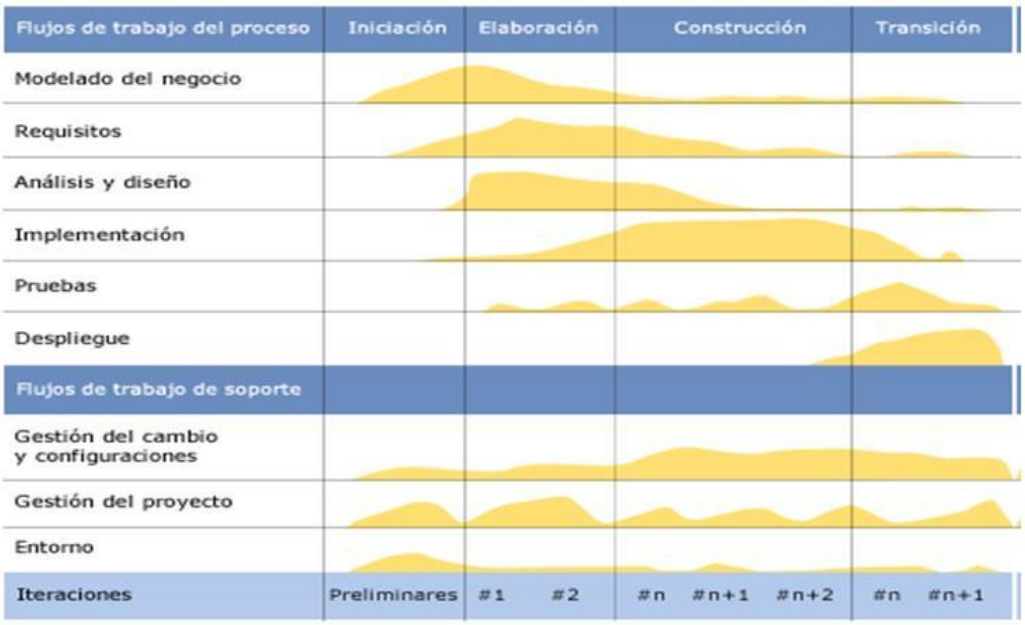


Figura 2: Flujos de trabajos y fases de la metodología RUP.

1.11.2.1 Relación con las fases de RUP

RUP propone que en cada una de las fases las pruebas se comporten de la forma siguiente:

- **Inicio:** En esta fase se desarrolla el prototipo exploratorio de demostración; no requiere la elaboración de pruebas.
- **Elaboración:** Esta es la fase donde se prueban los componentes ejecutables que se han implementado y que deben corresponderse con la arquitectura básica de la aplicación.
- **Construcción:** En la fase constructiva del software se desarrollan los casos de prueba y procedimientos de prueba para hacerlos.
- **Transición:** Ya el producto está en su entorno de operación por lo que es probado por usuarios reales.

1.11.2.2 Relación con los demás flujos de trabajo

- El flujo de trabajo de **Requerimientos** captura los requisitos para el producto del software, los cuales son una de las entradas principales para identificar qué pruebas se deben ejecutar.
- Durante el **Análisis y Diseño** se determina el diseño apropiado para el software, el cual es otra de las entradas principales para identificar qué pruebas ejecutar.
- En la disciplina de **Implementación** se producen versiones operacionales del sistema (builds) que son validados por las pruebas. Dentro de una iteración múltiples builds serán probados (1 por ciclo de prueba).

- La disciplina de **Despliegue** entrega el producto de software completo al usuario final. Antes el software es validado mediante pruebas, aunque las pruebas de aceptación y pruebas betas son ejecutadas como parte del despliegue.
- La **Gestión de proyecto** planifica el proyecto y las iteraciones, descrito en el Plan de Iteración, artefacto que es una de las principales entradas usada cuando se va a definir la misión de evaluación para la prueba.
- En el flujo de trabajo **Gestión de configuración y cambios** se controlan los cambios dentro del proyecto. Las pruebas verifican que cada cambio ha sido completado apropiadamente.

El flujo de trabajo de prueba es el encargado de evaluar la calidad del producto que se está desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida.

Las actividades de este flujo deben comenzar en los inicios del proyecto con el Plan de Pruebas (el cual contiene información sobre los objetivos generales y específicos de las prueba en el proyecto, así como las estrategias y recursos con que se dotará a esta tarea), o incluso antes con alguna evaluación durante la fase de inicio y continuará durante todo el proyecto.

1.11.2.3 Trabajadores del flujo de trabajo de prueba

Para llevar a cabo todo el proceso de pruebas RUP define los siguientes roles:

- **Administrador de Prueba**: Este trabajador es el responsable del éxito de la prueba.
- **Analista de Prueba:** Identifica y define las pruebas requeridas, monitorea el progreso de la prueba y el resultado en cada ciclo de prueba y evalúa la calidad total experimentada como un resultado de las actividades de prueba.
- **Diseñador de prueba:** Responsable de definir el método de prueba y asegurar su implementación exitosa.
- **Probador:** Es el responsable de varias actividades de las pruebas, las cuales incluyen la conducción de las pruebas necesarias y el registro del resultado de la prueba.

En el flujo de trabajo de prueba los objetivos principales son: planificar las pruebas necesarias en cada iteración incluyendo las pruebas de integración y las pruebas de sistema, además del diseño e implementación de las pruebas creando los casos de prueba que especifican qué probar, los procedimientos y componentes de prueba. Otro objetivo fundamental es la realización de las pruebas y manejar los resultados de cada prueba sistemáticamente.

1.11.2.4 Artefactos del flujo de trabajo de pruebas

Un producto o artefacto es un trozo de información que es producido, modificado o usado durante el proceso de desarrollo de software. (Puede ser un documento, modelo, elemento de un modelo o subsistema). No siempre en todo proyecto se crean los mismos artefactos, esto dependerá de las características del proyecto y los requisitos del cliente. Para el flujo de trabajo de pruebas se tienen los siguientes artefactos [15]:

Artefacto: Modelo de prueba

Describe principalmente cómo se prueban los componentes ejecutables (como las construcciones) en el modelo de implementación con pruebas de integración y de sistema. El modelo de pruebas describe también cómo se han probado aspectos específicos del sistema, por ejemplo, si la interfaz de usuario es utilizable y consistente o si el manual de usuario del sistema cumple con su cometido y describe el cumplimiento de los requisitos funcionales y no funcionales del sistema. Es una colección de casos de prueba, procedimientos de prueba y componentes de prueba.

Artefacto: Caso de prueba

Cada prueba es especificada mediante un documento que establece las condiciones de ejecución, las entradas de la prueba y los resultados esperados. Estos casos de prueba son aplicados como pruebas de regresión en cada iteración. Cada caso de prueba llevará asociado un procedimiento de prueba con las instrucciones para realizar la prueba y dependiendo del tipo de prueba, dicho procedimiento podrá ser automatizable mediante un script de prueba.

Existen varios casos de prueba que son realizados en la práctica, por ejemplo:

- Un caso de prueba que especifica cómo probar un caso de uso o un escenario específico de un caso de uso. Estos incluyen la verificación del resultado de la interacción entre los autores y el sistema, que satisfacen las precondiciones y postcondiciones especificadas por el caso de uso, como es el caso de las pruebas de caja negra.

- Otro de caso de prueba es aquel que especifica cómo probar una realización de caso de uso-diseño o un escenario específico de la realización. Un caso de uso de este tipo puede incluir la verificación de la interacción de los componentes que implementan dicho caso de uso como por ejemplo las prueba de caja blanca.

Artefacto: Procedimiento de prueba

Un procedimiento de prueba especifica cómo realizar uno o varios casos de prueba ó partes de éstos. Puede ser una instrucción para un individuo sobre cómo realizar un caso de prueba manualmente, o una especificación de cómo interactuar manualmente con una herramienta de automatización de pruebas para crear componentes ejecutables de prueba. El cómo llevar a cabo

un caso de prueba puede ser especificado por un procedimiento de prueba, pero es a menudo útil reutilizar un procedimiento de prueba para varios casos de prueba y reutilizar varios procedimientos de prueba para un caso de prueba.

Artefacto: Componente de prueba

Un componente de prueba automatiza uno o varios procedimientos de prueba o partes de ellos. Éstos pueden ser desarrollados utilizando un lenguaje de guiones o un lenguaje de programación, o pueden ser gravados con una herramienta de automatización de pruebas. Los componentes de pruebas se utilizan también para probar los componentes en el modelo de implementación, proporcionando entradas de pruebas, controlando y motorizando la ejecución de los componentes a probar. Los componentes de pruebas pueden ser implementados usando tecnología de objetos.

Artefacto: Plan de Pruebas

El propósito del Plan de Pruebas es dejar de forma explícita el alcance, el enfoque, los recursos requeridos, el calendario y los responsables del proceso de pruebas. El Plan de Pruebas describe las estrategias, recursos y planificación de la prueba, el nivel de cobertura de prueba y de código necesario, además del porcentaje de prueba que deberían ejecutarse con un resultado específico. Cada prueba debe dejar claro qué tipo de propiedades se quieren probar así como tener una visión clara de cómo medir los resultados.

La construcción de un buen Plan de Pruebas es la piedra angular y en consecuencia el principal factor crítico de éxito para la puesta en práctica de un proceso de pruebas que permita entregar un software de mejor nivel.[16]

Artefacto: Defecto

Un defecto es un síntoma de un fallo software o un problema descubierto en una revisión, el cual puede ser utilizado para localizar cualquier cosa que los desarrolladores necesiten registrar cómo síntoma de un problema en el sistema.

Artefacto: Evaluación de Prueba

Es una evaluación de los resultados de los esfuerzos de prueba, tales como la cobertura del caso de prueba, de código y el estado de los defectos. Los diseñadores evalúan los resultados de la prueba comparando los resultados obtenidos con los objetivos trazados en el Plan de Pruebas. Se realizan métricas que les permiten determinar el nivel de calidad del software y qué cantidad de pruebas es necesario realizar.

Estrategia de prueba:

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a la construcción correcta del software. Es el proceso

de analizar cómo plantear las pruebas en el ciclo de vida. La estrategia de pruebas suele seguir estas etapas: comenzar pruebas a nivel de módulo, continuar hacia la integración del sistema completo y a su instalación y culminar con la aceptación del producto por parte del cliente. La implementación de estrategias de pruebas exitosas pueden reducir los costes de pruebas en un 32%. Las estrategias de pruebas es la clave para la finalización exitosa de cualquier actividad de pruebas de software. En el proyecto la prueba a veces requiere mayor esfuerzo que cualquier otra actividad de la Ingeniería de Software, si se efectúa sin un plan estratégico, el tiempo se desaprovecha y el esfuerzo es consumido innecesariamente y, en el peor de los casos, los errores inadvertidos quedarán sin detectar, por tanto, puede ser muy conveniente establecer una estrategia sistemática para probar el software.

1.12 Herramientas para implementar pruebas unitarias

Una herramienta ofrece los servicios necesarios para dar soporte a una tarea determinada. Para la realización de pruebas unitarias existen herramientas y entornos de desarrollo (frameworks) que facilitan su creación en multitud de lenguajes de programación.

Algunas de estas son [17]:

JUnit: Entorno de pruebas para Java creado por Erich Gamma y Kent Beck. Se encuentra basado en SUnit creado originalmente para realizar pruebas unitarias para el lenguaje Smalltalk. Es un framework de pruebas unitarias en Java y permite la automatización del código de los casos de prueba Java.

Con JUnit se logra ejecutar pruebas automatizadas fácilmente. Contiene mecanismos para recolectar resultados de manera estructurada, produce varios tipos de reportes a partir de los resultados obtenidos en la ejecución de las pruebas, permite que existan relaciones entre las pruebas y que unas reutilicen código de otras. Además, soporta la jerarquía entre las pruebas pudiendo establecerse la prioridad y el orden de unas con otras.

DbUnit: DbUnit es una extensión de JUnit para proyectos que utilizan bases de datos. Entre otras cosas, su objetivo es colocar la base de datos en un estado conocido para la ejecución de las pruebas.

Ayuda a evitar muchos problemas que usualmente ocurren cuando una prueba corrompe la base de datos y subsiguientes pruebas puedan fallar o empeorar el daño. DbUnit tiene la habilidad de exportar e importar los datos de la base de datos hacia y desde ficheros XML y puede verificar si la información contenida en la base de datos coincide con conjuntos de valores esperados.

SimpleTest: Entorno de pruebas para aplicaciones realizadas en PHP. Es similar a JUnit / PHPUnit. Compatible con los objetos simulados y se puede utilizar para automatizar las pruebas de regresión de aplicaciones web con un cliente de secuencias de comandos HTTP que puede

analizar las páginas HTML y simular cosas como hacer clic en los enlaces y la presentación de formularios [18].

PHPUnit: Framework para realizar pruebas unitarias en PHP. Es un miembro de la familia xUnit de los marcos de las pruebas, proporciona un marco que hace que la escritura de las pruebas sea fácil y tiene la funcionalidad de ejecutar fácilmente las pruebas y analizar sus resultados.

Zend Framework [19] Es un framework de código abierto, orientado a objeto, donde cada componente tiene una baja dependencia respecto a otros componentes, lo cual permite a los desarrolladores utilizar los componentes por separado. Entre estas herramientas podemos encontrar:

- Componentes modelo-vista-controlador (MVC).

- Abstracción a Base de Datos.

- WebServices.

Facilidades que aporta el uso de este framework:

- Simplicidad en el uso.

- Códigos más estables y con menos probabilidad de error.

- Simplicidad en el mantenimiento del código.

Este framework insiste fundamentalmente en la calidad del código, a través de pruebas unitarias, utilizando PHPUnit [20].

Symfony [21] es un framework para construir aplicaciones web con PHP. Su licencia es de tipo software libre. Contiene un conjunto de herramientas y utilidades que simplifican el desarrollo de las aplicaciones web. Emplea el tradicional patrón de diseño Modelo Vista Controlador para separar las distintas partes de una aplicación web. La vista convierte la información obtenida por el modelo en páginas web a las que acceden los usuarios. El controlador es el encargado de ordenar los demás elementos y convertir las peticiones del usuario en operaciones sobre el modelo y la vista.

Sus características principales son:

- Fácil de instalar y configurar en la mayoría de las plataformas.
- Independiente del sistema gestor de bases de datos.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y es adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.

- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.
- Incorpora Lime, un sistema propio de Symfony para realizar pruebas unitarias.
- Validación automatizada y relleno automático de datos en los formularios.
- Usa técnicas de escape para evitar inyección de código.
- Sistema de caché eficiente.

Symfony es un framework para desarrollar aplicaciones web con PHP 5. Añade una nueva capa por encima de PHP y proporciona herramientas que simplifican el desarrollo de las aplicaciones web complejas.

Conclusiones

En el presente capítulo se analizó el estado actual de las pruebas del software y de los distintos tipos y niveles de pruebas que existen, enfocándose principalmente en las pruebas de unidad. Se realizó además una breve descripción de la metodología RUP, así como del flujo de trabajo de pruebas según esta metodología para entender mejor su funcionamiento. Finalmente se analizaron distintas herramientas y framework que se utilizan para realizar las pruebas de unidad, haciendo énfasis en el framework Symfony, pues es el utilizado por el proyecto Sistema Único de Aduanas.

CAPÍTULO 2: PROCEDIMIENTO PARA LA REALIZACIÓN DE PRUEBAS DE UNIDAD

Introducción

Las pruebas de unidad tienen un impacto importante en el código y la calidad del producto final, por esta razón en el presente capítulo se ofrece una propuesta de solución para garantizar la realización de este tipo de pruebas. Se elabora un procedimiento para ser aplicado en el proyecto Sistema Único de Aduanas (SUA). Se define el alcance del procedimiento con el objetivo de mostrar la magnitud que tendrá el mismo especificando qué y cómo se va a probar, también los objetivos que se persiguen, las actividades propuestas a desarrollar para lograr esos objetivos, así como una descripción de las mismas. Se utiliza el Plan de Pruebas y además se crea y describe de forma detallada la Plantilla de Casos de Prueba de Unidad y la automatización de estas pruebas en el framework Symfony.

2.1 Principal problema para realizar las pruebas unitarias en el proyecto Sistema Único de Aduanas (SUA).

Pruebas unitarias manuales

En el Sistema Único de Aduanas (SUA) actualmente las pruebas unitarias se realizan de manera manual, lo cual es un trabajo muy tedioso y a la larga dificulta el desarrollo posterior de estas pruebas. Esto se debe principalmente a la falta de conocimiento respecto al tema en cuestión.

Con cambios constantes en los requisitos las pruebas, principalmente las unitarias, deben poder ser repetibles o reutilizables, independientes, profesionales y automatizables; lo cual ayudan a producir códigos más seguros que si se realizan de manera manual [22].

- **Reutilizables o Repetibles**: No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. Las pruebas deben poder ser ejecutadas varias veces.
- **Independientes**: La ejecución de una prueba no debe afectar a la ejecución de otra, es decir, éstas deben ser independientes de otras pruebas para que ellas se ejecuten solas.
- **Profesionales**: Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, entre otros.
- **Automatizables**: La automatización es una de las formas de mantener la calidad de las pruebas, generando cada vez más grandes y complejas cantidades de código con menos esfuerzo.

Por esta razón se describirá un procedimiento para la realización de estas pruebas de forma automática.

2.2 Automatización de pruebas

Cualquier programador con experiencia en el desarrollo de aplicaciones web conoce de sobra el esfuerzo que supone probar correctamente la aplicación. Crear casos de prueba, ejecutarlos y analizar sus resultados es una tarea tediosa. Además, es habitual que los requisitos de la aplicación varíen constantemente, con el consiguiente aumento del número de versiones de la aplicación y la refactorización continua del código, la realización de las pruebas de forma automática facilitan todo el trabajo.

Éste es el motivo por el que la automatización de pruebas es una recomendación, útil para crear un entorno de desarrollo satisfactorio. Los conjuntos de casos de prueba garantizan que la aplicación hace lo que se supone que debe hacer.

Incluso cuando el código interno de la aplicación cambia constantemente, las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación. Además, este tipo de pruebas obligan a los programadores a crear pruebas en un formato estandarizado y muy rígido que pueda ser procesado por un framework de pruebas.

En ocasiones, las pruebas automatizadas pueden reemplazar la documentación técnica de la aplicación, ya que ilustran de forma clara el funcionamiento de ésta.

2.3 Pruebas unitarias en Symfony

En la descripción del procedimiento para realizar las pruebas unitarias se detallarán los pasos para implementar estas pruebas de forma automática en Symfony, a continuación se darán las principales características y ventajas por las que se utilizará este framework.

El framework Symfony se ha convertido en uno de los frameworks de PHP más populares gracias a sus características avanzadas y su gran documentación [23]. Además es el framework utilizado por el proyecto Sistema Único de Aduanas, para el cual se va a elaborar el procedimiento que describe este trabajo.

Symfony permite crear pruebas unitarias, gestionar correctamente los errores, incluir validación de datos y por supuesto crear una aplicación muy segura. Este framework ya dispone de todas las herramientas necesarias para incluir cada una estas opciones sin necesidad de escribir mucho código. Todo esto es así porque no sólo consiste en código PHP, sino que también utiliza las mejores prácticas para crear aplicaciones profesionales para el mundo empresarial.

La validación, la gestión de errores, las pruebas y la seguridad están completamente integradas en Symfony. Ésta es una más de las razones por las que se debería utilizar este framework para desarrollar proyectos del mundo real.

Un buen conjunto de pruebas muestra la salida que produce la aplicación para una serie de entradas de prueba, por lo que es suficiente para entender el propósito de cada método. Symfony aplica este principio a su propio código. El código interno del framework se valida mediante la automatización de pruebas. Además cuenta con un entorno de pruebas, el cual se utiliza para ejecutar automáticamente las pruebas unitarias [24].

A continuación se detallan sus principales ventajas en cuanto a varios aspectos:

- **Configuración:** Permite establecer diferentes opciones de configuración para cada entorno. Un entorno es un conjunto de opciones que permiten variar el comportamiento de la aplicación en función de si se ejecuta en el servidor de desarrollo o en el de producción.
- **Depuración:** Symfony incluye muchas utilidades para ayudar a los programadores a depurar los errores más fácilmente, como por ejemplo los archivos de log, la barra de depuración web y las excepciones útiles.
- **Objetos de Symfony:** Incluye varios objetos que abstraen las necesidades habituales de los proyectos web: la petición, la respuesta, el usuario, los mensajes de log, el sistema de enrutamiento y el gestor de la cache de la vista.
- **Seguridad:** Incluye protección frente a ataques de tipo XSS y CSRF. Estas opciones se pueden configurar desde la línea de comandos o editando el archivo de configuración. El framework de formularios también incluye varias medidas de seguridad.
- **Formularios:** Incluye un subframework de formularios, con numerosos widgets y validadores. Uno de los puntos fuertes de estos formularios es que sus plantillas se pueden personalizar muy fácilmente.
- **Pruebas:** Para realizar las pruebas unitarias en Symfony se emplea la librería lime, que incluye numerosos métodos para pruebas. También se pueden probar los objetos Propel mediante una base de datos específica y unos archivos de datos específicos.

2.3.1 Automatización de las pruebas unitarias utilizando Symfony

Symfony introduce herramientas y utilidades para automatizar las pruebas, las cuales aseguran la calidad de la aplicación incluso cuando el desarrollo de nuevas versiones es muy activo.

Las pruebas unitarias de Symfony son archivos PHP cuyo nombre termina en Test.php y que se encuentran en el directorio test/unit/ de la aplicación. Su sintaxis es sencilla y fácil de leer [23].

La tarea test-unit, que se utiliza para ejecutar las pruebas unitarias desde la línea de comandos, admite como argumento una serie de nombres de pruebas o un patrón de nombre de archivos.

2.3.2 Propel y las pruebas unitarias

Escribir pruebas unitarias para la clase de un modelo es un poco complicado porque requiere una conexión con la base de datos. Aunque se dispone de la conexión que se configura para el entorno de desarrollo, es una buena práctica crear una conexión con la base de datos específica para las pruebas.

El framework Propel es utilizado para la conexión a la base de datos, pues la mayoría de las funcionalidades de los módulos del proyecto Sistema Único de Aduanas necesitan el acceso a los datos de los vuelos, pasajeros, entre otros.

Por defecto, las pruebas se ejecutan en un entorno llamado test, por lo que se configura una base de datos diferente para el entorno test. Para crear pruebas unitarias para la clase del modelo, se crea un archivo llamado propel.php en el directorio bootstrap/ de las pruebas, se inicializa un objeto de tipo configuración para el entorno test, se crea un gestor de bases de datos y se inicializa la conexión Propel cargando el archivo de configuración databases.yml:

```
$configuration = ProjectConfiguration::getApplicationConfiguration('frontend','test', true);

new sfDatabaseManager($configuration);
```

2.4 El framework de pruebas Lime

Symfony incluye su propio framework llamado Lime, el cual se basa en la librería Test::More de Perl, se crea para facilitar la lectura de los resultados de las pruebas. Lime proporciona el soporte para las pruebas unitarias, es más eficiente que otros frameworks de pruebas de PHP y tiene las siguientes ventajas [24]:

- Ejecuta los archivos de prueba en un entorno independiente para evitar interferencias entre las diferentes pruebas. No todos los frameworks de pruebas garantizan un entorno de ejecución “limpio” para cada prueba.
- Las pruebas de Lime son fáciles de leer y sus resultados también lo son. En los sistemas operativos que lo soportan, los resultados de Lime utilizan diferentes colores para mostrar de forma clara la información más importante.
- Symfony utiliza Lime para sus propias pruebas, por lo que el código fuente de Symfony incluye muchos ejemplos reales de pruebas unitarias y funcionales.
- El núcleo de Lime se valida mediante pruebas unitarias.
- Está escrito con PHP, es muy rápido y está bien diseñado internamente. Consta únicamente de un archivo, llamado lime.php y no tiene ninguna dependencia.

Las pruebas que se muestran en los epígrafes siguientes utilizan la sintaxis de Lime, por lo que funcionan directamente en cualquier instalación de Symfony.

2.4.1 Métodos para las pruebas unitarias

El objeto lime_test dispone de un gran número de métodos para las pruebas unitarias, como se muestra en la siguiente tabla.

Método	Descripción
diag($mensaje)	Muestra un comentario, pero no ejecuta ninguna prueba.
ok($prueba, $mensaje)	Si la condición que se indica es true, la prueba tiene éxito.
is($valor1, $valor2, $mensaje)	Compara 2 valores y la prueba pasa si los 2 son iguales (==).
isnt($valor1, $valor2, $mensaje)	Compara 2 valores y la prueba pasa si no son iguales.
like($cadena, $expresionRegular, $mensaje)	Prueba que una cadena cumpla con el patrón de una expresión regular.
unlike($cadena, $expresionRegular, $mensaje)	Prueba que una cadena no cumpla con el patrón de una expresión regular.
cmp_ok($valor1, $operador, $valor2, $mensaje)	Compara 2 valores mediante el operador que se indica.
isa_ok($variable, $tipo, $mensaje)	Comprueba si la variable que se le pasa es del tipo que se indica.
isa_ok($objeto, $clase, $mensaje)	Comprueba si el objeto que se le pasa es de la clase que se indica.
can_ok($objeto, $metodo, $mensaje)	Comprueba si el objeto que se le pasa dispone del método que se indica.
is_deeply($array1, $array2, $mensaje)	Comprueba que 2 arrays tengan los mismos valores.
include_ok($archivo, $mensaje)	Valida que un archivo existe y que ha sido incluido correctamente.
fail()	Provoca que la prueba siempre falle (es útil para las excepciones).

pass()	Provoca que la prueba siempre se pase (es útil para las excepciones).
skip($mensaje, $numeroPruebas)	Cuenta como si fueran $numeroPruebas pruebas (es útil para las pruebas condicionales).
todo()	Cuenta como si fuera 1 prueba (es útil para las pruebas que todavía no se han escrito).

Tabla 1: Métodos del objeto lime_test para las pruebas unitarias.

La sintaxis es tan clara que prácticamente se explica por sí sola. Casi todos los métodos permiten indicar un mensaje como último parámetro. Este mensaje es el que se muestra como resultado de la prueba cuando ésta tiene éxito. La mejor manera de aprender a utilizar estos métodos es utilizarlos.

2.4.2 Parámetros para las pruebas

En la inicialización del objeto lime_test se indica como primer parámetro el número de pruebas que se van a ejecutar. Si el número de pruebas realmente realizadas no coincide con este valor, la salida producida por Lime muestra un aviso.

El método diag() no cuenta como una prueba. Se utiliza para mostrar mensajes, de forma que la salida por pantalla esté más organizada y sea más fácil de leer. Por otra parte, los métodos todo() y skip() cuentan como si fueran pruebas reales. La combinación pass()/fail() dentro de un bloque try/catch cuenta como una sola prueba.

Una estrategia de pruebas bien planificada requiere que se indique el número esperado de pruebas. Indicar este número es una buena forma de validar los propios archivos de pruebas, sobre todo en los casos más complicados en los que algunas pruebas se ejecutan dentro de condiciones y/o excepciones. Además, si la prueba falla en cualquier punto, es muy fácil de verlo porque el número de pruebas realizadas no coincide con el número de pruebas esperadas.

El segundo parámetro del constructor del objeto lime_test indica el objeto que se utiliza para mostrar los resultados. Se trata de un objeto que extiende de la clase lime_output. La mayoría de las veces, como las pruebas se realizan en una interfaz de comandos, la salida se construye mediante el objeto lime_output_color, que muestra la salida coloreada en los sistemas que lo permiten.

2.5 Procedimiento para realizar pruebas de unidad

La construcción de un sistema software tiene como objetivo satisfacer una necesidad planteada por el usuario. Para asegurar que se han alcanzado los niveles de calidad acordados es necesario evaluar el producto software a medida que se va construyendo. Por lo tanto se hace necesario llevar a cabo, en paralelo al proceso de desarrollo, un proceso de evaluación o comprobación de los distintos productos o modelos que se van generando.

Con la ejecución de los diferentes tipos de pruebas mostradas en el siguiente gráfico a lo largo del ciclo de desarrollo, se minimiza el número de defectos y se asegura la entrega de productos satisfactorios que cumplen los niveles de calidad. Se resalta en rojo en qué lugar se encuentran las pruebas de unidad en el ciclo de desarrollo.

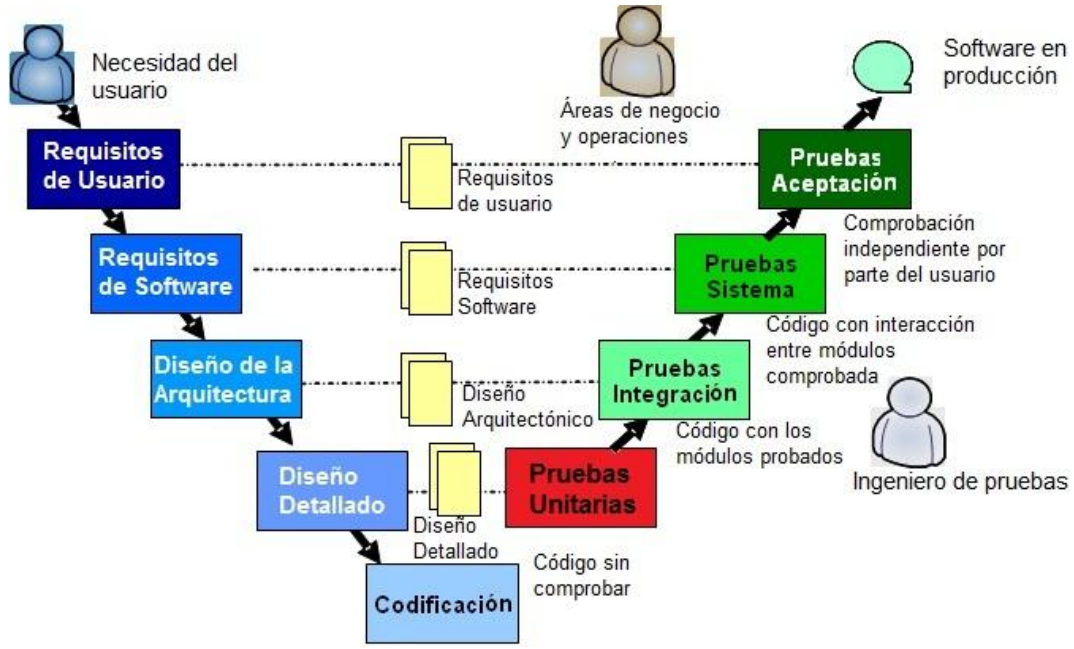


Figura 3: Niveles de pruebas a lo largo del ciclo de desarrollo.

2.5.1 Descripción

El procedimiento para realizar pruebas de unidad definirá de forma detallada los pasos para llevar a cabo estas pruebas. Analiza en detalle cada una de las fases que forma este procedimiento, describiendo, las actividades a realizar y la documentación de entrada y salida que las conforman.

2.5.2 Alcance

Este procedimiento está dirigido a realizar las pruebas de unidad.

¿Qué se va a probar?

Las **funciones individuales o métodos**: se probarán las entradas y las salidas y se comprobará que los valores obtenidos son los esperados. Es decir, se prueba el código aislado, independiente del resto del sistema.

2.5.3 Objetivos

Este procedimiento describe los objetivos de la realización de las pruebas de unidad, el enfoque a seguir en la realización de las mismas por fases, y una descripción detallada de estas.

En base a lo anterior, se definen los objetivos de las pruebas de unidad:

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 trozo de código debe satisfacer. Son fragmentos de unidades estructurales del programa encargados de una tarea en específico. El objetivo principal sería producir las piezas de código de la manera más eficiente y eficaz posible generando pruebas de unidad para las mismas que aseguren su correcto comportamiento.

2.5.4 Fases del Procedimiento

El presente procedimiento de pruebas de unidad se divide en las siguientes fases:

- Planificación de las Pruebas – PP
- Diseño de las Pruebas – DP
- Ejecución de las Pruebas – EP

Figura 4: Fases del procedimiento de la realización de pruebas de unidad.

Para describir este procedimiento, los siguientes epígrafes se dedican a cada fase del proyecto de pruebas, describiendo en cada una las actividades a realizar y las salidas que se generarán.

2.5.4.1 PP1. Fase de Planificación

En esta fase se planificarán las pruebas unitarias que se le realizarán a la aplicación, así como los roles y los recursos necesarios para realizar este procedimiento. Recogiéndose todo en el Plan de Pruebas, en el cual se seleccionan y documentan una serie de elementos que aseguran una mejor calidad del producto. El siguiente gráfico resume las actividades de esta fase:

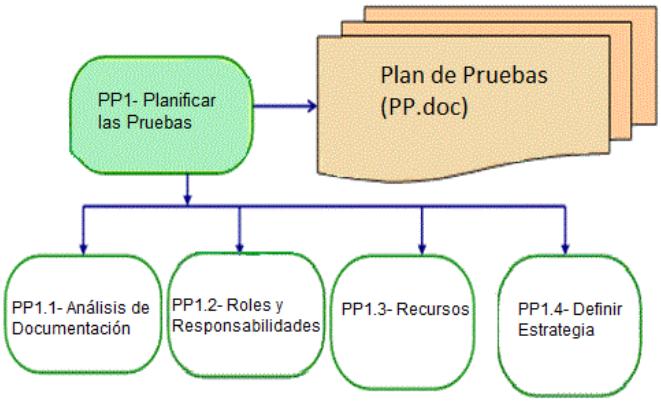


Figura 5: Actividades de la Fase de Planificación de Pruebas.

La documentación de entrada de la fase de planificación del proyecto de pruebas es la documentación propia del proyecto y la de salida, la planificación del proyecto de pruebas que asegure su ejecución en el plazo estimado.

Fase	Salidas
Planificación	Plan de Pruebas (.doc) - Alcance de las Pruebas - Estrategia de las Pruebas - Criterios de Automatización de Pruebas - Roles y Responsabilidades - Recursos Necesarios - Herramientas a utilizar

Tabla 2: Salidas de la Fase de Planificación de Pruebas de Unidad.

PP1.1 Análisis de la Documentación

El análisis de la documentación tiene como objetivo la revisión de toda la documentación recibida como entrada para la fase de planificación. Es necesario que se estudie toda la documentación, incluyendo las plantillas elaboradas en el proyecto que se deben conocer antes de realizar las pruebas. Además se debe estudiar el software y el hardware que se vaya a utilizar para la realización de las pruebas, así como todos los detalles de lo que son las pruebas de unidad.

PP1.1 – Análisis de la Documentación
Objetivo: Estudio de la documentación que sirva de partida para realizar las pruebas unitarias.
Entradas: Glosario de términos. Documento de especificación de requerimientos. Manual de usuario del software a utilizar para la realización de las pruebas. Requerimientos mínimos de hardware y software para realizar las pruebas. Plantilla de Descripción de los Casos de Uso (DCU).
Observaciones: Esta actividad debe realizarse al inicio de la fase de planificación de las pruebas y es de obligatorio cumplimiento.

Tabla 3: Descripción de la actividad Análisis de la Documentación.

PP1.2 Roles y Responsabilidades

El Jefe de Pruebas definirá claramente los siguientes puntos a fin de establecer la composición de los equipos involucrados en la realización de las pruebas y las tareas encomendadas a cada miembro del equipo.

- Definir los equipos involucrados en la realización de las pruebas (se define quien ocupará cada rol). Dependiendo de las particularidades del proyecto se deberán definir los equipos.
- Identificar los recursos necesarios para la ejecución de pruebas.
- Realizar el reparto de responsabilidades entre los distintos participantes.

Durante el procedimiento de pruebas de unidad es necesario asumir 4 roles fundamentales, en la aplicación del procedimiento, se deben especificar todas las responsabilidades de cada trabajador, así como la matriz de responsabilidad, aunque no es de obligatorio cumplimiento esta última.

En la siguiente figura se muestran la relación de los trabajadores con las actividades más importantes a desarrollar.

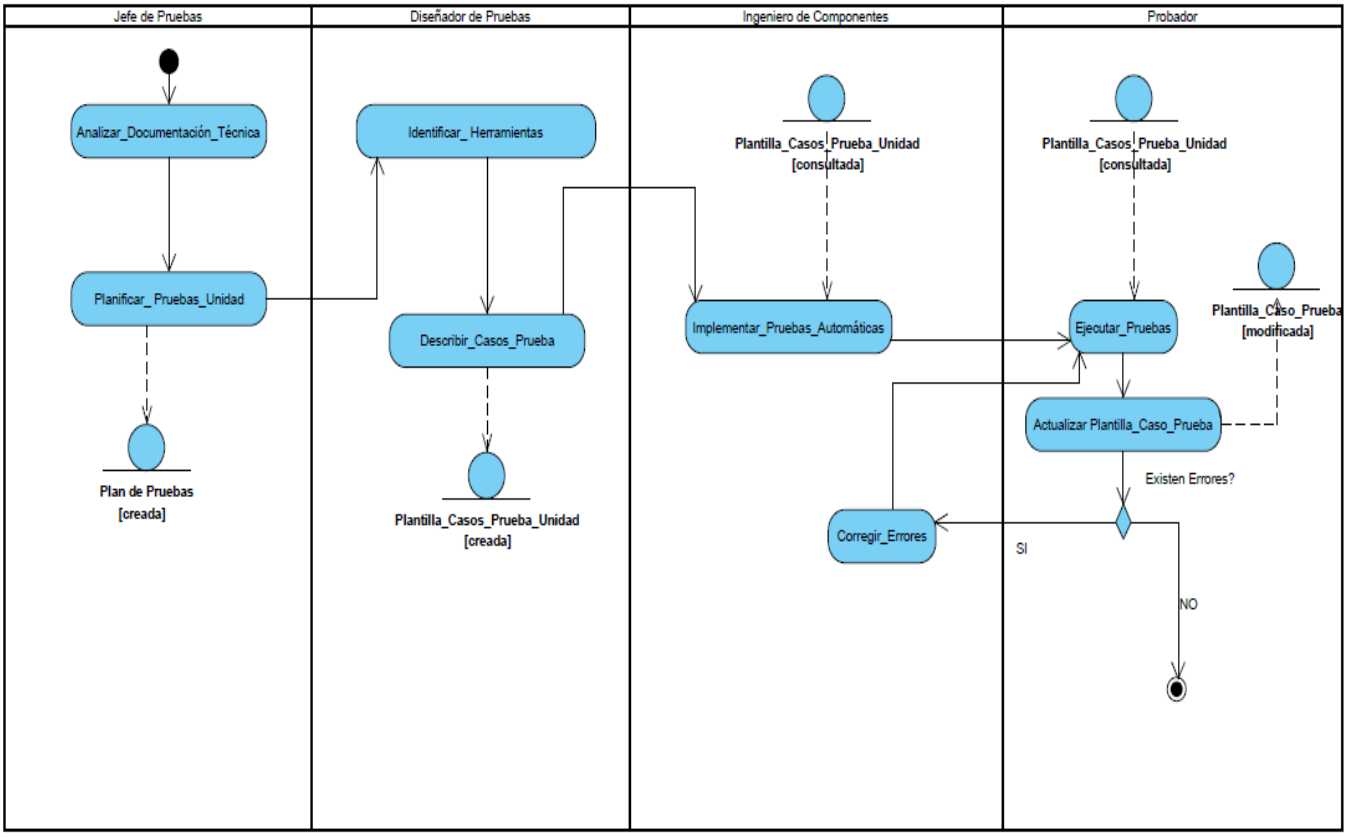


Figura 6: Relación de los roles con sus principales actividades.

PP1.3 Recursos Necesarios

En este paso se especifica el escenario en el que se realizarán las pruebas unitarias, realizando especificaciones de hardware y de software, recogiéndose en la tabla que más adelante se detalla,

también se puede incluir un diagrama de despliegue, este tópico es responsabilidad del jefe de pruebas.

Recursos del Sistema	
Recursos	**Descripción**
Recurso 1	[Características del software] [Características de hardware]
Recurso 2	[Características de hardware] [Características de hardware]

Tabla 4: Descripción de la actividad Recursos Necesarios.

PP1.4 Definir Estrategia

El objetivo de esta tarea es detallar la estrategia a seguir a lo largo del procedimiento de pruebas de unidad. En este epígrafe se describe el flujo de trabajo que será implementado durante todo el período de realización de las pruebas unitarias, de igual forma se detallan los pasos que serán realizados.

PP1.4.1 Propósito

El propósito de la Estrategia de Prueba es el de dirigir todo el proceso estratégico, definir la forma en que será probado el software mediante las pruebas de unidad y definir como serán evaluados los resultados de esa comprobación. Integra un conjunto de actividades y cómo se diseñan los casos de prueba en una serie de pasos bien planificados que dan como resultado la obtención de un producto final con calidad. Proporciona una guía que describe los pasos que se van a llevar a cabo durante el proceso de pruebas.

PP1.4.2 Descripción del flujo de trabajo

El flujo de trabajo se inicia cuando el diseñador de prueba comienza a describir los casos de prueba para su posterior utilización. Se implementan estas pruebas con la ayuda de los casos de prueba. Luego si todas las condiciones están creadas se pasa a la 1ra iteración de pruebas, se ejecutan las pruebas. En la medida que detecten errores, estos serán anotados por el probador que es quien ejecuta las pruebas, y corregidos por el ingeniero de componentes.

En la siguiente figura se muestra el flujo de trabajo de forma general:

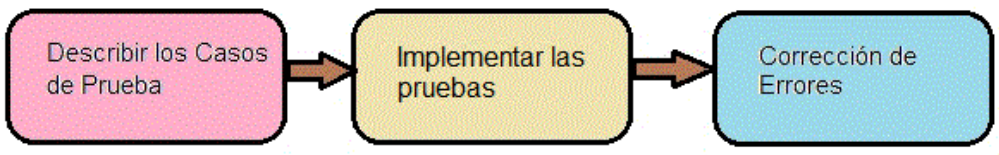


Figura 7: Descripción del Flujo de Trabajo.

PP1.5 Plan de Pruebas

El documento más importante de la fase de planificación es el Plan de Pruebas, pues es en esta fase donde se genera una primera versión de este plan.

Para el diseño del Plan de Pruebas se va a utilizar la Plantilla Plan de Pruebas derivada de la plantilla Plan de Pruebas de RUP. El Plan de Pruebas identificará el propósito, alcance, los elementos de prueba y los recursos necesarios para la ejecución de las pruebas, se va a definir y recomendar la estrategia de prueba. Como se puede apreciar, el Plan de Pruebas de contiene las actividades de la fase de planificación. Ver Anexo 2: Plantilla de Plan de Pruebas.

PP1.5 – Plan de Pruebas		
Objetivo: Elaborar una primera versión del Plan de Pruebas, el cual es la base para el resto del proyecto de pruebas. Dicho plan se irá completando en sucesivas fases del procedimiento.		
Entradas	**Tareas**	**Salidas**
Especificación de Requisitos (.doc) Prototipo de la Interfaz (.html utilizando la herramienta Axure)	Determinar la estrategia de las pruebas. Especificar herramientas a utilizar. Detallar el entorno en que se va a probar.	Documento de Plan de Pruebas (.doc)
Observaciones: Aunque en esta fase se elabora el Plan de Pruebas, éste puede ser modificado a lo largo de la realización de pruebas.		

Tabla 5: Descripción del documento Plan de Pruebas.

2.5.4.2 DP2. Fase de Diseño

La fase de diseño comprende la especificación de casos de prueba necesarios para completar el Plan de Pruebas definido en la fase de planificación de pruebas. El siguiente gráfico resume las actividades de esta fase:

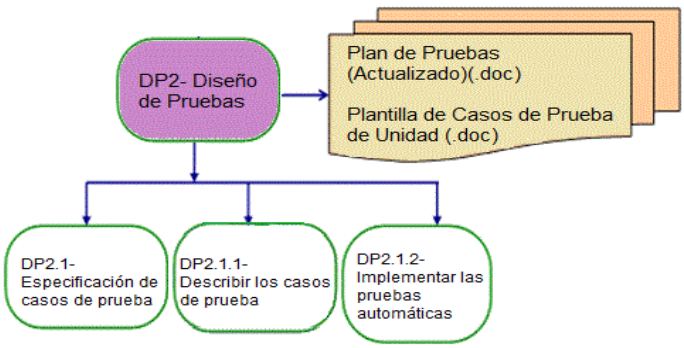


Figura 8: Actividades Fase de Diseño de Pruebas.

Cada caso de prueba describe cada uno de los pasos a ejecutar para comprobar el correcto funcionamiento de cada una de las funciones o métodos y el cumplimiento de la lógica de negocio.

FASE	ENTRADAS	SALIDAS
Diseño	Descripción de los Casos de uso del sistema Prototipo de interfaz Plan de Pruebas Especificación de Requisitos de Pruebas	Plantilla de Casos de Prueba (.doc) Plan de Pruebas (.doc)(actualizado)

Tabla 6: Entradas y Salidas de la Fase de Diseño de Pruebas de Unidad.

DP2.1 Especificación de casos de prueba

En esta actividad se define el nivel más bajo de cada una de las funcionalidades a probar dentro del Plan de Pruebas.

DP2.1 -Especificación de casos de prueba		
Objetivo: Se especifican los casos de prueba, haciendo las respectivas descripciones de los casos de prueba.		
Entradas	**Tareas**	**Salidas**
Especificación de Requisitos (.doc) Prototipo de Interfaz(con la herramienta Axure (.html)) Plan de Pruebas(.doc) Especificación de Requisitos de Pruebas (.doc).	Definir y especificar los casos de prueba. Describir secuencia de pasos para automatizar un Caso de Prueba automático. Actualizar el Plan de Pruebas con la especificación de casos prueba.	Plantilla de Casos de Prueba de Unidad (.doc) Plan de Pruebas Actualizado (.doc).
Observaciones:		

Tabla 7: Descripción de la actividad Especificación de casos de prueba.

A continuación se detalla la información correspondiente al diseño de casos de prueba propuesto y se describe la plantilla del diseño de los casos de prueba.

2.1.1 Descripción de la Plantilla Casos de Prueba de Unidad

El proyecto Sistema Único de Aduanas no cuenta con una Plantilla de Casos de Prueba de Unidad, por lo que en este epígrafe se propone y se describe esta plantilla, para que pueda ser utilizada en el desarrollo de este procedimiento. Ver Anexo 2: Plantilla de Casos de Prueba de Unidad.

La plantilla tiene en su encabezado el nombre del proyecto, del módulo que se va a probar, la versión y el tipo de prueba que se realiza. Luego se detalla en una tabla el nombre de las funcionalidades que se van a probar del módulo escogido, así como una pequeña descripción de éstas.

Funcionalidades a Probar	Descripción
[Nombre de la Funcionalidad]	[breve descripción de la funcionalidad]

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 Lime para comprobar el correcto funcionamiento de las funcionalidades a probar.

Como encabezado se escribe el nombre del tipo de prueba que se diseña. Entre los primeros datos que recoge se encuentra el código del caso de prueba, la iteración en la que se realiza la prueba y una descripción de ésta, así como el nombre del encargado de la prueba.

Como parte de la realización del caso de prueba se toma en cuenta en el primer campo (Funcionalidad), el cual corresponde al nombre de la funcionalidad que será comprobada, el segundo (Método utilizado) es el método de la clase lime_test utilizado, el tercer campo (Recibe) contiene los parámetros que recibe la funcionalidad al ser comprobada, el cuarto es el resultado que se espera que devuelva el método, en el quinto se escribe el resultado que se espera devuelva la prueba luego de ser ejecutada y la última columna es el valor resultado real que obtuvo finalmente dicha prueba. El resultado es satisfactorio si coinciden los resultados reales de las pruebas con los esperados, en caso de no coincidir es detectado un error.

Casos de Prueba de Unidad					
Código del Caso de Prueba:					
Iteración:					
Descripción de la prueba:					
Nombre del encargado:					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado Esperado del método**	**Resultado Esperado de la prueba**	**Resultado Real de la prueba**

Tabla 8: Plantilla Casos de Prueba de Unidad.

2.1.2 Implementar pruebas automáticas

A continuación se explica cómo implementar las pruebas de unidad a través de un ejemplo. Se prueba una función llamada strtolower, la cual elimina los caracteres problemáticos de una cadena de texto. La función elimina todos los espacios en blanco del principio y del final de la cadena; además,

reemplaza todos los caracteres que no son alfanuméricos por guiones bajos y transforma todas las mayúsculas en minúsculas.

En primer lugar se piensa en todos los posibles casos de funcionamiento de este método y se elabora una serie de entradas de prueba junto con el resultado esperado para cada una, como se muestra en la tabla 9.

Dato de entrada	Resultado esperado
” valor “	”valor“
“valor otrovalor”	“valor_otrovalor”
“-)valor:..=otrovalor-”	“__valor____otrovalor_“
“OtroValor”	“otrovalor”
“¡Un valor y otro valor!”	“_un_valor_y_otro_valor_”

Tabla 9. Lista de un fragmento de casos de prueba para la función que elimina caracteres problemáticos.

El siguiente listado muestra un conjunto típico de pruebas unitarias para la función strtolower (), que son las descritas anteriormente.

Listado 1. Archivo de ejemplo de prueba unitaria, en test/unit/ strtolowerTest.php

```
<?php

include(dirname(__FILE__).'/../bootstrap/unit.php');

$t = new lime_test(5, new lime_output_color());

// strtolower()

$t->diag('strtolower()');

$t->isa_ok(strtolower('Foo'), 'string','strtolower() retorna un string');

$t->is(strtolower('FOO'), 'foo','strtolower()Transforma la entrada a minúscula');

$t->is(strtolower('foo'), 'foo','strtolower() Los caracteres en minúscula se mantienen iguales');

$t->is(strtolower('prueba4'), 'prueba4','strtolower() Deja los caracteres  alfanuméricos sin
cambio');

$t->is(strtolower(''), 'foo', 'strtolower()Transforma los espacios vacíos en foo');
```

En primer lugar se instancia el objeto lime_test. Cada prueba unitaria consiste en una llamada a un método de la instancia de lime_test. El último parámetro de estos métodos siempre es una cadena de texto opcional que se utiliza como resultado del método. Para ejecutar el conjunto de pruebas, se utiliza la tarea test-unit desde la línea de comandos. El resultado de esta tarea en la línea de

comandos es muy explícito, lo que permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente.

La figura siguiente muestra el resultado del ejemplo anterior.

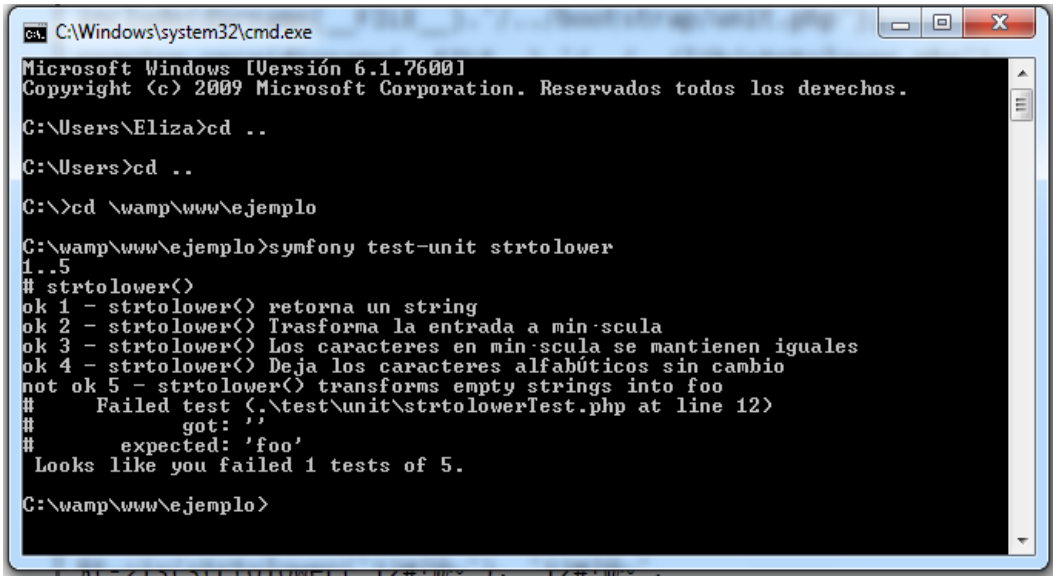


Figura 9: Ejecutando pruebas unitarias desde la línea de comandos.

Estas pruebas también pueden ser implementadas y ejecutadas en un Entorno de Desarrollo Integrado (IDE).

2.5.4.3 EP3. Fase de Ejecución

Se realiza la ejecución de las pruebas implementadas, con la ayuda de los casos de prueba que se hayan identificado anteriormente. El siguiente gráfico muestra las actividades que se deben realizar en esta fase:

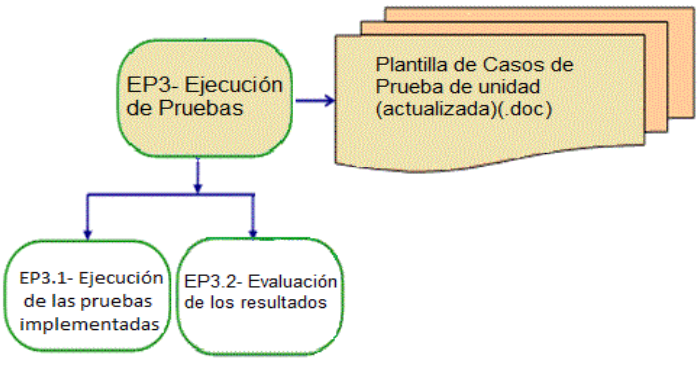


Figura 10: Actividades Fase de Ejecución de Pruebas.

Antes de realizar la ejecución de las pruebas, el probador necesita realizar una validación e inspeccionar los casos de prueba definidos, verificando que se han contemplado todas las pruebas necesarias, que no existen pruebas duplicadas o implícitas en otras y que el tiempo estimado no

sobrepasa la fecha límite de ejecución. Se recomienda que la inspección de los casos de prueba la efectúe una persona diferente a quien diseño los casos de prueba inicialmente, por esta razón la propuesta de este documento es que la realice el probador.

Luego se ejecutan las pruebas unitarias de forma automática, haciendo uso de los casos de prueba diseñados. El probador ejecuta las pruebas, actualiza la Plantilla de Casos de Prueba con los resultados de éstas y al mismo tiempo va registrando los fallos que puedan surgir en la ejecución de las pruebas, luego el ingeniero de componentes corrige los errores encontrados en el código y se vuelven a ejecutar las pruebas hasta no encontrar fallos.

FASE	ENTRADAS	SALIDAS
Ejecución	Especificación de casos de prueba (.doc) Plan de Pruebas(.doc) Resultado pruebas unitarias (.doc) Manual de usuario (.doc)	Plantilla de Casos de Prueba de Unidad (.doc) (actualizada)

Tabla 10: Entradas y Salidas de la Fase de Ejecución de Pruebas de Unidad.

EP3.1 Ejecución de los casos de prueba

Para esta tarea, en general, se recomienda mantener un repositorio de las pruebas implementadas para los casos de prueba automatizados, a fin de mantener una estructura de almacenamiento de los mismos que permita su reutilización. Para cada uno de los casos de prueba que componen los planes de prueba, se ejecutan y se registran los resultados de los mismos en la Plantilla de Casos de Prueba de Unidad. De este modo se tiene un inventario de todos los casos ejecutados, las iteraciones de los mismos, sus incidencias asociadas, así como el histórico de resolución de las mismas.

EP3.1 Ejecución de los casos de prueba		
Objetivo: Ejecutar los casos de prueba, registrando los resultados en la Plantilla de Casos de Prueba de Unidad.		
Entradas	**Tareas**	**Salidas**
Especificación de casos de prueba (.doc) Plan de Pruebas (.doc) Resultado pruebas unitarias (RP<XX>.doc) Manual de usuario.	Ejecutar los casos de prueba especificados. Registrar los resultados en Plantilla de Casos de Prueba de Unidad.	Plantilla de Casos de Prueba de Unidad (.doc) (actualizada con los resultados obtenidos)
Observaciones:		

Tabla 11: Descripción de la actividad Ejecución de los casos de prueba.

EP3.2 Evaluación de los resultados

Una vez obtenidos los resultados después de ejecutadas las pruebas, el siguiente paso es evaluar la información obtenida en los mismos y si se deben ejecutar nuevamente las pruebas. El objetivo de esta evaluación es la obtención de errores, conclusiones y recomendaciones tras las pruebas, así como la toma de decisiones acerca del número de iteraciones de pruebas a realizar.

EP3.2 Evaluación de los resultados		
Objetivo: Evaluar los resultados obtenidos y analizar los errores del código encontrados.		
Entradas	**Tareas**	**Salidas**
Plantilla de Casos de Prueba de Unidad (.doc).	Analizar los resultados obtenidos al ser ejecutadas las pruebas. Actualizar los casos de prueba de unidad.	Plantilla de Casos de Prueba de Unidad(.doc) (Actualizada con los resultados obtenidos)
Observaciones: Se debe tomar la decisión de si realizar una nueva iteración de pruebas o no.		

Tabla 12: Descripción de la actividad Evaluación de Resultados.

Para lograr visualizar la mayor cantidad de errores en el código del software se tienen que considerar cuáles podrían ser los errores más propensos a ocurrir. Esto puede hacerse de varias formas, una de ellas es listando defectos conocidos en la práctica del desarrollo, generando suposiciones a partir de la naturaleza del software, teniendo en cuenta la complejidad de algunos métodos, entre otros. De cualquier forma, tener una idea previa de los defectos que puedan existir será aprovechable para todos los encargados de probar el software.

Para mostrar los resultados de las pruebas unitarias realizadas se detallan a continuación los errores los cuales son recogidos en una tabla con los siguientes datos:

Nro. Reporte de Prueba			
Errores Encontrados:			
Id Caso de Prueba:			
Funcionalidad Probada	**Error Encontrado**	**Clasificación del error**	**Observación**

Tabla 13: Descripción del reporte de los errores encontrados

La descripción de los errores debe ser clara y precisa. Los errores según la prioridad se clasifican en:

1. Prioridad Crítica (Error que necesita ser solucionado inmediatamente).

2. Prioridad Alta.

3. Prioridad Media.

4. Prioridad Baja.

Los reportes de los errores encontrados de las pruebas son recogidos en la Plantilla de Casos de Prueba de Unidad.

Conclusiones

En este capítulo se realizó la descripción del procedimiento de pruebas de unidad diseñado. A lo largo del procedimiento se verificaron y se les dio respuesta a todos los aspectos que se definieron en un principio como objetivos para desarrollar un proceso correcto y eficaz de pruebas de unidad. 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 este trabajo para la aplicación de pruebas de unidad.

CAPÍTULO 3: APLICACIÓN DEL PROCEDIMIENTO PARA REALIZAR PRUEBAS DE UNIDAD Y EVALUACIÓN DE LOS RESULTADOS.

Introducción

El objetivo de este capítulo es aplicar el procedimiento descrito anteriormente y exponer los resultados obtenidos. Se realizarán las actividades y se documentarán las pruebas llenando la Plantilla de Casos de Prueba de Unidad y se implementarán automáticamente, con el objetivo de llegar a conclusiones respecto a los resultados. Para ello se utilizará la estrategia desarrollada para realizar pruebas de unidad. El subsistema seleccionado donde se aplicará el procedimiento es Mecanismo de Control de Acceso y Autenticación, perteneciente al proyecto Sistema Único de Aduanas.

3.1 Descripción del subsistema Mecanismo de Control de Acceso y Autenticación

En la Aduana toda la información que se maneja es de suma importancia, por lo que se debe garantizar su seguridad, integridad y disponibilidad. Este sistema se encarga de gestionar todo lo referente a la autenticación y control de acceso de los usuarios de la entidad. El sistema está compuesto por cinco módulos, de ellos cuatro corresponden a la autenticación y control de acceso, a continuación se relacionan los requisitos del sistema correspondientes a cada módulo.

Requisitos funcionales del Módulo suAdminAuth (Ver Especificación de los Requisitos Funcionales Módulo < suAdminAuth >):

R1 Requisito funcional Autenticar Usuario.

- R1.1 Requisito funcional Comprobar Acceso a Acción.
- R1.2 Requisito funcional Construir Menú de Navegación.

Requisitos funcionales del Módulo suAdminDomain (Ver Especificación de los Requisitos Funcionales Módulo < suAdminDomain >):

R2 Requisito funcional Gestionar Dominio.

- R2.1 Insertar Dominio.
- R2.2 Eliminar Dominio.
- R2.3 Modificar Dominio.
- R2.4 Buscar y Visualizar Dominios.

Requisitos funcionales del Módulo suAdminRole (Ver Especificación de los Requisitos Funcionales, Módulo <suAdminRole>):

R3 Requisito funcional Gestionar Rol.

- R3.1 Insertar Rol.
- R3.2 Eliminar Rol.
- R3.3 Modificar Rol.
- R3.4 Buscar y Visualizar Roles.
- R3.5 Asignar Permisos a Roles.

Requisitos funcionales del Módulo suAdminUser (Ver Especificación de los Requisitos Funcionales Módulo< suAdminUser >):

R6 Requisito funcional Gestionar Usuario.

- R6.1 Insertar Usuario.
- R6.2 Eliminar Usuario.
- R6.3 Modificar Usuario.
- R6.4 Mostrar y Visualizar Usuarios.
- R6.5 Asignar Roles a Usuarios.

3.2 Alcance

Se realizarán pruebas de unidad de forma automática utilizando el framework symfony. Estas pruebas están orientadas a descubrir errores en el código del programa, en este caso se realizan este tipo de pruebas a 3 unidades (módulos) del subsistema Mecanismo de Control de Acceso y Autenticación, los cuales son: suAdminDomain que se encarga de gestionar todo lo referente a los dominios, suAdminRole, el cual gestiona a los usuarios y suAdminUser que gestiona los usuarios o grupos.

3.3 Objetivos

Verificar que el código de las funcionalidades de 3 de los módulos del subsistema Mecanismo de Control de Acceso y Autenticación descritos en el alcance, está funcionando correctamente, se comparan los resultados obtenidos respecto a los resultados que se esperan de los métodos. Y en caso de que existan errores documentarlos e informarlos para su posterior corrección.

3.4 Fases a desarrollar

PP1. Aplicación Fase de Planificación

PP1.1 Análisis de la Documentación

El subsistema Mecanismo de Control de Acceso y Autenticación, donde se van a realizar las pruebas, tiene documentados los siguientes artefactos dentro del expediente del proyecto, los cuales fueron estudiados:

- Documento de especificación de requisitos.
- Especificación de los casos de uso.
- Glosario de términos.

PP1.2 Roles y Responsabilidades

En el capítulo anterior se definieron los 4 roles que serán los encargados de realizar las pruebas de unidad y la relación con las principales actividades a realizar. A continuación se muestran los roles con todas las responsabilidades a desarrollar para la realización de las pruebas de unidad, en este caso al Subsistema Mecanismo de Control de Acceso y Autenticación.

Recursos Humanos	
Trabajadores	**Responsabilidades Específicas**
Jefe de Pruebas	• Analizar la documentación técnica. • Planificar las pruebas, decidir los objetivos de prueba apropiados. • Realizar el Plan de Pruebas.
Diseñador de Pruebas	• Identificar las técnicas apropiadas, herramientas y pautas para llevar a cabo las pruebas. • Definir la configuración del entorno para realizar las pruebas. • Identificar, priorizar, seleccionar y describir los Casos de Prueba.
Ingeniero de pruebas	• Instruir al equipo de prueba sobre el ambiente de la aplicación. • Implementar las pruebas automáticas. • Corregir los errores.

Probador	• Preparar y ejecutar las Pruebas. • Verificar y llevar el control de la ejecución de las pruebas. • Evaluar los resultados de las pruebas. • Conformar documentación al culminar el proceso.

Tabla 14: Descripción de la actividad Roles y Responsabilidades.

MATRIZ DE RESPONSABILIDAD				
Perfil	**Jefe de Pruebas**	**Diseñador de Pruebas**	**Ingeniero de Pruebas**	**Probador**
Alcance de Pruebas	X			
Estimación de los recursos	X			
Definición del entorno de Pruebas		X		
Plan de Pruebas	X			
Estrategia de pruebas	X			
Especificación de casos de prueba		X	X	
Matriz de Trazabilidad		X	X	
Ejecución de Pruebas Manuales		X	X	
Automatización de Pruebas			X	
Corregir los errores			X	
Evaluar los resultados de la prueba.				X
Ejecutar las pruebas	X			X

Tabla 15: Matriz de Responsabilidades a lo largo del procedimiento de pruebas.

PP1.3 Recursos Necesarios

En la aplicación del procedimiento para realizar pruebas de unidad se identifica un entorno con las siguientes características:

Recursos del Sistema	
Recursos	**Descripción**
Servidores	
Servidor de Base de Datos.	- Características del software: • Oracle Standard Edition ONE versión 11g. • Sistema Operativo: LINUX. - Características de hardware: • 2 GB de RAM 80 GB de capacidad en disco duro.
PC Cliente para pruebas.	- Características de hardware: • 1 GB de RAM y 160 GB de capacidad en disco duro. - Características de software: • Sistema Operativo Windows XP. Servi Pack 2 o Linux. • Kaspersky Workstation 6.0 como antivirus. • Instalado Paquete de Office 2003 o 2007. • Instalado symfony-1.2.8.
Red o subred.	• Las computadoras se conectan entre sí por cable Par Trenzado, de tipo UTP de categoría 5. • Conectados a puntos de red y estos a un Switch que existe en cada laboratorio.

Tabla 16: Tabla de los Recursos Necesarios.

En esta fase se realiza la primera versión del Plan de Pruebas, donde se recogen las actividades descritas anteriormente, así como la estrategia que se definió en el capítulo anterior.

DP2. Aplicación Fase de Diseño

En la fase de diseño se elaboraron casos de prueba para las distintas funcionalidades de 3 de los módulos del sistema Mecanismo de Control de Acceso y Autenticación.

2.1.1 Descripción de la Plantilla Casos de Prueba de Unidad

En este paso se prosigue a llenar los datos solicitados en la Plantilla Casos de Prueba de Unidad para el sistema Mecanismo de Control de Acceso y Autenticación.

Nombre del Proyecto: Sistema Único de Aduanas.

Caso de prueba 1:

Nombre Módulo: suAdminDomain.

Versión: 1.0.

Tipo de prueba: Prueba de Unidad.

Funcionalidades a Probar	Descripción
modificarDominio	Modifica o Actualiza un dominio en la BD.
obtenerDominios	Retorna un arreglo de Dominios.
buscarDominioPorNombre	Verifica que se encuentre en la BD el dominio del nombre especificado.
eliminarDominio	Deshabilita el Dominio en la Base de Datos.
insertarDominio	Inserta un dominio en la Base de Datos.
existeDominio	Verifica que exista el dominio especificado.

Casos de Prueba de Unidad					
Código del Caso de Prueba: CPU1					
Iteración: 1era					
Descripción de la prueba: Se realizan pruebas de unidad a las principales funcionalidades del módulo suAdminDomain.					
Nombre del encargado: Elizabeth Quintas Sánchez					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado Esperado del método**	**Resultado Esperado de la prueba**	**Resultado Real de la prueba**
SuAdminDomain	can_ok	'toJson'	'toJson'	ok	ok
modificarDominio	is	1, ''	false	not ok	ok
obtenerDominios	isa_ok		'array'	ok	ok
buscarDominioPorNombre	isa_ok	'Dominio 1'	'SuAdminGrupo'	ok	ok
buscarDominioPorNombre	cmp_ok	'Dominio 1'	'Dominio 1'	ok	ok
buscarDominioPorNombre	is	'Dominio 1'	true	ok	ok

buscarDominioPorNombre	isa_ok	'Dominio 2'	'TIMESTAMP'	ok	not ok
eliminarDominio	is	1	true	ok	ok
insertarDominio	is	'Dominio 10'	true	ok	Ok
existeDominio	is	'Dominio 3'	true	not ok	not ok

Caso de prueba 2:

Nombre Módulo: suAdminRole.

Versión: 1.0.

Tipo de prueba: Prueba de Unidad.

Funcionalidades a Probar	**Descripción**
existeGrupo	Devuelve true si encuentra un grupo con el nombre especificado.
modificarGrupo	Modifica el nombre del grupo dado el id de éste.
obtenerGrupos	Retorna un arreglo de grupos.
buscarGrupoPorNombre	Buscar el grupo con el nombre especificado.
eliminarGrupo	Deshabilita el Grupo en la base de datos.
insertarGrupo	Inserta un Grupo en la base de datos.

Casos de Prueba de Unidad					
Código del Caso de Prueba: CPU2					
Iteración: 1era					
Descripción de la prueba: Se realizan pruebas de unidad a las principales funcionalidades del módulo suAdminRole					
Nombre del encargado: Elizabeth Quintas Sánchez					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado Esperado del método**	**Resultado Esperado de la prueba**	**Resultado Real de la prueba**
modificarGrupo	is	1, ''	false	not ok	ok
obtenerGrupos	isa_ok		'array'	ok	ok
buscarGrupoPorNombre	isa_ok	'Grupo 1'	'SuAdminGrupo'	ok	ok

buscarGrupoPorNombre	isa_ok	'Grupo 1'-> getCreatedAt()	'TIMESTAMP'	ok	not ok
buscarGrupoPorNombre	cmp_ok	'Grupo 1'	'Grupo 1'	ok	ok
eliminarGrupo	is	1	true	ok	ok
insertarGrupo	is	'Grupo 10'	true	ok	ok
SuAdminGrupo	isa_ok	Un Objeto: SuAdminGrupo()	'SuAdminGrupo'	ok	ok
SuAdminGrupo	can_ok	Un Objeto: SuAdminGrupo()	'save'	ok	ok

Caso de prueba 3:

Nombre Módulo: suAdminUser.

Versión: 1.0.

Tipo de prueba: Prueba de Unidad.

Funcionalidades a Probar	**Descripción**
existeUsuario	Devuelve true si encuentra un usuario con el nombre especificado.
modificarUsuario	Modifica el nombre del usuario dado el id de éste.
obtenerUsuarios	Retorna un arreglo de usuarios.
cambiarContrasena	Cambiar contraseña por una nueva.
eliminarUsuario	Deshabilita el Usuario en la base de datos.
insertarUsuario	Inserta un Usuario en la base de datos.

Casos de Prueba de Unidad					
Código del Caso de Prueba: CPU3					
Iteración: 1era					
Descripción de la prueba: Se realizan pruebas de unidad a las principales funcionalidades del módulo suAdminUser.					
Nombre del encargado: Elizabeth Quintas Sánchez					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado Esperado del**	**Resultado Esperado de**	**Resultado Real de la**

			método	la prueba	prueba
SuAdminUsuario	isa_ok	Un Objeto: SuAdminGrupo()	'suAdminUsuario'	ok	ok
SuAdminUsuario	can_ok	Un Objeto: SuAdminGrupo()	'save'	ok	ok
modificarUsuario	is	1, ''	false	not ok	ok
obtenerUsuarios	isa_ok		'array'	ok	ok
cambiarContrasena	is	'Usuario 1', 'elizabeth ', 'tesis', 'tesis'	true	ok	ok
cambiarContrasena	is	'Usuario 3','tesista', 'lo esencial es invisible a los ojos', 'lo esencial es invisible a los ojos'	false	not ok	ok
eliminarUsuario	is	1	true	ok	ok
insertarUsuario	is	'Usuario 10'	true	ok	ok

2.1.2 Implementar pruebas automáticas

El Ingeniero de Componentes es el encargado de implementar las pruebas de unidad de forma automática, con la ayuda de los casos de prueba descritos en el epígrafe anterior.

A continuación se muestran imágenes de las pruebas automáticas que se le realizaron a 3 de los módulos del subsistema Mecanismo de Control de Acceso y Autenticación.

```php
<?php
include(dirname(__FILE__).'/../bootstrap/unit.php');
new sfDatabaseManager(ProjectConfiguration::getApplicationConfiguration('aplicacion', 'test', true));
$t = new lime_test(9, new lime_output_color());

$t->comment('----- Haciendo pruebas unitarias al modulo suAdminDomain ------');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad modificarDominio ------');
$t->is(SuAdminDominioPeer::modificarDominio(1, ''), false, 'Verifica que se actualizo el dominio');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad obtenerDominios ------');
$t->isa_ok(SuAdminDominioPeer::obtenerDominios(), 'array', 'Retorna un arreglo de dominios');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad buscarDominioPorNombre ------');
$t->isa_ok(SuAdminDominioPeer::buscarDominioPorNombre('Dominio 1'), 'SuAdminDominio', 'Retorna un objeto');
$t->cmp_ok(SuAdminDominioPeer::buscarDominioPorNombre('Dominio 1')->getNombreDominio(),'===', 'Dominio 1',
        'El grupo que retorna el metodo buscarDominioPorNombre("Dominio 1") y "Dominio 1" son exactamente
         lo mismo');
$t->is(SuAdminDominioPeer::buscarDominioPorNombre('Dominio 1'), 'true', 'Verifica que se encuentre
         Dominio 1 en la BD');
$t->isa_ok(SuAdminDominioPeer::buscarDominioPorNombre('Dominio 2')->getCreatedAt(),'TIMESTAMP', 'El metodo
        buscarDominioPorNombre("Dominio 2") retorna la fecha creada del Dominio 2');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad eliminarDominio ------');
$t->is(SuAdminDominioPeer::eliminarDominio(1), true, 'Verifica que se elimino el dominio con id 1');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad existeDominio ------');
$t->is(SuAdminDominioPeer::insertarDominio('Dominio 10'), true, 'Verifica que se inserto Dominio 10 en la BD');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad insertarDominio ------');
$t->is(SuAdminDominioPeer::existeDominio('Dominio 3'), true, 'Verifica que el Dominio 3 existe
     en la BD');

$t->diag('----- Haciendo pruebas unitarias SuAdminDomain ------');
$t->isa_ok(new SuAdminDominio(), 'SuAdminDominio', 'Se crea un objeto de la clase SuAdminDominio');
$t->can_ok(new SuAdminDominio(), 'toJson', 'El objeto de tipo suAdminDominio tiene el metodo toJson()');
?>
```

Figura 11: Vista de las pruebas realizadas a métodos del módulo suAdminDomain.

```php
<?php
include(dirname(__FILE__).'/../bootstrap/unit.php');
new sfDatabaseManager(ProjectConfiguration::getApplicationConfiguration('aplicacion', 'test', true));
$t = new lime_test(7, new lime_output_color());

$t->comment('----- Haciendo pruebas unitarias al modulo suAdminRole ------');

$t->diag('----- Haciendo pruebas unitarias a SuAdminGrupo ------');
$t->isa_ok(new SuAdminGrupo(), 'SuAdminGrupo', 'Se crea un objeto de la clase SuAdminGrupo');
$t->can_ok(new SuAdminGrupo(), 'save', 'El objeto de tipo suAdminGrupo tiene el metodo save()');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad buscarGrupoPorNombre ------');
$t->isa_ok(SuAdminGrupoPeer::buscarGrupoPorNombre('Grupo 1'), 'SuAdminGrupo', 'Retorna un objeto');
$t->isa_ok(SuAdminGrupoPeer::buscarGrupoPorNombre('Grupo 1')->getCreatedAt(),'TIMESTAMP', 'El metodo
        buscarGrupoPorNombre("Grupo 1" ) retorna la fecha creada del Grupo 1');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad modificarGrupo ------');
$t->is(SuAdminGrupoPeer::modificarGrupo(1, ''), false, 'Verifica que se actualizo el grupo');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad existeGrupo ------');
$t->is(SuAdminGrupoPeer::existeGrupo('Grupo 1'), true, 'Verifica que el Grupo 1 este
     en la BD');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad insertarDominio ------');
$t->is(SuAdminGrupoPeer::insertarGrupo('Grupo 10'), true, 'Verifica que se inserto el grupo
    con el nombre Grupo 10 en la BD');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad eliminarDominio ------');
$t->is(SuAdminGrupoPeer::eliminarGrupo(1), true, 'Verifica que se elimino el grupo con id 1
    en la BD');
?>
```

Figura 12: Vista de las pruebas realizadas a métodos del módulo suAdminRole.

```php
<?php
include(dirname(__FILE__).'/../bootstrap/unit.php');
new sfDatabaseManager(ProjectConfiguration::getApplicationConfiguration('aplicacion', 'test', true));
$t = new lime_test(7, new lime_output_color());
$t->comment('----- Haciendo pruebas unitarias al modulo suAdminUser ------');

$t->diag('----- Haciendo pruebas unitarias al moduloSuAdminGrupo ------');
$t->isa_ok(new SuAdminUsuario(),'SuAdminUsuario','Se crea un objeto de la clase SuAdminUser');
$t->can_ok(new SuAdminUsuario(), 'toJson', 'El objeto de tipo SuAdminUsuario tiene el metodo toJson()');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad modificarUsuario ------');
$t->is(SuAdminUsuarioPeer::modificarUsuario(2,'' ), false, 'Verifica que no se debe actualizar el Usuario
    con una cadena de texto vacia');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad existeUsuario ------');
$t->is(SuAdminUsuarioPeer::existeUsuario('Usuario 1'), true, 'Verifica que el Usuario 1 existe
     en la BD');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad obtenerUsuarios ------');
$t->isa_ok(SuAdminUsuarioPeer::obtenerUsuario('Usuario 1'), 'array', 'Retorna como tipo de dat un arreglo
    de Usuarios');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad cambiarContrasena ------');
$t->is(SuAdminUsuarioPeer::cambiarContrasena('Usuario 1','elizabeth ', 'tesis', 'tesis'), true, 'Verifica
    que se cambió la contraseña');
$t->is(SuAdminUsuarioPeer::cambiarContrasena('Usuario 3','tesista ', 'lo esencial es invisible a los ojos',
        'lo esencial es invisible a los ojos'), false, 'Verifica que no se cambie la contrasena por
         ser muy larga');

$t->diag('----- Haciendo pruebas unitarias a la funcionalidad eliminarUsuario ------');
$t->is(SuAdminUsuarioPeer::eliminarUsuario(1), true, 'Verifica que se eliminó el usuario con id 1
    en la BD');
?>
```

Figura 13: Vista de las pruebas realizadas a métodos del módulo suAdminUser.

EP3. Aplicación Fase Ejecución de las pruebas

3.1.1 Ejecución de las pruebas

Se procede a la ejecución de las pruebas, una vez que esté todo listo y bien definido y el probador esté capacitado para proceder con las mismas.

En esta fase se realiza la actividad Evaluación de los Resultados, la cual se detalla a continuación.

3.1.2 Evaluación de los resultados

Las pruebas de unidad fueron implementadas de forma automática, ejecutándose utilizando una PC cliente con las características antes mencionadas. El sistema exigía una conexión a una Base de Datos que se encontraba en un servidor.

Se le realizaron pruebas a la mayoría de las funcionalidades de los módulos escogidos. Los casos de prueba que se habían definido anteriormente en la Plantilla de Casos de Prueba de Unidad fueron ejecutados satisfactoriamente siguiendo la estrategia que se había planteado. Los distintos errores encontrados fueron clasificados según su prioridad y a través del mecanismo correspondiente fueron informados a los responsables para su corrección.

En las siguientes tablas se muestran los resultados de la realización de las pruebas, exponiéndose los errores encontrados en los casos de prueba a través de reportes.

Reporte errores de pruebas realizadas al módulo suAdminDomain:

Reporte de Prueba Nro. 1			
Errores Encontrados: 2			
Id Caso de Prueba: CPU1			
Funcionalidad Probada	**Error Encontrado**	**Clasificación del error**	**Observación**
modificarDominio	La funcionalidad permite modificar el dominio cambiándole el nombre por una cadena de texto vacía.	Media	El nombre del dominio no puede ser una cadena de texto vacía.
buscarDominioPorNombre	El tipo de dato que devuelve esta funcionalidad es string, cuando debería ser de tipo 'TIMESTAMP'.	Baja	Para probar esta funcionalidad se realizó llamando al método getCreateAt. 'TIMESTAMP' es del tipo Fecha.

Reporte errores de pruebas realizadas al módulo suAdminRole:

Reporte de Prueba Nro. 2			
Errores Encontrados: 2			
Id Caso de Prueba: CPU1			
Funcionalidad Probada	**Error Encontrado**	**Clasificación del error**	**Observación**
modificarGrupo	La funcionalidad permite modificar el grupo cambiándole el nombre por una cadena de texto vacía.	Media	El nombre del grupo no puede ser una cadena de texto vacía.
buscarGrupoPorNombre	El tipo de dato que devuelve esta funcionalidad llamando al método getCreateAt, es string, cuando debería ser de tipo 'TIMESTAMP'.	Baja	'TIMESTAMP' es del tipo Fecha.

Reporte errores de pruebas realizadas al módulo suAdminUser:

Reporte de Prueba Nro. 3			
Errores Encontrados: 2			
Id Caso de Prueba: CPU3			
Funcionalidad Probada	**Error Encontrado**	**Clasificación del error**	**Observación**
modificarUsuario	La funcionalidad permite modificar al usuario cambiándole el nombre por una cadena de texto vacía.	Media	El nombre del usuario no puede ser una cadena de texto vacía.
cambiarContrasena	La funcionalidad permite cambiar la contraseña por una cadena muy larga.	Media	La contraseña no debe de ser de exceder de 30 caracteres.
cambiarContrasena	La funcionalidad permite cambiar la contraseña por una cadena muy corta.	Media	La contraseña no debe ser menor de 8 caracteres.

Como se puede apreciar en las tablas anteriores, la mayoría de los errores encontrados fueron:

1. Problemas en las funcionalidades modificar (modificarDominio, modificarGrupo y modificarUsuario) de los 3 módulos que se probaron, porque permite modificar el nombre por una cadena de texto vacía.

2. Problemas en las funcionalidades buscarDominioPorNombre y buscarGrupoPorNombre, pues donde se esperaba que devolviera un tipo de dato fecha mientras lo que realmente devuelve es un string. Este error es difícil de detectar en otros tipos de pruebas, sin embargo, con la implementación de esta se pudo detectar fácilmente.

3. Problemas con la funcionalidad cambiarContrasena, pues permite cambiar la contraseña por una cadena de caracteres muy larga y también por una demasiado corta.

Se debe añadir que a estos módulos se le realizaron varios tipos de pruebas antes de aplicarle el procedimiento y a pesar de esto luego de haber ejecutado las 26 pruebas que fueron implementadas en este capítulo, se encontraron 6 funcionalidades con problemas, lo cual representa el 23% del total de las pruebas.

Conclusiones

Luego de la aplicación del procedimiento para la realización de pruebas de unidad al Sistema Único de Aduanas y dentro de éste al subsistema Mecanismo de Control de Acceso y Autenticación, se llegó a la conclusión de la necesidad del establecimiento de un procedimiento para la aplicación de las mismas, que permite una organización del trabajo además de la obtención de resultados satisfactorios para la mejora de la calidad.

En este capítulo se aplicó el procedimiento desarrollado en el capítulo 2, con el objetivo de demostrar su validez y de que sea utilizado para realizar pruebas de unidad en el proyecto Sistema Único de Aduanas; partiendo de que no existe ningún antecedente de procedimientos para la realización de pruebas de unidad, ni la existencia de plantillas como la que se propone en este trabajo para la documentación de los casos de prueba de unidad, incluyendo los resultados obtenidos.

CONCLUSIONES GENERALES

Durante el desarrollo de este trabajo se le dio cumplimiento a los objetivos propuestos al inicio del mismo. Se desarrolló y aplicó un procedimiento para pruebas de unidad de acuerdo con las características del Sistema Único de Aduanas, lo que permite su aplicación en cualquiera de los subsistemas que forman parte del mismo, demostrando lo antes dicho con la aplicación exitosa del procedimiento para pruebas de unidad en el subsistema Mecanismo de Control de Acceso y Autenticación. Además se elaboró una Plantilla de Casos de Prueba de Unidad, creando casos de prueba que cubrieron la mayoría de las funcionalidades de los módulos probados.

Se evaluaron los resultados de las pruebas comparando los resultados esperados y los reales, generando reportes de los errores encontrados. Se logró desarrollar una adecuada documentación de todo el procedimiento de pruebas utilizado, el cual servirá de apoyo para futuros subsistemas que se quieran probar.

Partiendo de la generación de casos de prueba, la propuesta de solución que se presenta brinda la posibilidad de documentar las pruebas unitarias realizadas al software.

El subsistema Mecanismo de Control de Acceso y Autenticación tendrá mayor calidad una vez que se revise la documentación resultante de la aplicación del procedimiento y se realicen los cambios pertinentes para eliminar los errores encontrados.

El procedimiento de pruebas unitarias se organizó de forma apropiada planificando los recursos disponibles, roles y las distintas fases en las que fue dividido el proceso. Con la elaboración del Plan de Pruebas se obtuvo una mayor organización y planificación de los recursos utilizados en el procedimiento propuesto. La estrategia que se trazó sirvió para describir todo un flujo de trabajo que es implementado durante el período de realización de las pruebas.

RECOMENDACIONES

Los objetivos generales de este trabajo fueron alcanzados, pero durante su desarrollo han surgido ideas que sería recomendable tener en cuenta para su futuro perfeccionamiento:

- Utilizar en otros proyectos con características similares al Sistema Único de Aduanas el nuevo procedimiento propuesto para las pruebas de unidad.
- Brindar cursos sobre pruebas unitarias a estudiantes y profesores de la Universidad de las Ciencias Informáticas, para que así puedan aplicar mejor el procedimiento.
- Continuar documentando el proceso de automatización de las pruebas unitarias.
- El estudio de los diferentes tipos de pruebas de unidad que existen y que no son desarrolladas en este procedimiento para su posterior aplicación.
- Continuar la investigación sobre el resultado obtenido y proponerlo como base referencial o material auxiliar en el desarrollo de futuros trabajos.
- Se recomienda que el resultado propuesto en la investigación sea implantado como uso cotidiano y sea introducido en todos los sistemas de gestión de la universidad.

REFERENCIAS

1. [cited; Available from: http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm
2. Pressman, R.S., *Un enfoque práctico*. 2005.
3. Daniel Chudnovsky, A.L., Silvana Melitsko, *El sector de software y servicios informáticos en la Argentina: Situación actual y perspectivas de desarrollo*. 2001.
4. Date, C.J., *Introducción a los sistemas de bases de datos*. 2001, SÉPTIMA EDICIÓN.
5. Estrada, A.F., *Calidad de software*. 2006, Universidad de Matanzas "Camilo Cienfuegos".
6. Ballesteros, J.M.M.M.J.F.H., *La calidad del software y su medida*. Editorial Centro de estudio Ramón.
7. Lovelle, J.M.C., ed. *Calidad del Software*. 1999, Universidad de Oviedo: España.
8. Myers, *The art of software testing*. John Wiley & Sons ed. 1979.
9. López, J., *Oracle. Fundamentos para el desarrollo de aplicaciones Web*. MP Edisiones S.A ed, Buenos Aires-Argentina.
10. Jacobson, I., *Proceso unificado del desarrollo del software*. 2003.
11. Natalia Juristo, A.M.M., Sira Vegas, *Técnicas de evaluación del software*. 2006: 12.0.
12. UCI, D.C.I.d.S. *Flujo de trabajo de pruebas*. Asignatura: Ingeniería de Software II Curso: 2008-2009.
13. Booch, G.R., J. y Jacobson, I. , *El Lenguaje Unificado de Modelado*, ed. Addison-Wesley, 2000.
14. Computación, D.d.S.I.y. *Proceso de desarrollo de software*. [cited; Available from: http://www.dsic.upv.es/asignaturas/facultad/lsi/doc/IntroduccionProcesoSW.doc.
15. Ivar Jacobson, G.B., James Rumbaugh,, *El proceso unificado de desarrollo del software*. 2000, Madrid España: Rational Software Corporation
16. Eneko, A., *El método Delphi*, ed. F.d.C.E.y. Empresariales.
17. Callón, J. *Pruebas unitarias en integración contínua*. [cited; Available from: http://javier.callon.org/pruebas-unitarias-en-integracion-continua.
18. Rodríguez, E. *Útiles herramientas para desarrolladores de PHP*. 2009 [cited; Available from: http://www.tecnovi.net/utiles-herramientas-para-desarrolladores-de-php/.
19. M, I.A. *Mejora de procesos*. 2007 [cited; Available from: http://www.sg.com.mx/sg07/presentaciones/Mejora%20de%20procesos/SG07.P02.Scrum.pdf.

REFERENCIAS

20. Jiménez, Y.P., *Procedimiento para el Aseguramiento de la Calidad en el Proyecto INSIGNE*. 2008, Ciudad de la Habana.

21. *PX Programacion Extrema* 22 de 12 de 2007.

22. Humprhrey, W.S., *Introduction to the team software process*.

23. Fabien Potencier, F.Z., *Symfony la guía definitiva*. 2008.

24. Potencier, F., *El tutorial Jobeet.* 2009.

BIBLIOGRAFÍA

ANSI/IEEE Standard 829-1983 for Software Test Documentation. (1983). IEEE Press.

Aplicada, ISCA- Ingeniería de Software y Calidad. [http://www.isca.com.ve/productos.htm]. (s.f.). *Productos IBM Rational* .

B. Boehm, C. A. (2000). *Software Cost Estimation with COCOMO II.* Prentice-Hall.

Binder, R. (2000). *Testing Object-Oriented Systems: Models, Patterns and Tools.* Addison-Wesley.

Carr, C. (1996). *Team Leader's Problem Solver.* Prentice-Hall .

Cay Horstmann, G. C. (1999). *Core Java 2. Volume 1: Fundamentals.* Prentice-Hall.

Collofello, J. S. (1994). *Assessing the software process maturity of software engineering courses.*

Cortés, O. H. (2004). *Aplicación práctica del diseño de pruebas de software a nivel de programación.*

David Sykes, J. M. (2001). *A Practical Guide to Testing Object-Oriented Software.* Addison-Wesley.

Erich Gamma, R. H. (1995). *Elements of Reusable Object-Oriented SoftwareAddison-Wesley.*

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.

Green, R. (17 de diciembre 1999). Lista de características de Java que tienden a conducir a errores de programación. *Java Gotchas* .

http://www.adrformacion.com/cursos/calidad/calidad.html . (revisado, 25/4/2010).

http://www.giro.infor.uva.es/Publications/2004/MLC04/JENUI2004.pdf. (revisado, 17/Febrero/2010.).

http://www.ldc.usb.ve/~teruel/ci4713/clases2001/planPruebas.html. (revisado, 16/3/2010.).

Humphrey, W. (2000). *Introduction to the Team Software ProcessSM.* Addison-Wesley.

Humphrey, W. (1989). *Managing the Software Process.* Addison-Wesley .

Humphrey, W. S. (2001). *Introducción al proceso de software personal. .* Addison Wesley.

Ian, S. *Ingeniería de Software.* Séptima Edición, Pearson Education .

ISO 9000:2000 [2000]. Sistema de Gestión de la calidad. Principios Fundamentales y Vocabulario. Ginebra, Suiza: Secretaria General ISO, Traducción certificada.

Ivar Jacobson, G. B. (2000). *El proceso unificado de desarrollo de software.* Addison Wesley.

Jornada sobre Testeo de Software. [http://web.iti.upv.es/~squac/JTS/JTS2005/contenido.html] . (2005).

Lemus, Y. P. (Ciudad de la Habana : s.n., 2007). *SIMETSE – Sistema de Metricas para evaluar el Software Educativo.*

León, R. A. (2002). *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA .* Ciudad de la Habana: Editorial Universitaria.

Lovelle, J. M. (1999). *Calidad del Software Calidad del Software Grupo GIDIS.* Universidad Nacional de la Pampa: www.uniovi.es.

M, F. C. (1995). *Un enfoque actual sobre la calidad del software.*

Marick, B. (1995). *The Craft of Software Testing: Subsystem testing including object-based and object-oriented testing.* Prentice-Hall.

Martínez, N. M. (2005). En busca de respuestas para la ingeniería de software. [http://is.ls.fi.upm.es/udis/docencia/proyecto/docs/FilosofiaIS.pdf] .

Meyer, B. *Fundamentos de Ingeniería de Software.*

Mónica Villavicencio, J. M. (2005). *Aspectos de la Calidad y Dificultades en la Gestión de Proyectos de Software: "Estudio exploratorio".*

Montesdeoca, C. (2005). *Tesis Profesional, http://www.pue.udlap.mx/~tesis/lis/pelaez_r_jj/capitulo3.pdf.* Universidad de las Américas, Puebla.

Morgan, M. *Descubre Java 1.2.* Prentice Hall.

Myers. (1979). *The art of software testing.*

Pérez, Y. J. (2008). *Procedimiento para el Aseguramiento de la Calidad en el Proyecto INSIGNE.* Ciudad de Habana : s.n., .

Perry, W. (2000). *Effective Methods for Software Testing.* Segunda edición, Wiley.

Pressman, R. S. (2002). *Ingeniería del Software: Un enfoque práctico .* Madrid: Quinta edición. McGraw-Hill.

Pressman, R. S. (2001). *Software Engineering: A Practitioner's Approach.* Fifth edition. McGraw-Hill.

Pruebas de Sotware. [http://lsi.ugr.es/~ig1/docis/pruso.pdf]. (s.f.).

PX Programacion Extrema [en línea], http://www.programacionextrema.org/. (12/4/2010).

Rodríguez, D. E. (Septiembre, 2003). *Introducción a la Calidad del Software .*

Rodríguez, E. C. (2008). *Estrategia para el Aseguramiento de la Calidad del Proyecto Juegos CNeuro.* Ciudad de La Habana : s.n.

S., P. R. (1998). *Ingeniería del software: Un enfoque práctico.* 4ª Edición. McGrawHill.

Santiesteban, Y. C. (2008). *Propuesta de Plan de Aseguramiento de la Calidad del Proyecto Servicios Comunitarios.* Ciudad de la Habana : s.n.

Técnicas de Verificación y Pruebas. [http://trevinca.ei.uvigo.es/~rlaza/teoria/Verificacion.pdf]. (s.f.).

Tomayko, J. E. (1987). *Teaching a project-intensive introduction to software engineering.* Software Engineering Institute.

Villalobos Hernández María de la luz, A. F. (Agosto, 2001.). *Investigación sobre las prácticas de ingeniería de software .* México.

GLOSARIO DE TÉRMINOS

UCI: Universidad de las Ciencias Informáticas

SUA: Sistema Único de Aduanas.

Framework: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

IDE: Entorno de Desarrollo Integrado.

Artefacto: Productos tangibles del proyecto que son producidos, modificados y usados. Pueden ser modelos, elementos dentro del modelo, documentos, gráficos, código fuente y ejecutables.

Casos de prueba: Conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito. También se puede referir a la documentación en la que se describen las entradas, condiciones y salidas de un caso de prueba.

Fase: Son los pasos en que se descomponen las metodologías. Cada fase puede o no estar subordinada a otra fase, pudiendo existir entre ellas relaciones de dependencia.

Falla: Error en el funcionamiento que emite el sistema.

Hardware: Conjunto de componentes físicos de una computadora. Refiérase a objetos tangibles y palpables como son los discos, lectores de discos, monitores, teclados, las impresoras, tarjetas y chips.

Prueba: Prueba de software. Ejecución de un sistema bajo condiciones específicas, se observan y se analizan los resultados realizándose una evaluación de los mismos.

Procedimiento: Forma específica de llevar a cabo una actividad. En muchos casos los procedimientos se expresan en documentos que contienen el objeto y el campo de aplicación de una actividad; que debe hacerse y quien debe hacerlo; cuando, donde y como se debe llevar a cabo; que materiales, equipos y documentos deben utilizarse; y como debe controlarse y registrarse.

Producto: Es cualquier cosa que puede ser ofrecida al mercado para su compra, para su utilización o para su consideración. Es cualquier bien, servicio o idea capaz de motivar y satisfacer a un comprador.

Proyecto: Elemento organizativo a través del cual se gestiona el desarrollo del software, el resultado de un proyecto es una versión del producto.

ANEXOS

Anexo 1: Plantilla Plan de Pruebas

1. Introducción

Este documento se confecciona con el objetivo de definir el Plan de Pruebas a la aplicación [nombre de la aplicación].

2. Organización del Equipo de Pruebas

[Descripción del equipo de probadores, por quienes está compuesto, responsabilidad de cada miembro y ponerlos en el siguiente formato].

Recursos Humanos	
Trabajadores	**Responsabilidades Específicas**
[Nombre del trabajador].	[Responsabilidades]

3. Especificaciones del Software y Hardware

[Corresponde a una lista individualizada de todo el hardware y el software que utiliza la aplicación, distribuyéndolos en el siguiente formato].

Recursos del Sistema	
Recursos	**Descripción**
Recurso 1	[Características del software] [Características de hardware]
Recurso 2	[Características de hardware] [Características de hardware]

4. Estrategia de Prueba

De la estrategia de las pruebas debe detallarse lo siguiente:

4.1 Objetivo

[El objetivo global de esta estrategia debe alcanzarse].

4.2 Técnica

[Especifica como los casos de prueba serán desarrollados, se deben describir los pasos a realizar los casos de prueba. El instrumento o herramienta usado para almacenarlos y donde pueden ser encontrados; como ellos serán ejecutados y los datos que serán usados].

Anexo 2: Plantilla de Casos de Prueba de Unidad

Nombre del Proyecto:

Caso de prueba 1:

Nombre Módulo:

Versión:

Tipo de prueba:

[En esta tabla se recogen las funcionalidades del módulo que se va a probar].

Funcionalidades a Probar	Descripción

[En esta tabla se recogen los Casos de Prueba de Unidad diseñados para el módulo a probar].

Casos de Prueba de Unidad					
Código del Caso de Prueba:					
Iteración:					
Descripción de la prueba:					
Nombre del encargado:					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado Esperado del método**	**Resultado Esperado de la prueba**	**Resultado Real de la prueba**

[Reportes de los Errores encontrados en las pruebas].

Reporte de Prueba Nro.			
Errores Encontrados:			
Id Caso de Prueba:			
Funcionalidad Probada	**Error Encontrado**	**Clasificación del error**	**Observación**