Universidad de las Ciencias Informáticas

Facultad 7

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: Propuesta de procedimiento general para normalizar las pruebas en el desarrollo de Aplicaciones Web en la Facultad 7.

Autor: *Edel Ávila Cruz*

Tutor: *Ing. Jacqueline Marín Sánchez*

Ciudad de La Habana, Mayo de 2008.

"Año 50 de la Revolución"

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la facultad 7 de La Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los 31 días del mes de Mayo del año 2008.

Autor: Edel Ávila Cruz

Tutor: Ing. Jacqueline Marín Sánchez

_____ _____

Datos de Contacto

Jacqueline Marín Sánchez (jmarin@uci.cu): Profesor graduado de Ingeniería en Ciencias Informáticas en el año 2007 en la Universidad de las Ciencias Informáticas. Imparte la asignatura de Ingeniería de Software en la facultad 7. Es asesor de Calidad y Jefe del Área Temática de Calidad de la facultad 7. Posee la categoría docente de Adiestrado.

Agradecimientos

"El agradecimiento es la memoria del corazón"

A La Revolución y a Fidel por ser el máximo creador de esta nuestra casa de altos estudios.

A mis padres, por el apoyo incondicional que me han dado durante toda mi vida.

A La Universidad y su colectivo de profesores que me han formado durante estos cinco años.

A todos mis compañeros, que de una forma u otra han contribuida en la realización de esta investigación.

Dedicatoria

A mi hijo por ser lo más grande que tengo en este mundo y la fuente de mi inspiración diaria de ser cada vez mejor y esa luz que me ilumina a seguir adelante.

A mis padres y mi esposa por tanta dedicación y apoyo en todo momento, especialmente a mi padre que fue el que me inspiró y siempre alentó a formarme en esta especialidad.

A todos mis compañeros y amigos que en todo este tiempo me han acompañado y apoyado en todas mis decisiones, en especial a mi a migo Maikel por estar siempre en las buenas y malas como ese amigo que todos necesitan.

Resumen

Esta investigación propone un procedimiento general para normalizar las pruebas en el desarrollo de aplicaciones Web implementadas en la facultad 7 de la Universidad de las Ciencias Informáticas. En ella se explica de forma general que pruebas realizar en cada fase del ciclo de vida y en que momento deben ser aplicadas, planifica las pruebas para las próximas fases, proponiendo además algunas herramientas y el personal que debe realizarlas.

Teniendo en cuenta que es un procedimiento a aplicar dentro de la Facultad 7, estas pruebas se enfocan en las aplicaciones Web con Arquitectura Orientada a Servicios y Basada en Componentes que son las más utilizadas, aunque puede ser aplicado a otro tipo.

Se hizo un análisis de las principales bibliografías especializadas en el tema, profundizando en los diferentes métodos de pruebas que existen, y modelos para la elaboración de procedimientos que permitan desarrollar pruebas durante todo el ciclo de vida de una aplicación Web. Además se analizó la situación existente actualmente en la facultad 7 en cuanto a la realización de pruebas y se demostró la falta de una metodología que guie el desarrollo de pruebas en las diferentes áreas temáticas.

El procedimiento garantiza un mejor desempeño en cada etapa del ciclo de vida en aplicaciones Web desarrolladas en la Facultad 7. Será aplicado en todas las áreas temáticas de la facultad, normalizando el proceso de desarrollo de pruebas y garantizando la satisfacción del cliente al recibir un producto más óptimo.

Índice.

Introducción

El constante desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) han llevado a la sociedad a insertarse en un mundo, donde el impacto de los Software (SW) ha revolucionado la forma de pensar y actuar de las personas con las que interactúan. La lucha de los grandes productores por ser cada día mejores, al ofrecer productos más baratos y asequibles a los usuarios, ha transformado en pocos años la forma de hacer y desarrollar las actividades; que antes se hacían con otros programas y que ahora se realizan de forma mas sencilla, rápida y menos costosa, mediante software de una alta calidad en el mercado mundial.

En Cuba, hoy se trabaja por lograr la inserción en el mundo del desarrollo del SW, y en particular por la informatización del país; en todos los sectores, con prioridad en los básicos para la sociedad, sin descartar la producción para el mercado mundial. El perfeccionamiento empresarial y el uso de las TIC ha incrementado en el país el empleo de aplicaciones Web en la mayoría de las empresas y La Universidad de las Ciencias Informáticas (UCI) es un pilar decisivo en el funcionamiento de esta gran maquinaria de producción de SW.

Actualmente, La Universidad de las Ciencias Informáticas presenta un creciente desarrollo de aplicaciones Web. En la Facultad 7 hoy se trabaja básicamente con Arquitectura Basada en Componentes y Orientada a Servicios (CBA-SOA) y ya se han obtenido algunos productos con estas características. Estos han sido revisados por el área de calidad y se han detectado dificultades como:

- No existe una metodología que guíe el proceso de aplicación de pruebas en cada una de las fases.
- Existe un diseño incoherente de las pruebas.
- No se realizan pruebas de integración,
- No se revisan los modelos de despliegue.

Esta situación provoca que el grupo de calidad de la Facultad 7 en ocasiones realice pruebas que deberían ser ejecutadas dentro del equipo de desarrollo y no en el laboratorio de calidad. Muchos de los proyectos no tienen una visión de la importancia que tiene el aseguramiento de la calidad desde que comienza la concepción del SW, ni emplean herramientas que permitan simular pruebas que perfeccionen con su uso la calidad del producto final. Éstos además, no llevan un control estricto de las pruebas que se realizan, ni de sus iteraciones, lo que dificulta la realización de las pruebas en el proceso de certificación del SW.

Partiendo de esta situación, se plantea como **problema a resolver**:

¿Cómo garantizar el éxito en el desarrollo de aplicaciones Web implementadas en la Facultad 7 durante todo su ciclo de vida?

Conociendo el problema, se define como **objeto de estudio** el proceso de aplicación de pruebas en el desarrollo de software, y **el campo de acción** se enmarca en las pruebas que se realizan durante el ciclo de vida de una aplicación Web en la Facultad 7.

Teniendo en cuenta lo antes expuesto el **objetivo general** de la investigación es: Definir un procedimiento general que proponga las pruebas a realizar en aplicaciones Web implementadas en la Facultad 7 durante todo su ciclo de vida.

La idea a defender seria: La propuesta de un procedimiento general que garantice un mejor desempeño en cada etapa del ciclo de vida en aplicaciones Web implementadas en la Facultad 7.

Para lograr el objetivo de esta investigación y dar solución a la problemática antes expuesta, es necesaria la realización de las siguientes **tareas:**

1. Analizar los antecedentes sobre procesos, metodologías y modelos existentes para la realización de pruebas.
2. Analizar los procedimientos existentes a nivel nacional e internacional para desarrollar pruebas a aplicaciones Web.
3. Valorar el uso de herramientas en la realización de pruebas en aplicaciones Web.
4. Aplicar entrevistas a los desarrolladores de los proyectos productivos donde se desarrollan aplicaciones Web en la Facultad 7.
5. Valorar los resultados de las entrevistas.
6. Definir la estrategia de pruebas para Aplicaciones Web.

Este trabajo tiene una estructura de tres capítulos:

El capítulo I ¨Fundamentación teórica¨, como su título indica, fundamenta teóricamente este trabajo de diploma en el que se estudia el estado actual de los diferentes procedimientos y procesos de pruebas de software utilizados a nivel mundial; se definen los conceptos empleados más importantes y las técnicas de pruebas y herramientas tratadas. En conjunto, este capítulo es una base teórico-práctica de los métodos más empleados en la actualidad a nivel mundial para abordar el problema de tesis.

En el capítulo II se hace una caracterización de la producción de aplicaciones Web en la Facultad 7, donde se demuestra la ausencia de procedimiento, y se exponen experiencias con productos

revisados por el grupo de calidad de esta Facultad. Se usan métodos de obtención de información como las encuestas y entrevistas que contribuyen al diagnostico de la producción de dicha Facultad.

Finalmente, en el capítulo III se hace la propuesta del procedimiento general para garantizar el éxito en las aplicaciones Web implementadas en la facultad 7. Este procedimiento abarca todo el ciclo de vida del producto y propone los tipos de pruebas y algunas herramientas para la aplicación de las mismas.

Capítulo1: Fundamentación Teórica

1.1 Introducción

El presente capítulo tiene como objetivo abordar los principales conceptos y aspectos más significativos relacionados con las principales temáticas abordadas en diferentes fuentes bibliográficas acerca del estado del arte que presenta el tema que se investiga. Hacer mención de los distintos tipos de pruebas que puedan ser aplicadas en productos Web, teniendo en cuenta el uso de herramientas existentes a nivel mundial. Presenta además un estudio de los diferentes procesos y estrategias de pruebas que se llevan a cabo en las grandes industrias del software. Así como el estudio de modelos que permiten la elaboración de un procedimiento de prueba que pueda ser aplicado durante todo el ciclo de vida de una aplicación Web.

1.2 Conceptos Básicos

Patrón

Un patrón es un modelo que se puede seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Los patrones capturan la experiencia existente y probada para promover buenas prácticas. (1)

- Ayudan a construir la experiencia colectiva de Ingeniería de Software.
- Son una abstracción de "problema – solución".
- Se ocupan de problemas recurrentes.
- Identifican y especifican abstracciones de niveles más altos que componentes o clases individuales.
- Proporcionan vocabulario y entendimiento común.

Patrón arquitectónico

Un patrón arquitectónico determinará cómo se incorporará este patrón de usabilidad en una arquitectura software; es decir, qué efecto tendrá la inclusión del patrón de usabilidad en los componentes de la arquitectura del sistema. Al igual que los patrones de diseño, los patrones arquitectónicos reflejarán una posible solución a un problema: la incorporación de un patrón de usabilidad concreto en un diseño software. (2)

Componente

En este trabajo se utiliza la definición de Sterling Software, que plantea que un componente es: (3)

- Una parte reutilizable que encapsula elementos del modelo (por ejemplo una DLL, documentos, tablas, biblioteca, etc.).
- Un paquete de software el cual ofrece servicios a través de sus interfaces.
- Un paquete de Software que pude ser usado para construir aplicaciones o componentes más grandes.(Software 2000)

La figura 1.1 muestra como se representa un componente en la ingeniería de software.

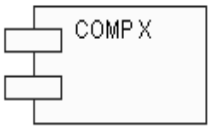

Figura 1.1 Representación de Componente

Estereotipos de los Componentes (4)

Ejecutable: Es un programa que se puede ejecutar en un nodo.

- **Biblioteca**: Es una biblioteca de objetos estática o dinámica.
- **Tabla**: Es una tabla de una BD.
- **Archivo**: Es un fichero que contiene código fuente o datos.
- **Documento**: Es un documento.
- **Página Web**: Es una página que se obtiene de la ejecución del sistema.

Los estereotipos y dependencia se ilustran en las figuras 1.2 y 1.3 respectivamente.

Características de los componentes:

- Tienen relaciones de traza con los elementos del modelo que implementan.
- Pueden implementar varios elementos. Por ejemplo, varias clases; Sin embargo la forma exacta en que se crea esta traza depende de cómo van a ser estructurados y modularizados los ficheros de código fuente, dado el lenguaje de programación que se esté usando. (5)

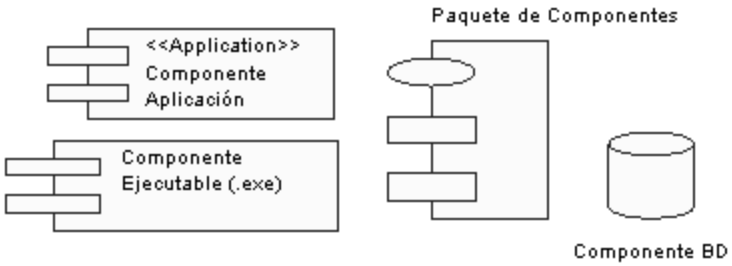

Figura 1.2 Tipos de componentes

Dependencias

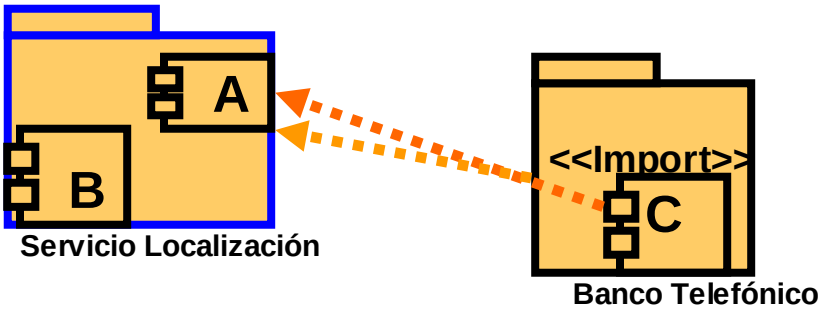

Figura 1.3 Dependencias entre componentes

Arquitectura Basada en Componentes

La definición de la arquitectura basada en componentes cubre aspectos únicamente lógicos y es totalmente independiente de la tecnología con la cual se implementarán los mismos y sobre la cual se hará el despliegue del sistema. Esta vista lógica permite medir el nivel de acoplamiento del sistema y razonar sobre los efectos de modificar o reemplazar un componente. La independencia de la tecnología permite abstraerse de los tecnicismos de éstas, así como elegir la más apta dependiendo del sistema que se esté desarrollando. (6)

Servicio

Un servicio es un elemento del Modelo de Servicios el cual es la base de una Arquitectura Orientada a Servicio. Es proporcionado por un proveedor de servicios y posee una especificación del servicio.

Servicio Informático.

Conjunto de actividades (planeamiento, análisis, diseño, programación, operación, entrada de datos, autoedición, bases de datos, etc.) asociadas al manejo automatizado de la información que satisfacen las necesidades de los usuarios de este recurso (7)

Arquitectura Orientada a Servicios

Estilo de arquitectura, que sustenta los servicios de una organización en contratos formalizados, promoviendo la reusabilidad como modelo para disminuir los costos de desarrollo, el desacoplamiento de tecnologías y la protección de la inversión. (8)

Definición de SOA

Para definir este tipo de arquitectura es necesario dividirla en varias partes para un mejor entendimiento. Estas son:

Servicio

Una función sin estado (Existen servicios asíncronos en los que una solicitud a un servicio crea , por ejemplo, un archivo, y en una segunda solicitud se obtiene ese archivo), auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos. La tecnología concreta utilizada para prestar el servicio no es parte de esta definición. (9)

Orquestación

Secuenciar los servicios y proveer la lógica adicional para procesar datos. No incluye la presentación de los datos. Coordinación. (10)

Sin estado

No mantiene ni depende de condición pre-existente alguna. En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio. (11)

Proveedor

La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor.

Consumidor

La función que consume el resultado del servicio provisto por un proveedor.

1.3 Pruebas

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. (12)

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida: se puede probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, etc. Lo que conduce al principal beneficio de la prueba: proporcionar feedback mientras hay todavía tiempo y recursos para hacer algo. (13)

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces. (14)

1.3.1 Objetivos de las Pruebas

- Encontrar y documentar los defectos que puedan afectar la calidad del software.
- Validar que el software trabaje como fue diseñado.
- Validar y probar los requisitos que debe cumplir el software.
- Validar que los requisitos fueron implementados correctamente
- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo. (15)

Clasificación y aplicación de Pruebas

Cuando se comienza a trabajar en un software determinado, es necesario realizarles distintos tipos de pruebas, para lograr una buena calidad en el mismo y así lograr la menor cantidad posible de errores. Es por esto que las pruebas se clasifican en dos grandes grupos, ya que toda aplicación puede ser probada bajo los siguientes esquemas:

1) Conociendo la función del producto (programa), demostrar que esa función anda bien. Este caso se realiza sobre las interfaces y se lo denomina **PRUEBA DE CAJA NEGRA**.

2) Demostrar que la operación interna del modulo se ajusta a lo especificado y que los componentes internos andan bien, (esta prueba se desarrolla en base a los caminos lógicos del modulo, se denomina **PRUEBA DE CAJA BLANCA**).

Esta última es la prueba que mejor deja ver las dificultades.

Prueba de Caja Blanca:

Permiten 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: (16)

1) Que se ejecutan al menos de una vez todos los caminos independientes de cada modulo.

2) Que se prueben todas las decisiones lógicas en sus ramas verdaderas y falsas.

3) Ejecutar todos los bucles o ciclos con los límites que se les haya definido.

4) Ejecutar las estructuras internas de datos para asegurar su validez.

Métodos o Técnicas de Caja Blanca

1)- **Prueba del camino básico:**

Pasar el diagrama lógico a un grafo de flujo. Utiliza el grafo de flujo en distintos módulos. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (17)

Componentes del Grafo de Flujo:

Arista: Representa flujo de control.

Nodo: representa un conjunto de sentencias.

Región: Área limitada por aristas y nodos. Se considera como un área además lo que está fuera del grafo (otra región).

Complejidad Ciclomática:

Es aquella medida cuantitativa que define la complejidad de un programa, y lo que mide es el número de caminos independientes que tiene el programa (cuando se habla de caminos independientes, se hace referencia a cualquier camino que introduzca un nuevo conjunto de sentencias o un conjunto de decisiones, expresado en el grafos por un nodo). La complejidad Ciclomática da un límite superior que define la cantidad de pruebas que se deben hacer para asegurar que cada línea/sentencia se ejecute al menos una vez. (18)

2) **Prueba de Bucles**

A - Eventualmente puede tener una repetición de ciclos

B - Tiene un tope “n” que dice la cantidad máxima de veces que se puede repetir.

Pruebas que hay que hacer:

1º) Saltar por sobre el bucle.

2º) Pasar una sola vez por el bucle.

3º) Pasar 2 veces por el bucle.

4º) Pasar m veces (siendo m < n).

5º) Pasar n-1 veces.

6º) Pasar n veces.

7º) Pasar n+1 veces.

Si se trata de un bucle anidado:

La técnica es empezar primero por el bucle más interior dejando al resto en sus valores mínimos, probar al bucle interior como si fuera un bucle simple, avanzar hacia fuera manteniendo los bucles que

siguen siendo exteriores en sus valores mínimos y a los interiores ponerlos en valores típicos. (19)
Si se trata de bucles concatenados:

Si son independientes se usan para cada uno la técnica del bucle simple y si no son independientes, o sea si hay alguna relación, usan la técnica de bucles anidados.

Otro Caso, Bucles no estructurados:

En este caso rediseñar el software (borrar y empezar de cero) (20)

Métodos o Técnicas de Caja Negra:

Son complementarias a las de caja blanca. Pero en la práctica se hace normalmente solo la prueba 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.

En este tipo de pruebas se derivan un conjunto de condiciones de entrada que ejercitan completamente todos los requerimientos funcionales del programa. Por eso se pueden encontrar distintos tipos de errores al aplicar este tipo de pruebas.

Errores habituales que se encuentran con pruebas de caja negra:

- Funciones inexistentes o incorrectas.
- Errores de interface.
- Errores de iniciación, comienzo o finalización.
- Errores de rendimiento.
- Errores en estructuras de datos o en accesos a bases de datos externas

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

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

Lo fundamental en este tipo de pruebas es encontrar el subconjunto de todas las entradas posibles del programa, esto es muy complejo, para conseguirlo se siguen diferentes criterios:

- Particiones de Equivalencia.
- Análisis de Valores límite.
- Valores típicos de error.
- Valores Imposibles.

La prueba de caja negra responde a las preguntas:

- Que casos de entrada terminan definiendo buenos casos de prueba.
- Si el sistema es sensible a determinados casos de entrada.
- Que volúmenes (niveles de datos) tolera el sistema.

Se plantean 2 técnicas:

1)- Análisis de Valores Límites:

Pressman dice que no hay una identificación clara de los valores límites, dice que suele haber más errores en los valores límites que en los típicos. Cuando se habla de Valores Limites lo que se dice es que se eligen casos de prueba en los límites o bordes de la clase que se está probando. (Ej.: Si una condición de entrada o salida exige un rango entre B y R, se realizan los casos de prueba para los valores B y R, y además a los valores que están por encima de ellos, es decir A y S). (21)

2)- Partición Equivalente:

Es una técnica de simplificación de pruebas. Hay que dividir el dominio de entrada de un programa en clases de datos que tengan algo en común, de ahí derivan clases de prueba y por lo tanto también derivan clases de errores, de esta forma no hay necesidad de ejecutar muchas pruebas para encontrar errores genéricos. (22)

Para lograr esta se evalúan las clases de equivalencias posibles para una condición de entrada.

Clases de equivalencias: Son los conjuntos de estados (válidos o no) para las condiciones de entrada. Condiciones de entrada: Valor numérico, rango de valores, condición booleana (si o no).

- Si una condición de entrada especifica un rango, se propone definir una clase de equivalencia valida y 2 inválidas.
- Si una condición de entrada especifica un número, un valor, se define una clase de equivalencia igual al caso anterior, (1 valida y 2 inválidas).

- Si una condición de entrada especifica un miembro de un conjunto, se define una condición valida y una invalida.

- Si es booleana, se define una clase de equivalencia igual al caso anterior (1 valida y 1 inválida).

Realización de las Pruebas.

Las diferentes pruebas que deben realizarse se basan en realizar pruebas a diferentes niveles, es necesario probar si cada unidad funciona, luego es necesario probar si los distintos componentes encajan entre sí y por último es necesario probar el sistema globalmente. Este proceso es algo bastante lógico, pues si por ejemplo sólo se prueba el sistema, sería difícil encontrar determinados tipos de errores.

1.3.2 Principios básicos de pruebas.

Deben ser llevadas a cabo por personas distintas a los diseñadores de los programas, así se puede verificar además del correcto funcionamiento del programa su correcta concepción e interpretación. (23) Además de seguir estos 10 principios: (24)

1. La prueba puede ser usada para mostrar la presencia de errores, pero nunca de su ausencia.
2. La principal dificultad del proceso de prueba es decidir cuando parar.
3. Evitar casos de pruebas no planificados, no reusables y triviales a menos que el programa sea verdaderamente sencillo.
4. Una parte necesaria de un caso de prueba es la definición del resultado esperado.
5. Los casos de pruebas tienen que ser escritos no solo para condiciones de entrada válidas y esperadas sino también para condiciones no válidas e inesperadas.
6. Los casos de pruebas tienen que ser escritos para generar las condiciones de salida deseadas.
7. El número de errores sin descubrir es directamente proporcional al número de errores descubiertos.
8. Las pruebas deberían empezar por “lo pequeño” y progresar hacia “lo grande”.

9. Con la excepción de las pruebas de unidad e integración, un programa no deberá ser probado por la persona u organización que lo desarrolló.

10. las pruebas deberían ser realizadas por un equipo independiente.

1.3.3 Tipos de Pruebas

Pruebas de Unidad:

La prueba de unidad se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca. (25)

Pruebas de Integración:

El objetivo es coger los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. (26)

Hay dos formas de integración:

- Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.
- Integración incremental: El programa se construye y se prueba en pequeños segmentos.

En la prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

Las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra, aunque se pueden llevar a cabo unas pocas pruebas de caja blanca.

Pruebas del Sistema:

Verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Algunas de estas pruebas son: (27)

- Prueba de validación: Proporciona una seguridad final de que el software satisface todos los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha sido construido. Durante la validación se usan exclusivamente técnicas de prueba de caja negra.

- Prueba de recuperación: Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.
- Prueba de seguridad: Verificar los mecanismos de protección.
- Prueba de resistencia: Enfrenta a los programas a situaciones anormales.
- Prueba de rendimiento: Prueba el rendimiento del software en tiempo de ejecución.
- Prueba de instalación: Se centra en asegurar que el sistema software desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepciones, por ejemplo con espacio de disco insuficiente o continuas interrupciones.

Pruebas de regresión:

Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad. El propósito de estas pruebas es asegurar que:

- Los defectos identificados en la ejecución anterior de la prueba se ha corregido.
- Los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores.

La prueba de regresión puede implicar la re-ejecución de cualquier tipo de prueba. Normalmente, las pruebas de regresión se llevan a cabo durante cada iteración, ejecutando otra vez las pruebas de la iteración anterior. (28)

Pruebas de Seguridad

La prueba de seguridad intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de accesos impropios. Por supuesto, la seguridad del sistema debe ser probada en su invulnerabilidad frente a un ataque frontal, pero también debe probarse en su invulnerabilidad a ataques por los flancos o por la retaguardia. (29)

Durante la prueba de seguridad, el responsable de la prueba desempeña el papel de un individuo que desea entrar en el sistema. ¡Todo vale! Debe intentar conseguir las claves de acceso por cualquier medio, puede atacar al sistema con software a medida, diseñado para romper cualquier defensa que se haya construido, debe bloquear el sistema, negando así el servicio a otras personas, debe producir a propósito errores del sistema, intentando acceder durante la recuperación o debe curiosear en los datos sin protección, intentando encontrar la clave de acceso al sistema, etc. (30)

Con tiempo y recursos suficientes, una buena prueba de seguridad terminaría por acceder al sistema. El papel del diseñador del sistema es hacer que el coste de la entrada ilegal sea mayor que el valor de la información obtenida.

1.3.4 Pruebas en SOA (Arquitecturas Orientadas a Servicios)

En arquitecturas orientadas en servicios, es muy importante realizar algunas pruebas específicas para lograr interoperabilidad y la identificación de vulnerabilidades, logrando así tener una la infraestructura robusta, escalable, interoperable y segura.

Los tipos de pruebas mas comunes que deben ser aplicadas en este tipo de arquitecturas son:

Pruebas carga:

El objetivo de las pruebas de carga es determinar el rendimiento del sistema bajo condiciones de carga que se aproximan a la realidad esperada en producción. Esto permite: (31)

1. Determinar la escalabilidad del sistema en TX / seg vs. # Usuarios concurrentes.
2. Validar que el dimensionamiento de la infraestructura de hardware fue el adecuado.
3. En este tipo de pruebas, a diferencia de las pruebas de estabilidad, la carga se va aumentando cada vez más para determinar la escalabilidad del sistema. Una vez determinada la escalabilidad de la plataforma, se usa este valor para las pruebas de estabilidad.

Pruebas de Volumen:

Encontrar debilidades en el sistema al momento de manejar grandes volúmenes de datos durante prolongados períodos de tiempo, el objetivo principal es determinar si la plataforma de integración se degrada o deja de funcionar al manejar grandes volúmenes de datos. (32)

Pruebas de Estabilidad:

Miden la estabilidad de los servicios con una fuerte carga sostenida en el tiempo (la carga no cambia). Los servicios no deben fallar y los tiempos de respuesta para una carga constante deben permanecer constantes. Gráfica tipo: tiempo de respuesta de un WS vs. Tiempo (debe tender a líneas de pendiente 0).

Las **aserciones** que deben considerarse para realizar pruebas unitarias en Web Services son: (33)

1. Conformidad con el XML Schema.
2. Contenido simple, Expresiones X path.
3. SOAP Fault.

4. Transacción por segundo (TPS).
5. Máximo Tiempos de Respuesta (Response Time).
6. Máximo numero de errores.
7. Average o promedio.

1.4 Tendencias actuales sobre desarrollo de pruebas: Uso de Herramientas.

Actualmente el desarrollo de las tecnologías de la Información y las Comunicaciones crece vertiginosamente y se hace necesario el uso de herramientas potentes que verifiquen la calidad de los productos que salen al mercado mundial, para garantizar su total funcionamiento. Proceso este que se hace muy difícil o casi imposible realizarlo manualmente, si se quieren obtener resultados óptimos. A continuación se presentan detalladamente algunas herramientas que se usan en la actualidad, con muy buenos resultados.

1.4.1 Herramientas para el entorno de pruebas.

Dan soporte a las actividades de QA, pruebas de integración, pruebas de sistema, diagnóstico o afinado de las aplicaciones en los entornos de preproducción o certificación y los de producción, que generalmente son difíciles de realizar de una forma integrada, comprenden herramientas de análisis estático, automatización de pruebas funcionales, de carga, de rendimiento, de estrés... el afinado del código, etc. (34)

Las herramientas de mayor utilidad para los entornos de pruebas son:

JMeter es una herramienta Java dentro del proyecto Jakarta, que permite realizar Pruebas de Rendimiento y Pruebas Funcionales sobre Aplicaciones Web. (35)

ChecKing de ALS ChecKing es la herramienta de monitorización del proceso de desarrollo software y sus resultados, que cubre las necesidades de organizaciones que desean controlar la calidad del software antes de su puesta en producción. (36)

JKing QA de ALS

JKing QA es una herramienta de análisis estático, pensada para facilitar y automatizar el proceso de adopción de los estándares de calidad para un departamento de calidad. (37)

JTest, C++Test y Insure++ de Parasoft

Herramientas de Parasoft, que ayudan a prevenir errores por medio de su capacidad de análisis estático personalizable, que permite hacer cumplir automáticamente alrededor de 300 estándares de codificación de la industria, crear y hacer cumplir estándares propios de codificación, o construir estándares destinados a un proyecto o grupo en particular. (38)

Open Load Tester de Open Demand Systems

Open Load Tester es la primera solución rápida de optimización de rendimiento basada en navegador, fácil de usar, para las pruebas de carga y stress de aplicaciones y sitios web dinámicos. (39)

QACenter Enterprise Edition de Compuware

QACenter Enterprise Edition analiza cómo se distribuye el tiempo y permite a los equipos de calidad tomar decisiones estudiadas acerca del cambio de prioridades, cuánto tiempo adicional es necesario o cuántos recursos adicionales se requieren para cumplir los plazos establecidos. (40)

QaKing de ALS

QaKing es la herramienta de análisis estático, pensada para facilitar y automatizar el proceso de certificación del cumplimiento de los estándares de codificación y buenas prácticas del código Java, JSP, Javascript, HTML y XML. (41)

QALoad de Compuware

Es una herramienta de pruebas de carga que ayuda a los equipos de pruebas, desarrolladores y jefes de proyecto a realizar pruebas de carga efectivas a aplicaciones distribuidas. (42)

Security Tester de Fortify

Fortify Security Tester para Visual Studio 2005 Team System, proporciona las pruebas de seguridad eficaces a los equipos de desarrollo y QA, permitiéndoles verificar la adecuación a los estándares de seguridad, y posibles vulnerabilidades en el código de sus aplicaciones antes de su despliegue. (43)

SOATest de Parasoft

Herramienta de Parasoft, que proporciona las pruebas y verificación instantáneas de Web Services, simplifica los desarrollos SOA automatiza las pruebas funcionales cliente/servidor, pruebas de regresión, pruebas de carga y rendimiento, etc. (44)

.TEST de Parasoft

.TEST es una unidad de pruebas automatizada y productos de análisis de código estándar, que trabaja sobre clases escritas en la plataforma Microsoft .NET, sin requerir que los desarrolladores realicen un solo caso de prueba o stub. (45)

Webking de Parasoft

Herramienta de Parasoft, para automatizar las pruebas de las aplicaciones Web (análisis de riesgos de los sites, pruebas funcionales, pruebas de carga y rendimiento y análisis de seguridad).

Luego de mencionar brevemente los distintos tipos de herramientas que se usan a nivel mundial para la realización de pruebas, se detalla a continuación algunas de las mas importantes y que se consideran sean las mas apropiadas para ser utilizadas por los proyectos productivos de la faculta 7 de la UCI, así como por el laboratorio de calidad de dicha facultad. (46)

1.4.1.1 JMeter

JMeter es una herramienta de carga para llevar acabo simulaciones sobre cualquier recurso de Software.

Inicialmente diseñada para pruebas de estrés en aplicaciones web, hoy en día, su arquitectura ha evolucionado no sólo para llevar acabo pruebas en componentes habilitados en Internet (HTTP), sino además en Bases de Datos, programas en Perl, requisiciones FTP y prácticamente cualquier otro medio. (47)

Además, posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de requisiciones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción.

En este sentido, simula todas las funcionalidades de un Navegador ("Browser"), o de cualquier otro cliente, siendo capaz de manipular resultados en determinada requisición y reutilizarlos para ser empleados en una nueva secuencia. (48)

Una de las ventajas que posee el JMeter, es que tiene una estructura en árbol que le da potencia, permitiendo que sea la imaginación de quien la use la que ponga los límites a la hora de diseñar el plan de prueba. Y brinda mayor cantidad de variantes para recoger los resultados obtenidos, que el resto de las herramientas gratis, lo que permiten hacer un análisis exhaustivo de las pruebas realizadas. Sin duda, este es un producto a tener en cuenta cuando no se dispone de una herramienta

comercial más potente como Load Runner (que no es gratis), pero es una buena alternativa para hacer las pruebas en cuestión. (49)

1.4.1.2 SOATest de Parasoft

SOATest proporciona verificación instantáneas de servicios web, simplifica el desarrollo SOA, automatiza las pruebas funcionales de cliente/servidor, pruebas de regresión, pruebas de carga/rendimiento y más. (50)

Descripción:

La automatización de pruebas de servicios web permite a los usuarios verificar todos los aspectos asociados, desde la validación WSDL, hasta la unidad y pruebas funcionales del cliente y del servidor. SOATest localiza los elementos claves de los servicios web y tales como la interoperabilidad, seguridad, cambio de mantenimiento y escalabilidad. (51)

SOATest de Parasoft es el estándar de Arquitectura Orientada a Servicios más aceptado por la industria para su seguridad y fiabilidad. SOATest es una solución probada con clientes como: Yahoo!, Sabre, LexisNexis, IBM, y el gobierno de los Estados Unidos. Las empresas seleccionan SOATest de Parasoft por su gran variedad de funcionalidad, incluyendo validación WSDL, pruebas unitarias y funcionales del cliente, pruebas unitarias y funcionales del servidor, pruebas de desarrollo así como la habilidad de modelar y probar gráficamente escenarios complejos. SOATest localiza los aspectos fundamentales de los servicios web, como la interoperabilidad, seguridad, cambios de gestión y escalabilidad. SOATest permite que cualquier recurso de prueba creado por un ingeniero pueda ser aprovechado por su equipo de calidad en un escenario de pruebas unitarias y pruebas de carga sin utilizar scripting. SOATest de Parasoft es también la única solución de servicios web que crea automáticamente pruebas de intrusión de seguridad en servicios web, como inyecciones SQL, inyecciones Xpath, sobrecarga de parámetros, bombas XML y uso de entidades externas. Utilizando SOATest a lo largo del ciclo de desarrollo del software, los usuarios pueden prevenir errores, mejorar la calidad y fiabilidad del producto. (52)

Debido a su naturaleza flexible, SOATest es una elección ideal para ingenieros de desarrollo y profesionales asociados a la gestión de calidad, dado que sus pruebas unitarias pueden ser aprovechadas en diferentes escenarios, así como las pruebas de carga, sin realizar scripting adicional.

SOATest se distribuye en varias ediciones, personalizadas según roles específicos y procesos:

- SOAtest Professional Edition

- SOAtest Enterprise Edition

Características (53)

- Pruebas de servicios web sin scripting
- Seguridad-WS, SAML, Username Token, X.509, encriptación XML y firma XML.
- Verificación de consultas, validación y carga de pruebas.
- Creación automática de pruebas desde WSDL, WSIL, UDDI y HTTP Traffic.
- Test asíncronos: JMS, Parlay (X), SCP, WS-Addressing.
- Completa la cobertura del workflow de pruebas mediante escenarios complejos y múltiples puntos de finalización.
- Verificación de esquemas y semántica WSDL, y compilación en WS-I Basic Profile 1.1
- Soporte de tipos MIME
- Soporta SNMP y Monitores JMX
- Generación de informes detallados en HTML, XML y formatos de texto.
- Gráficos y tablas en tiempo real.

Beneficios (54)

- Asegura la fiabilidad, calidad, seguridad e interoperabilidad de un servicio web.
- Pruebas de intrusión integradas con pruebas funcionales, para completar la cobertura.
- Prevención de errores e identificación de posibles.
- Verificación de la integridad de los datos y la funcionalidad servidor/cliente.
- Acelera el tiempo de mercado.

Requerimientos

Protocolos admitidos

- HTTP 1.0
- HTTP 1.1 w/Keep-Alive Connection
- HTTPS
- TCP/IP
- JMS

Plataformas

- Windows 2000/XP
- Linux
- Solaris

Requerimientos del sistema (55)

Windows

- Windows 2000 o Windows XP
- Mínimo 512 MB RAM por procesador (1024 MB recomendado para pruebas de carga)
- JRE 1.4.2

Unix

- Linux o Solaris
- Mínimo 512 MB RAM por procesador (1024 MB recomendado para pruebas de carga)
- JRE 1.4.2

1.4.1.3 JTest de Parasoft

JTest 8.0 ofrece varios avances para las industrias desarrolladoras de software de alta calidad. Estos avances tecnológicos están concentrados en la realización de pruebas, para ayudar a los equipos a verificar de manera automática la funcionalidad de aplicaciones cada vez mas complejas, en empresas con sistemas en permanente cambio (J2EE, servicios de SOA/Web), todo esto para generar un incremento en la satisfacción del cliente, una reducción en tiempo de entrega del software así como una disminución del riesgo de generar software defectuoso o con problemas de vulnerabilidad. (56)

Descripción

JTest 8.0 cuenta con un nuevo módulo el **"BUG DETECTIVE"**. Aun siendo un análisis estático, el **detector de bugs** es capaz de detectar errores, que antes sólo podían ser detectados en las fases de integración y ejecución del código. (57)

"Bug detective" encuentra y muestra errores por automatización y simulación de las rutas, errores que serían difícilmente detectados al ejecutar los test o con test manuales; de esta manera JTest 8.O enriquece y fortalece nuestra política de prevención. Los usuarios pueden encontrar, diagnosticar, y fijar clases de los errores del software que pueden evadir los test de código estándar y los test unitarios.

Para reducir complejidad de los test, JTest ahora ofrece la generación y la ejecución automatizada de los casos de la prueba de tipo " Cactus test cases " en un formato " in-container " es decir JTest 8.0 a través de la simulación de un entorno real, posibilita un test en tiempo de ejecución que permite la

temprana detección de defectos en el código que de otro modo pasarían inadvertidos hasta etapas tan avanzadas como el aseguramiento de la calidad, la integración o peor aun durante la producción de la aplicación, encontrar un error tipo "resource leak", en etapas tan avanzadas representan un alto costo y/o la imposibilidad de detectar y fijar dichos errores. (58)

El **"TRACER"** de JTest sustituye al Test Case Sniffer. El ahora llamado "Tracer" ha aumentado su utilidad y mejorado su eficacia.

El "TRACER" como el Sniffer en la versión precedente, permite a los usuarios **crear de una manera fácil y rápida casos de prueba reales** para los test funcionales de Junit, que reflejan el correcto comportamiento funcional y sus operaciones individuales en uso. Estos casos de test "positivos" pueden ser utilizados para identificar si nuevos cambios afectan a la aplicación ya existente (test de no regresión). Con JTest Tracer es mucho más fácil controlar el trazo "tracing" de una aplicación en ejecución y profundizar en los detalles del test. (59)

Además, JTest 8.0 incluye un nuevo módulo de revisión del código, que ayuda a automatizar el proceso de revisión del mismo facilitando la participación y la comunicación y por lo tanto hace revisiones de código más productivas y prácticas para las organizaciones. El módulo permite que los usuarios definan y administren listas de distribución, las agrupaciones para las notificaciones y los lineamientos de la revisión de código. Este módulo de revisión de código beneficiará a equipos distribuidos del desarrollo que no pueden participar logísticamente en sesiones físicas de la revisión de código. (60)

Finalmente, JTest 8.0 incluye la **parametrización del caso de prueba**, una capacidad que permite a Jtest-generated o a user-difined Junit test extender y personalizar de acuerdo a las especificaciones particulares los casos de pruebas, Los usuarios pueden ahora controlar la gama, el tipo, y ordenar los valores de entrada de la prueba. Esto permite el desarrollo y la ejecución de escenarios complejos del caso de la prueba que aseguran una amplia y cuidadosa cobertura del test, y que validan la respuesta del código a una amplia gama de las acciones previstas e inesperadas del usuario. (61)

JTest se distribuye en varias ediciones, personalizadas según roles específicos y procesos

- JTest Professional Edition
- JTest Architect Edition
- JTest Server Edition

Características (62)

- Comprueba el cumplimiento de un conjunto configurable de 700 reglas, incluidas 100 reglas de seguridad.
- Corrige violaciones de 250 reglas.
- Permite la creación de reglas personalizadas mediante la modificación de los parámetros, usando una herramienta gráfica o proporcionando código que demuestre un ejemplo de la violación de la regla.
- Identifica código duplicado o que no se usa.
- Soporta Struts, spring, Hibernate, EJBs, JSPs, servlets...
- Prueba métodos individuales, clases, o grandes y complejas aplicaciones.
- Genera casos de prueba funcionales JUnit que capturan el comportamiento real del código.
- Genera casos de prueba JUnit y Cactus que muestran la fiabilidad y capturan el comportamiento.
- Parametriza los casos de prueba para usarlos con múltiples valores (generados en tiempo de ejecución, definidos por el usuario o provenientes de otras fuentes de datos).
- Monitoriza la cobertura de las pruebas y consigue una alga cobertura usando el análisis de cobertura de las ramas.
- Identifica fugas de memoria durante la ejecución de las pruebas.
- Permite realizar pruebas paso a paso con el depurador.
- Integrado con Eclipse, Intel, IDEA y RAD.
- Integrado con CVS, Clear Case, Subversión, StarTeam.
- Posibilidad de compartir la configuración de las pruebas con el equipo y con organización.
- Generación de informes tipo texto, HTML, y XML.
- Recopila los resultados de los test para poder ver la evolución de su calidad a lo largo del tiempo.
- Proporciona una interface interactiva y un modo de comandos.

Beneficios (63)

- Despliega pruebas de unidad y buenas prácticas.
- Mejora la fiabilidad del código, así como la funcionalidad, seguridad y ejecución rápida.
- Obtiene un experto feedback instantáneo de la calidad del código y los defectos potenciales.
- Previene de las modificaciones del código defectuosas, antes de verificar su funcionalidad.
- Efectúa pruebas complejas con la mínima intervención del usuario.
- Permite emplear menos tiempo en la fase de pruebas y más tiempo en tareas creativas.

- Reduce el riesgo en planificación, en estimación de presupuesto y la salida de releases incompletos.
- Optimiza el tiempo de revisión del código.
- Asegura que las buenas prácticas son aplicadas constantemente y uniformemente en todo el equipo.
- Monitoriza la calidad global del proyecto, especifica segmentos del proyecto y progresa a través de las metas de calidad.

Requisitos de sistema

Plataformas

- Windows 2000, Windows XP, Windows 2003 Server
- Solaris
- Linux

Hardware

- Intel Pentium III 1.0 GHZ ó superior recomendado
- SVGA (800x600) mínimo (1024 x 768 recomendado)
- 512 MB RAM mínimo; 1 GB RAM recomendado

IDE (solo para el plugin)

- Eclipse 3.2
- Eclipse 3.1
- Eclipse 3.0
- IBM Rational Application Developer 6.0

1.5 Estrategias de pruebas del software:

Para conseguir el éxito de las pruebas durante todo el ciclo de vida de una aplicación Web, hay que dividirlas en las siguientes etapas: (64)

- Planificación de las pruebas.
- Diseño de las pruebas.
- Implementación de las pruebas.
- Ejecución de las pruebas.
- Evaluación de las pruebas.

Los productos de desarrollo del software fundamentales que se desarrollan en la etapa de Pruebas son:

- Plan de Pruebas.
- Casos de Prueba.
- Informe de evaluación de Pruebas.
- Modelo de Pruebas, que incluye Clases de Prueba, Entorno de Configuración de Pruebas, Componentes de Prueba y los Datos de prueba.

Los participantes responsables de las realizar las actividades y los productos de desarrollo del software son:

- Diseñador de pruebas: Es responsable de la planificación, diseño, implementación y evaluación de las pruebas. Esto conlleva generar el plan de pruebas y modelo de pruebas, implementar los casos de prueba y evaluar los resultados de las pruebas. Los diseñadores de prueba realmente no llevan a cabo las pruebas, sino que se dedican a la preparación y evaluación de las mismas. (65)
- Probador (Tester): Es responsable de desarrollar las pruebas de unidad, integración y sistema, lo que incluye ejecutar las pruebas, evaluar su ejecución, recuperar los errores y garantizar los resultados de las pruebas. (66)

Durante la fase de Inicio puede hacerse parte de la planificación inicial de las pruebas cuando se define el ámbito del sistema. Sin embargo, las pruebas se llevan a cabo sobre todo cuando un producto de desarrollo software es sometido a pruebas de integración y de sistema. Esto quiere decir que la realización de pruebas se centra en las fases de Elaboración, cuando se prueba la línea base ejecutable de la arquitectura, y de construcción, cuando el grueso del sistema está implementado. Durante la fase de Transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y a las pruebas de regresión. (67)

Debido a la naturaleza iterativa del esfuerzo de desarrollo, algunos de los casos de prueba que especifican cómo probar los primeros productos de desarrollo software pueden ser utilizadas también como casos de prueba de regresión que especifican cómo llevar a cabo las pruebas de regresión sobre los productos de desarrollo software siguientes. El número de pruebas de regresión necesarias crece por tanto de forma estable a lo largo de las iteraciones, lo que significa que las últimas iteraciones requerirán un gran esfuerzo en pruebas de regresión. Es natural, por tanto, mantener el modelo de pruebas a lo largo del ciclo de vida del software completo, aunque el modelo de pruebas cambia constantemente debido a: (68)

- La eliminación de casos de prueba obsoletos.
- El refinamiento de algunos casos de prueba en casos de prueba de regresión.
- La creación de nuevos casos de prueba para cada nuevo producto de desarrollo de software.

Planificación de las Pruebas.

La fase de pruebas necesita una organización seria y fiable. Las pruebas funcionan cuando encuentran errores. Debe de realizarse una planificación exhaustiva.

Terminación de las Pruebas.

El objetivo de las pruebas consiste en encontrar errores, pero si ya no se encuentran errores (no quiere esto decir que no los haya) debe seguirse un criterio de terminación de las pruebas; el criterio puede ser:

- Cuando el tiempo de la prueba ha expirado.
- Cuando todos los casos de prueba se ejecutan sin error.

1.6 Modelos usados para la aplicación de pruebas de software.

Los modelos que a continuación se mencionan, constituyen la base para la elaboración de cualquier procedimiento de pruebas de software, los que pueden adaptarse, a cualquier arquitectura desarrollada, así como a cualquier metodología de desarrollo como es el caso de RUP.

1.6.1 Modelo Cascada:

El modelo cascada es el primer modelo según el cual se basan los demás. Fue definido por Winston Royce en el de 1970. El modelo de cascada se forma de seis fases, estas son:

Ingeniería y Análisis del Sistema: Debido a que el software es siempre parte de un sistema mayor el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software. (69)

Análisis de los requisitos del software: el proceso de recopilación de los requisitos se centra e intensifica especialmente en el software. El ingeniero de software (Analistas) debe comprender el ámbito de la información del software, así como la función, el rendimiento y las interfaces requeridas. (70)

Diseño: el diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El

proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación. (71)

Codificación: el diseño debe traducirse en una forma legible para la máquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada la codificación puede realizarse mecánicamente.

Prueba: una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren. (72)

Mantenimiento: el software sufrirá cambios después de que se entrega al cliente. Los cambios ocurrirán debido a que se hayan encontrado errores, a que el software deba adaptarse a cambios del entorno externo (sistema operativo o dispositivos periféricos), o debido a que el cliente requiera ampliaciones funcionales o del rendimiento. (73)

Desventajas:

- Los proyectos reales raramente siguen el flujo secuencial que propone el modelo, siempre hay iteraciones y se crean problemas en la aplicación del paradigma.
- Normalmente, es difícil para el cliente establecer explícitamente al principio todos los requisitos. El ciclo de vida clásico lo requiere y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos productos.
- El cliente debe tener paciencia. Hasta llegar a las etapas finales del proyecto, no estará disponible una versión operativa del programa. Un error importante no detectado hasta que el programa este funcionando puede ser desastroso. (74)

La ventaja de este método radica en su sencillez ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software. El mismo se muestra en la Figura 1.4.

En este modelo se tiene la particularidad de al término de cada fase se realiza la documentación pertinente. Pero tiene una falla, no permite una interacción entre fases; es decir, si al término de una de las fases se tomó o se detecta la falta de un parámetro en los requisitos el modelo no permite corregir este error. Años más adelante se realiza la corrección que permite dicha interacción en el modelo. (75)

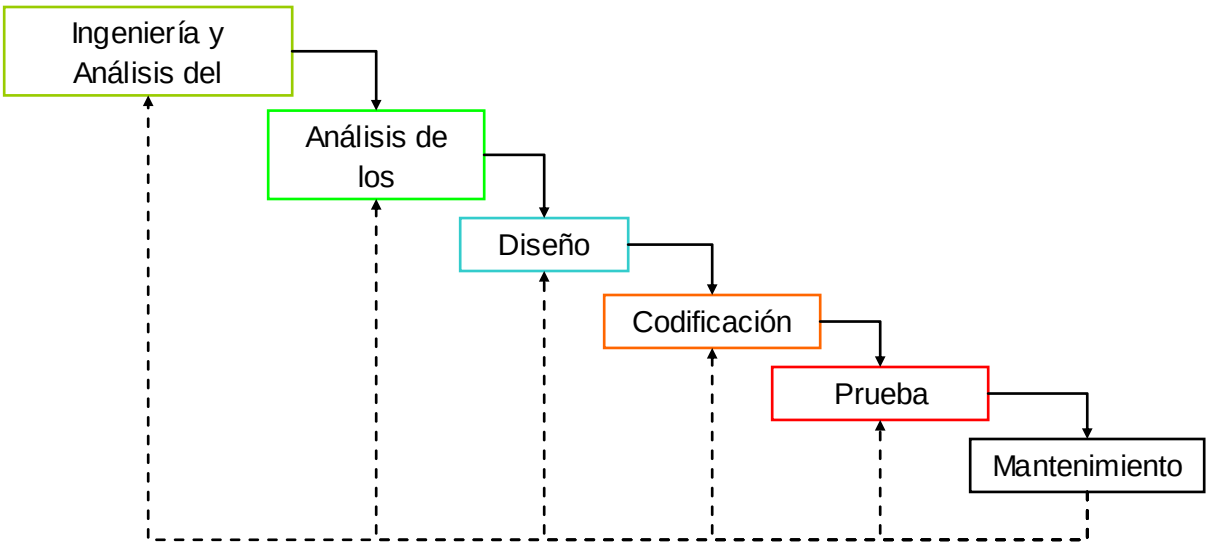

Figura 1.4 Modelo en Cascada

1.6.2 Modelo V:

EL modelo v es un modelo definido después que el modelo cascada , lo cual implica que posee características que el modelo cascada carece, por ejemplo en el modelo v se tiene la facilidad de que en el momento en el cual se está realizando una fase es posible realizar la documentación para las pruebas que se realizaran mas adelante; utilizando la analogía adoptada en clase al momento de estudiar para un examen es mucho mas sencillo hacerlo si antes es proporcionado un cuestionario en el cual se traten temas similares a los que se tratarán en el examen. (76)

El modelo en V deriva directamente de la aplicación de pruebas de verificación y validación a un ciclo de vida de desarrollo en cascada. Es fácilmente extensible a modelos de desarrollo iterativos si se considera la V como una Iteración del proyecto completo (77), como se muestra en la Figura 1.5.

En consecuencia, un plan de pruebas, inmerso en el proceso de desarrollo iterativo, irá evolucionando conforme el proyecto va transcurriendo por sus sucesivas iteraciones.

Cabe destacar que el modelo es suficiente en muchos de los proyectos porque se preocupa de que desde etapas tempranas se vayan diseñando las pruebas que más tarde en el ciclo de vida serán ejecutadas. (78)

1.6.3 Modelo W

El modelo en W surge como refinamiento del *modelo en V*. Refleja mejor la interdependencia que existe entre equipo de desarrollo y el equipo de pruebas a lo largo de todo el proceso de desarrollo del sistema destacando los siguientes dos puntos: (79)

- En las primeras etapas se consideran labores de pruebas que no aparecen reflejadas en el modelo original como son la revisión de los requisitos, la revisión de la especificación del sistema, la revisión de la arquitectura y del diseño de detalle y las revisiones de código.
- En las etapas finales también se distingue del modelo original en V porque se desglosan las tareas de pruebas propiamente dichas y las labores de depuración y corrección de los errores detectados, que serán llevadas a cabo por desarrolladores. (80)

Según el gráfico del modelo en W de la figura 1.6, las labores de pruebas aparecen en los bloques verdes. Los bloques azules representan el resto de actividades del ciclo de vida del software antes de su puesta en producción. En las primeras etapas se concentran labores de verificación de productos no software y de preparación para la verificación y validación del software. En las etapas finales se desarrolla todo el esfuerzo de verificación y validación del software producido. (81)

Figura 1.5 Modelo V

Figura 1.6 Modelo W

Al terminar el presente capítulo se arribaron a las siguientes conclusiones:

- La Arquitectura Basada en Componentes, con su reutilización de código, gana cada día mas espacio en las industrias del software, pues disminuye el costo de la producción, y el tiempo de ejecución de cualquier proyecto.
- La Arquitectura Orientada a Servicios crece cada día más, con la aplicación del modelo cliente servidor y la nueva era de la Web, que ha desplazado casi totalmente a las aplicaciones de escritorio.
- La aplicación de pruebas durante todo el ciclo de vida de un producto de software es muy importante, para lograr de una forma exitosa la satisfacción del cliente.
- El uso de herramientas para la aplicación de pruebas, es cada día mayor a escala internacional, lo que facilita en gran medida el desarrollo de las pruebas y mejora la calidad de los productos que llegan a manos del cliente.
- Las estrategias y métodos de pruebas aplicados por las distintas industrias del software, son muy variados, aunque todos persiguen el mismo objetivo.

Capítulo2: Caracterización de la producción en la Facultad 7 y Diagnóstico de los proyectos productivos que desarrollan aplicaciones Web.

2.1 *Introducción*

En este capítulo se realiza una caracterización de la producción de aplicaciones Web en la Facultad 7 con el objetivo de diagnosticar el estado actual de la producción, en cuanto a la necesidad de una metodología que guie el proceso de desarrollo del producto. Para ello se emplearon diferentes técnicas como la entrevista a diferentes personas relacionadas con la producción de software en la Facultad 7, así como a los integrantes de los proyectos productivos con diferentes roles y la revisión de documentos internos de los proyectos a través de auditorias y revisiones internas. Estas técnicas facilitan la detección de problemas en la planificación y revisión de los productos y que están basadas específicamente en la aplicación de pruebas de software desde la concepción del producto.

2.2 Producción de software en la Facultad 7

La producción de software en la facultad siete se divide en dos grandes polos productivos: El polo de Procesamiento de Imágenes y Señales y el polo Informática para la Salud, dentro de los cuales se encuentran todos los proyectos productivos que dan soporte a la facultad, divididos por áreas temáticas, además del área temática de calidad que es atendida directamente por la dirección central de Calidad de la Universidad.

Ambos polos tienen como objetivo elaborar y mantener software que permitan contribuir al proceso de informatización en el sector de la salud, aunque en el procesamiento de imágenes es aplicable a otros sectores. Esta responsabilidad es de todo los trabajadores de la facultad, a través de la dirección del vice decano de producción, el cual organiza la producción por áreas temáticas, facilitando que se creen por varios grupos diferentes proyectos. Cuando surge un nuevo proyecto se discute en la Infraestructura Productiva (IP), entonces se decide si se acepta o no. En caso positivo se lleva el proyecto a la facultad y se le asigna un área temática.

Por lo general cada proyecto tiene asignado un laboratorio con el equipamiento necesario, constituyendo este su puesto de trabajo. En este momento en la facultad existen 24 proyectos vinculados al polo Informática para la Salud y 5 proyectos generales con varios subproyectos en el polo Procesamiento de Imágenes y Señales, además del proyecto de calidad. De forma general la Facultad 7 cuenta hoy con 30 proyectos productivos en los cuales se encuentran vinculados directamente a la producción un total de 505 estudiantes y 66 profesores de los polos científicos y el área temática de Calidad, más 49 estudiantes que están vinculados a proyectos a nivel central, así como al MININT y MINFAR por lo que no tributan directamente a la facultad suman 554. Este número

representa un 53.3 % de la matrícula total de la Facultad que en estos momentos es de 1040 estudiantes.

La figura 2.1 muestra la estructura actual de la facultad, comenzando por la direccion de la facultad hasta cada uno de los proyectos productivos de las distintas áreas temáticas.

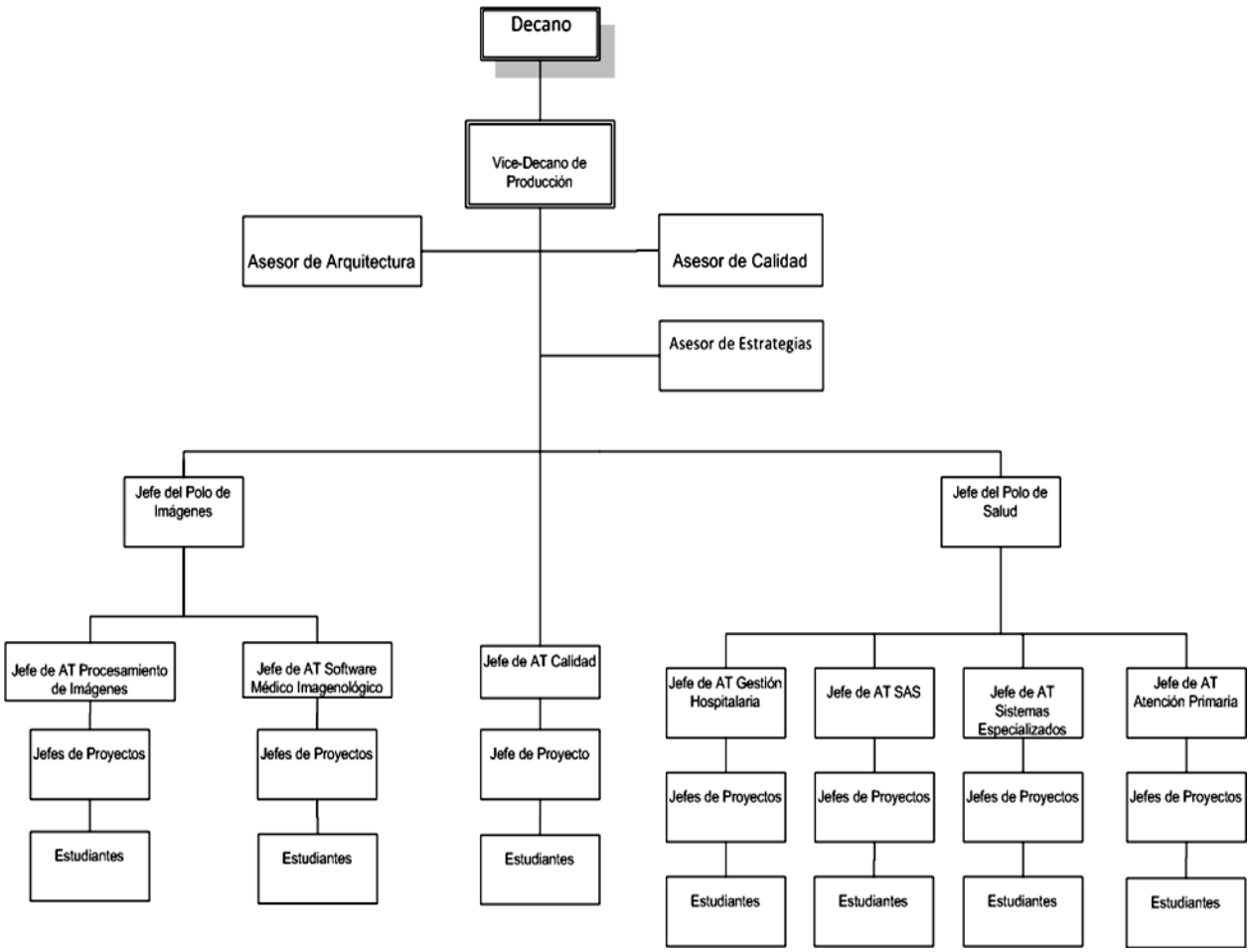

Figura 2.1 Estructura de la Facultad 7.

La Tabla 2.1, muestra la distribución de estudiantes y profesores en las distintas áreas temáticas, así como la cantidad de proyectos incluidos en las mismas y la Tabla 2.2 muestra la distribución de proyectos por las diferentes áreas Temáticas de la Facultad 7.

Polo Informática Para la Salud				
Área Temática	**Cantidad de Proyectos**	**Cantidad de Estudiantes**	**Cantidad de Profesores**	**Total Vinculados**

Sistemas Especializados	4	74	9	83
Sistemas de Atención para la Salud	7	153	13	166
Atención Primaria para la Salud	9	64	9	73
Gestión Hospitalaria	4	97	20	117
Total	24	388	51	439
Polo Procesamiento de Imágenes y Señales				
Software imagenológico para la Salud	2	32	4	36
Software para procesamiento de imágenes	3	48	6	54
Total	5	80	10	90
Área Temática de Calidad				
Calidad	1	37	5	42
Total general	30	505	66	571

Tabla 2.1 Distribución de la producción en la Facultad 7

Área Temática Sistemas Especializados
Red Nacional de Nefrología
Sistema Informatizado para el SIUM
Sistema de Rehabilitación Integral

Control Sanitario Internacional
Area Temática Sistemas de Atención para la Salud (SAS)
Atención Remota a Servidores (ARS)
Sistema de Información Estadística Complementario de Salud
Colaboración Médica
Sistema de Gestión de Información para la Planificación y Balance Material en el MINSAP
Sistema para la Formación de Recursos Humanos en las Secretarias Docentes del MINSAP
Portal de la Facultad
Clientes Inteligentes
Área Temática Atención Primaria para la Salud (APS)
Registro de Partos y Nacimientos
Registro de Vacunación

Registro de Fallecidos
Registro de Indicadores y Conductas de la Atención Primaria
Registro de Actividades Diarias
Registro de Ciudadano Extranjero.
Estadística Descriptiva del Registro de Áreas de Salud
Estadística Descriptiva del Registro de Enfermedades de Declaración Obligatoria.
Estadística Descriptiva del Registro de Población
Área Temática Gestión Hospitalaria
Bloque Quirúrgico Oftalmológico
Sistema de Gestión Hospitalaria
Médico de la Familia UCI
Hospital UCI

Área Temática Software Imagenológico para la Salud
Alas RIS a. Alas RIS b. Dictofonía y Transcripción c. Sintetizador de voz
Alas PACS a. Viewer b. Server c. DMail d. Burner e. Bandeja de Casos f. Viewer Lite g. DICOM Files Creator h. Cirugía guiada i. Visor oftalmología
Área Temática Software para procesamiento de imágenes
Neurociencia a. DIANA b. MAPIC c. Sonido d. Dislexia y discalculia
Sistema integral de vigilancia a. Chapas b. Cámaras c. Análisis de video
SASA

Tabla 2.2 Distribución de Proyectos por Áreas Temáticas

El proyecto de Calidad de la Facultad 7 constituye otra área temática en la misma pues este proyecto pertenece al área temática de calidad del software a nivel central. Este proyecto está formado por estudiantes de tercero, cuarto y quinto año de la facultad. Una vez concluido el software perteneciente

a cada proyecto, el grupo de calidad es el encargado de revisarlo aplicando varias pruebas para asegurar la calidad de estos productos, y luego es pasado a la dirección Central de calidad que es la encargada de certificar el software

Las pruebas que se aplican hasta el momento son de caja negra; los integrantes del proyecto realizan diferentes casos de pruebas con el objetivo de encontrar errores, una vez detectada las no conformidades del producto, se registran y se le comunica al equipo de desarrollo del software. Este proceso es muy importante porque permite que el software antes de ser entregado al cliente este libre de errores, y agiliza el proceso de obtención del certificado de calidad.

2.3 Aplicación de Técnicas para el diagnóstico.

Las principales técnicas empleadas que permitieron fundamentar la investigación de este trabajo fueron las siguientes:

La entrevista

Es un método de recopilación de datos empíricos, un proceso de comunicación entre dos o más personas, generalmente de forma oral donde las preguntas al entrevistado se hacen por vía directa, se deben desarrollar preguntas que permitan respuestas precisas.

La revisión de documentos

Se utiliza para recoger la información que se encuentra registrada en un documento establecido. Se utilizaron los expedientes de proyectos en las distintas áreas temáticas revisados en auditoría en la facultad 7, etc.

La encuesta

Es una técnica que tiene como objetivo obtener cierta información deseada de un sujeto (trabajador) preseleccionado de antemano dentro de una muestra representativa, por medio de una conversación directa cuyas pautas vienen indicadas en un guión o cuestionario que ha sido previamente diseñado y probado en una muestra piloto o muestra inicial.

Se realiza cuando la información que se realiza puede ser obtenida a partir de la respuesta que una persona o varias puedan dar a un cuestionario pre elaborado, y las mismas están dispuestas a colaborar con la investigación.

La encuesta es semejante a la entrevista pero escrita, donde a través de un conjunto de preguntas se pretende obtener una información sobre el mundo interior del encuestado o su percepción del fenómeno que se investiga, por lo que no puede ser obtenida por observación.

2.4 Muestreo Aleatorio Estratificado

Existen pocas áreas donde el impacto del desarrollo de la estadística se haya hecho sentir más que en la ingeniería. La estadística ha venido a ser una herramienta vital para ingenieros, les permite comprender fenómenos sujetos a variaciones y predecirlos o controlarlos eficazmente. En este trabajo se utilizó para determinar la cantidad de población que será encuestada, y el tamaño de la muestra es decir la cantidad de encuestas que se realizarán.

El Muestreo Aleatorio Estratificado consiste en subdividir la población en subpoblaciones de tal forma que la unión de ellos será la población, y la intersección de cualquiera de los dos dará como resultado el conjunto vacío, es decir, no tendrán elementos comunes.

A las subpoblaciones se les llamará estratos, se tratará de conformar estos de modo que los elementos dentro de ellas sean homogéneos. El tamaño de la muestra se distribuirá entre los estratos, en función de distintos criterios pero lo que caracteriza al MAE (Muestreo aleatorio estratificado) es que la selección de la muestra de cada estrato se hará bajo el procedimiento de muestreo irrestricto aleatorio y se realizará independientemente de los diferentes estratos.

Es recomendable estratificar en función del tamaño de las unidades y distribuir la muestra proporcionalmente al número de unidades de los estratos.

Notación

Se supone que la población consta de n unidades y están distribuidas en L estratos, constituyen una partición de la población; se representará por N_h en el n_i de u en el estado h-ésimo, de aquí:

$$N = N_1 + N_2 + \ldots + N_h + \ldots + N_L$$

Total de la población: $X_h = \sum_{1}^{N_h} X_{hi}$ media $\bar{X}_h = \frac{1}{N_h} \sum_{i}^{N_h} X_{hi}$

Fracción de muestreo del estrato: $f_h = \frac{n_h}{N_h}$

Si el tamaño de la muestra de los estratos se distribuye proporcionalmente al número de unidades en el estrato, es decir se cumple que:

$n_h = n\frac{N_h}{N}$ Para todo h. En este caso se dice que la distribución de la muestra se ha hecho con asignación proporcional.

Para la determinación del tamaño de muestra de una población con S^2 desconocida se puede determinar mediante la expresión:

$$n = \frac{\left(\frac{Z_{1-\frac{\alpha}{2}}}{d}\right)^2 P(-P)}{1+\frac{1}{N}\left(\frac{Z_{1-\frac{\alpha}{2}}}{d}\right)^2 p(-P)-\frac{1}{N}}$$

Donde $1-\alpha$ es el nivel de confianza, d es el error absoluto, Z_1 percentil de la distribución normal, P Proporción de la población y N tamaño de la muestra.

Realizando el cálculo del tamaño de muestra para un nivel de confianza $1-\alpha$ = 90%, con un error absoluto d = 0.10, se obtiene el percentil de la distribución normal Z_1 =1.64 y se asume como proporción de la población $P = 0.5$, N = 439 que es la cantidad total de todos los integrantes de los proyectos de la facultad dentro del polo Informática para la Salud, siendo este el polo a encuestar por ser el que presenta mayor aplicaciones Web, pues el otro polo mayormente está basado en el desarrollo de aplicaciones de escritorio, por lo que no es objetivo en esta investigación.

A partir de los datos anteriores se obtiene que:

$$n = \frac{\left(\frac{1.64}{0.10}\right)^2 0.5(1-0.5)}{1+\frac{1}{439}\left(\frac{1.64}{0.10}\right)^2 0.5(1-0.5)-\frac{1}{439}} = \frac{67.24}{0.9977} = 67.39 \approx 68$$

Siendo 68 el número total de encuestas a aplicar.

Conociendo que n = 68, N_h es la cantidad de integrantes en cada proyecto, N = 439 que es el total de integrantes de los diferentes proyectos del polo Informática para la Salud.

Calculando el tamaño de muestra por estrato (se considera estrato a cada Área Temática) aplicando

$n_h = n\frac{N_h}{N}$; resultan los tamaños de muestra por estrato que se muestran en la tabla 2.3

Estratos	**Cantidad**	**Tamaño de la Muestra**
Especialidades	83	13
APS	73	11
Hospitales	117	18
SAS	166	26
Total	439	68

Tabla 2.3 Tamaños de muestra por estratos de la población.

2.5 Diagnóstico de la producción de software en la Facultad 7

2.5.1 Resultados de la encuesta aplicada a los integrantes de los proyectos productivos de la facultad 7

Para comprobar el estado actual de la producción de la facultad 7 en cuanto a la aplicación y desarrollo de métodos de pruebas así como el uso de alguna metodología que guie el proceso de pruebas

durante el ciclo de vida de una aplicación Web, se aplicaron diferentes técnicas para el diagnóstico, entre ellas está la encuesta realizada a los integrantes de los diferentes proyectos productivos de la facultad la cual fue aprobada por la comisión de tesis de la Facultad 7 y el vicedecano de producción de la misma Eduardo Solís Céspedes.(Ver Anexo 1).

La muestra hallada por regiones representa el 15.66; 15.06; 15.38; 15.7; % respectivamente de los integrantes (estudiantes y profesores) de las diferentes áreas temáticas de la facultad. A estos grupos se les aplicó la encuesta relacionada con el proceso de pruebas en la que se recoge información acerca de la aplicación o no de pruebas, las fases, uso de herramientas, funcionamiento del grupo interno de calidad, tipos de pruebas y otros aspectos que permitan diagnosticar el estado actual de la producción de la Facultad 7.

Se decidió segmentarlos por regiones para evitar que los diferentes procesos de desarrollo del software que presentan cada proyecto puedan influir significativamente en los resultados, estratificándose entonces por las distintas regiones.

Los resultados obtenidos revelan cierto desconocimiento en cuanto al tipo de arquitectura que utilizan las aplicaciones y el porqué son usadas, donde solamente un 22.5% de los encuestados mostraron respuestas concretas, mientras un 68.7% mostraba respuestas incoherente o hacían referencia a otro tipo de arquitectura y el 8.8% no supo responder estas preguntas.

Es bueno destacar que el 98.6% de los encuestados manifestaron que se utiliza como metodología de desarrollo RUP. Sin embargo, siendo esta metodología una de las que propone la aplicación de pruebas durante todo el ciclo de vida, solo el 17% de los encuestados reflejaron que en sus proyectos se realizan pruebas y este por ciento corresponde básicamente al área temática de Hospitales y APS. Mientras que el resto de los encuestados reflejan que solo realizan pruebas cuando hacen entregas al proyecto de calidad de la facultad y éstas son solamente de caja negra. Las pruebas de caja blanca que reciben las aplicaciones son solamente las revisiones que realiza el implementador una vez concluido el código. Por tal motivo no se tiene un control de la fase en que se realizan las pruebas, en caso que las realicen, ni la cantidad de veces que deben ser realizadas las pruebas, aunque 97% de los encuestados coinciden en que las pruebas de aceptación son realizadas por el usuario final y solamente un 2.6% considera que es el grupo de calidad de la facultad u otro personal que interactúa con el producto.

Analizando más profundamente en el tema de la realización de las pruebas por fases se pudo constatar que en las fases de inicio y elaboración, donde se realizan el levantamiento de requisitos y

el análisis y diseño respectivamente, no se realizan ningún tipo de pruebas, ni se planifican revisiones a cada artefacto que ha sido generado hasta ese momento. La documentación no se lleva actualizada según crece o se desarrolla el proyecto. En la fase de elaboración, algunos proyectos realizan algunas revisiones informales, pero no se documentan, por lo que se consideran inválidas o nulas. Solo en la fase de construcción es que se puede comprobar que por lo menos en el área temática de hospitales realizan algunas pruebas y tienen noción sobre el uso de herramientas, aunque les falta mucho por madurar aún, teniendo en cuanta que es un avance en la facultad y que por lo menos en una de las áreas temáticas del polo que se encuestó se realizan pruebas. El cúmulo de pruebas en todas las aplicaciones Web que se desarrollan en la Facultad se realizan en la fase de construcción y transición donde el grupo de calidad de la Facultad 7 realiza un grupo de pruebas en varias iteraciones, hasta determinar si el software se encuentra listo o no para ser liberado.

La no realización de pruebas en estas fases trae como consecuencia la presencia de errores que podían detectarse en fases tempranas del desarrollo de la aplicación y se corre el riesgo de no ser detectadas en las revisiones que se realizan al final del ciclo de vida. Por otro lado un cambio en el diseño o alcance de un requisito muchas veces altera el funcionamiento del producto teniendo que ser cambiada una buena parte de este, retrasando el cronograma de entrega y la planificación en sentido general, lo que podría evitarse si se revisaran los requisitos en el momento adecuado.

En cuanto a la calidad de las aplicaciones el 53.8% de los encuestados refleja que es buena, mientras que el 19.2% piensa que es Excelente y solo un 26.9% considera que es regular, esto sin mencionar que ninguno considera que es mala.

Sobre el funcionamiento de los grupos internos de calidad se pudo comprobar que no está creado en todas las áreas temáticas o por lo menos existe gran desconocimiento por parte de sus integrantes donde en APS y Hospitales afirman que existe aunque no funciona aun como debería, mientras que en SAS y Especialidades no conocen de la existencia de un grupo de calidad dentro del área temática.

En el conocimiento y uso de herramientas el 92.8% de los encuestados desconocen herramientas para la ejecución o simulación de pruebas, mientras que el 7.2% conoce algunas herramientas pero no las emplean en sus proyectos. El 63.5% plantea que no usan listas de chequeo en su proyecto mientras que el 56.5% plantea que utilizan alguna lista en algún momento determinado. Por otro lado el 56.4% de los encuestados plantean que en su proyecto se lleva algún registro de defectos detectados durante las pruebas internas que realizan, mientras que el 37.3% afirma que no y un 6.3% desconoce si se lleva a cabo o no este tipo de registro.

Todos los encuestados afirman que es muy importante la revisión de su proyecto por el grupo de calidad de la facultad, para garantizar que el cliente reciba un producto que cumpla con las expectativas del cliente. Afirman además que el trabajo que realiza este grupo es de vital importancia en la terminación de las aplicaciones Web, por la cantidad de no conformidades que se detectan en cada una de las iteraciones en la revisión y aplicación de pruebas.

En sentido general se puede afirmar que en los proyectos productivos no existe un procedimiento que guie el proceso de desarrollo de pruebas durante el ciclo de vida de una aplicación Web con las arquitecturas antes mencionadas y que ocupan el tema de esta investigación. Se refleja además que cada área temática tiene implementado un sistema de control y revisión de sus productos y que los estudiantes vinculados directamente a la producción desconocen los temas fundamentales referentes a las pruebas del software y tienen una visión equivocada sobre el personal que debe desarrollar las mismas así como el momento, en que estas deben ser aplicadas.

A continuación aparecen tabulados los resultados de la encuesta con mayor precisión a través de las siguientes tablas por áreas temáticas.

CBA y SOA	Total	Porciento (%)
si	9	81.8
no	2	18.2

Tabla 2.4 Resultado sobre el tipo de arquitectura a usar en APS.

CBA y SOA	Total	Porciento (%)
si	15	83.3
no	3	16.6

Tabla 2.5 Resultado sobre el tipo de arquitectura a usar en Hospitales.

CBA y SOA	Total	Porciento (%)
si	16	61.5
no	10	38.5

Tabla 2.6 Resultado sobre el tipo de arquitectura a usar en SAS.

CBA y SOA	Total	Porciento (%)
si	9	69.2
no	4	30.7

Tabla 2.7 Resultado sobre el tipo de arquitectura a usar en Especialidades.

CBA y SOA	Total	Porciento (%)
si	49	72
no	19	27.9

Tabla 2.8 Resultado sobre el tipo de arquitectura a usar de forma general.

Tipos de Prueba	Total	Porciento (%)
Caja Blanca	6	54.5
Caja negra	9	81.8
Ninguna	0	0

Tabla 2.9 Resultados de tipos de pruebas aplicadas en APS.

Tipos de Prueba	Total	Porciento (%)
Caja Blanca	18	100
Caja negra	18	100
Ninguna	0	0

Tabla 2.10 Resultados de tipos de pruebas aplicadas en Hospitales.

Tipos de Prueba	Total	Porciento (%)
Caja Blanca	4	15.3
Caja negra	17	65.4
Ninguna	5	19.2

Tabla 2.11 Resultados de tipos de pruebas aplicadas en SAS.

Tipos de Prueba	Total	Porciento (%)
Caja Blanca	4	30.7
Caja negra	8	61.5
Ninguna	2	15.4

Tabla 2.12 Resultados de tipos de pruebas aplicadas en Especialidades.

Grupo Interno de Calidad	Total	Porciento (%)
si	8	72.7
no	3	27.3

Tabla 2.13 Existencia y Funcionamiento del grupo interno de calidad en APS.

Grupo Interno de Calidad	Total	Porciento (%)
si	16	88.8
no	2	11.1

Tabla 2.14 Existencia y Funcionamiento del grupo interno de calidad en Hospitales.

Grupo Interno de Calidad	Total	Porciento (%)
si	5	19.2
no	21	80.8

Tabla 2.15 Existencia y Funcionamiento del grupo interno de calidad en SAS.

Grupo Interno de Calidad	Total	Porciento (%)
si	2	15.4
no	11	84.6

Tabla 2.16 Existencia y Funcionamiento del grupo interno de calidad en Especialidades.

Listas de Chequeo	Total	Porciento (%)
si	5	45.4
no	6	54.5

Tabla 2.17 Uso de algún tipo de listas de chequeo en APS.

Listas de Chequeo	Total	Porciento (%)
si	15	83.3
no	3	16.6

Tabla 2.18 Uso de algún tipo de listas de chequeo en Hospitales.

Listas de Chequeo	Total	Porciento (%)
si	2	7.69
no	24	92.3

Tabla 2.19 Uso de algún tipo de listas de chequeo en SAS.

Listas de Chequeo	Total	Porciento (%)
si	7	53.8
no	6	46.1

Tabla 2.20 Uso de algún tipo de listas de chequeo en Especialidades.

Listas de Chequeo	Total	Porciento (%)
si	29	42.6

no	39	57.3

Tabla 2.21 Uso de algún tipo de listas de chequeo de Forma General.

2.5.2 Antecedentes del proceso de pruebas a aplicaciones Web en la Facultad 7

Desde el surgimiento del grupo de calidad en la Facultad 7, se han tenido experiencias con diferentes productos que han estado en fase de liberación. Entre estos productos cabe citar el Sistema de Gestión de Información para la Planificación y Balance Material en el MINSAP perteneciente al Área temática de Sistemas de Atención para la Salud, donde los requisitos desde un comienzo no fueron revisados y verificados antes de pasar a otras fases, ocasionando que en la actualidad se efectúen cambios por parte del cliente por ambigüedad en la escritura de algunos de ellos.

Por otro lado la elaboración y revisión del manual de usuario y el resto de la documentación de dicha aplicación no ha llevado el tiempo necesario, por lo que ha llegado a la fase de liberación por el grupo de calidad de la facultad con insuficiencias en el contenido y poca coherencia con la realidad de las aplicaciones, así como errores ortográfico y gramaticales, que afectan la calidad general tanto de la documentación como de la aplicación. Esto demuestra que no funciona el grupo interno de calidad del área temática y que no existen revisiones previas antes de liberar el producto.

Otra de las aplicaciones que ha sido revisada por el grupo de calidad es el Sistema de Información Estadística Complementario de Salud y el Sistema para la Formación de Recursos Humanos en las Secretarias Docentes del MINSAP donde se manifiestan algunos de los errores antes mencionados relacionados con la documentación además de presentar errores en la copia de la base de datos que utilizan proporcionadas por SOFTEL llamada RPS (Registro del Personal de Salud) la cual posee errores que no se pueden ser corregidos por estos proyectos , por ser solamente un componente brindado por esta entidad. Lo mismo pasa con el proyecto de Colaboración Médica sumando errores de diseño y la no correspondencia entre módulos del mismo proyecto que al integrarse formaran parte de un único sistema.

Todos estos defectos y no conformidades detectadas en la etapa de liberación podían haber sido depuradas en otras fases del ciclo de vida de la aplicación, si funcionara el grupo interno de calidad, como está establecido en cada una de las Áreas Temáticas de la facultad.

Además, los juegos de datos con que pasan las aplicaciones a la etapa de revisión son muy pobres e insuficientes lo que dificulta el proceso de pruebas. Otro de los aspectos que se descuidan un poco es

la validación de los campos que en muchas ocasiones permiten la entrada de cualquier valor cuando deben estar restringidos para un cierto rango de caracteres.

La mala descripción de los casos de uso es otro de los aspectos que influyen en el desarrollo de la aplicación de pruebas, así como en el diseño de las mismas, pues no son revisadas una vez surgen cada uno de los diagramas correspondientes y esto trae consigo una mala implementación del producto.

Estas deficiencias fueron detectadas también en el Bloque Quirúrgico Oftalmológico perteneciente al Área Temática de Hospitales, donde hasta el momento se ha liberado la documentación del mismo. Además de presentarse también en uno los proyectos del área temática de Hospitales que han sido revisados hasta el momento.

A pesar de que esta investigación se centra en las aplicaciones con Arquitecturas SOA-CBA que es el tipo de arquitectura que predomina dentro de la Facultad 7, es bueno destacar que en las revisiones realizadas a los productos de los proyectos pertenecientes al área temática de Procesamiento de Imágenes y Señales que son básicamente aplicaciones de escritorio, se han detectado este tipo de no conformidades, aunque este grupo de desarrollo está mejor organizado y sí realizan pruebas previas y llevan el registro de defectos en el expediente del proyecto, pero aun les falta mucho trabajo por desarrollar aunque es un ejemplo para el resto de las áreas temáticas.

De forma general el tema ortografía es un aspecto que golpea a la mayoría de las aplicaciones y documentación que cada proyecto elabora y realiza la entrega al grupo de calidad de la facultad. Donde en ocasiones a este grupo le resulta mas fácil arreglar cualquier error que aparezca, que redactar la no conformidad correspondiente a este error, incluyendo el camino y foto en el anexo.

El funcionamiento del grupo interno de calidad de cada área temática tiene que ser inmediato, para lograr un incremento en la producción, teniendo basada una buena planificación y revisión de cada una de las etapas del ciclo de vida por las que atraviesa una aplicación determinada. Este grupo debe ir ganando espacio en cada proyecto y formando parte de los mismos de forma tal que este pendiente de posibles errores que puedan ocurrir.

Después de haber diagnosticado y caracterizado la producción de la Facultad 7 en cuanto al desarrollo de aplicaciones Web; se puede concluir que:

- La estructura de la producción en la Facultad 7 mejora cada día más y tiende a la eficiencia y productividad dada su organización en cada uno de los puestos de trabajo.
- No existe aún un procedimiento que guie el proceso de desarrollo de las pruebas en la Facultad 7.razon por la cual cada Área Temática realiza sus actividades independientes lo que dificulta el trabajo de liberación de los productos
- No se realizan pruebas de caja blanca en la mayoría de los grupos de desarrollo, propiciando que los productos pasen a la etapa de prueba con errores y sin validaciones imprescindibles. Tampoco se emplean listas de chequeos para la revisión de cada artefacto que se va generando en las distintas fases del desarrollo del software.
- No se usan herramientas para el desarrollo y simulación de pruebas y existe poco conocimiento sobre el uso de las mismas.
- El grupo interno de calidad no está creado en todas las áreas temáticas y en las que existe no funciona correctamente.
- Existe poco intercambio con los clientes y vinculación con los mismos así como la firma de los requisitos en la fase inicial, lo que provoca frecuentes cambios en fases posteriores, implicando cambios en la interfaz documentación, diagramas y atrasándose el cronograma de entrega del producto final.

Capitulo 3 Propuesta de Procedimiento.

3.1 Nombre del Procedimiento:

Macro-procedimiento para la aplicación de pruebas en cada fase del ciclo de vida de un producto Web.

3.2 Objetivo:

Normalizar el proceso del desarrollo y aplicación de pruebas en todas las aplicaciones Web desarrolladas en la facultad 7.

3.3 Alcance:

Todas las aplicaciones Web desarrolladas en la Facultad 7 de la UCI.

3.4 Referencia:

Procedimientos para la elaboración de los Procedimientos y el Manual de la Universidad de las Ciencias Informáticas. (82)

3.5 Responsable:

Ejecuta: Jefes de Áreas Temáticas de la facultad 7, Lideres de proyecto y personal designado en los grupos de desarrollo, grupo interno de calidad y grupo de calidad de la facultad.

Responsable de su ejecución: Vicedecano de producción de la Facultad 7 y Asesor(a) de calidad de esta facultad.

Revisa y actualiza este procedimiento: Asesor(a) de calidad y jefes de los polos productivos de la Facultad 7 y personal designado para tal actividad.

Fiscaliza su cumplimiento: Asesor(a) de calidad y Vicedecano de producción de conjunto con los lideres de proyecto y jefes de área temática y jefes de los polos productivos, con especialistas en temas de calidad del software.

3.6 Términos y Definiciones:

CBA: Arquitectura Basada en Componentes

Pruebas de aceptación: La prueba de aceptación es realizada por un grupo de usuarios finales o los clientes del sistema, para asegurarse que el sistema desarrollado cumple sus requisitos. La prueba de

aceptación de usuario se distingue generalmente por la incorporación de un trayecto feliz o casos de prueba positivos. (83)

Pruebas de Liberación: Conjunto de pruebas y revisiones desarrolladas por el grupo de calidad de la facultad y posteriormente por el equipo de calidad central de la universidad para garantizar la satisfacción del cliente

Requisitos del Negocio: Todas las tareas relacionadas con la determinación de las necesidades o de las condiciones a satisfacer para un software nuevo o modificado, tomando en cuenta los diversos requerimientos de los inversores, que pueden entrar en conflicto entre ellos. Puede ser conocida también como "Análisis de requerimientos", "especificación de requerimientos", etc.

SOA: Arquitectura Orientada en Servicios.

UCI: Universidad de la Ciencias Informáticas

Usuario Final: Persona o personas que van a manipular de manera directa un producto de software, no es necesariamente sinónimo de cliente o comprador, una empresa puede ser un importante comprador de software, pero el utilizador final puede ser solamente un empleado o grupo de empleados dentro de la compañía

3.7 Disposiciones Generales:

3.7.1 Este procedimiento entrará en vigencia a partir del curso 2008-2009 luego de ser analizado y aprobado por el tribunal de tesis que evaluará esta investigación.

3.7.2 Debe ser analizado y discutido en la dirección de la producción de la facultad para aprobar su puesta en marcha en cada una de las áreas temáticas de la facultad. Como constancia será tomado como acuerdo en el consejo de producción de la Facultad 7.

3.7.3 Se recomienda realizar un estudio más profundo por cada una de las fases de este procedimiento con el objetivo de normalizar y establecer paso a paso cada una de las actividades y responsables de estas en cada proceso de prueba que aquí se menciona.

3.7.4 Este documento contiene información propietaria de la Universidad de las Ciencias Informáticas y fue elaborado confidencialmente para un propósito específico.

3.7.5 El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

3.7.6 Estas reglas son aplicables para todas las páginas de este documento.

3.8 Desarrollo del Procedimiento.

3.8.1 Introducción.

En la actualidad existen muchas formas de detectar defectos y errores en la producción de software, por otro lado es cierto que las personas cometen errores en la medida en que se trabaje más tiempo, se asume la responsabilidad de tareas complejas, además del cansancio o la distracción de cada uno de los integrantes del grupo de desarrollo.

En el proceso de elaboración de software se requiere atención, precisión, comunicación, comprensión y creación a lo largo del tiempo de vida, inevitablemente se introducen defectos, que pueden ser detectados y corregidos para que el producto llegue al usuario final lo mas cercano posible a sus requisitos

El principal objetivo de realizar pruebas desde la concepción del producto es prevenir posibles fallos que pueden ser planificados con anticipación, por experiencia en otras aplicaciones que coinciden, pues el probador desde afuera tiene una mejor visión, y diferente forma de pensar a los desarrolladores que se encuentran dentro del grupo de producción.

Después de realizar un estudio de los procedimientos de pruebas que abarquen el ciclo de vida de un producto determinado, se considera adecuado adaptar el modelo V de pruebas, con una versión mejor refinada, en el modelo W en las aplicaciones Web que se desarrollan en la facultad 7 de la UCI. Estos modelos fueron analizados en el capitulo 1 de esta investigación.

El modelo W se puede adaptar a cualquier metodología de desarrollo de software, por lo que es válida su aplicación dentro del Proceso Unificado de Desarrollo de Software (RUP) que es uno de los procesos más usados dentro de la facultad 7, en el desarrollo de aplicaciones Web con las arquitecturas antes mencionadas.

Según Pressman y otros grandes de la Ingeniería de software, plantean que el proceso de prueba se desarrolla en espiral desde adentro hacia afuera, de lo mínimo a lo mas extenso. Donde primero se

deben realizar las pruebas de unidad, luego pruebas de integración y finalmente las pruebas al sistema global como un todo.

Por otro lado queda demostrado que las pruebas deben planificarse con antelación para garantizar el desarrollo exitoso de la aplicación, mediante un plan de pruebas que se debe comenzar desde el inicio del ciclo de vida, con algunas revisiones previas en cada una de las fases.

Sin alejarnos de estas metodologías y siguiendo como espina dorsal el modelo antes mencionado se propone desarrollar el siguiente procedimiento en el ciclo de vida de las aplicaciones Web.

3.8.2 Fase Requisitos del Negocio**.**

En la fase inicial de una aplicación Web, donde se definen los requisitos de la misma, en presencia de los clientes, es necesario verificar si cada requerimiento está expresado de manera suficientemente precisa como para poder elaborar casos de prueba que permitan indicar si se satisface el requerimiento o no.

En general, el hecho que los requerimientos estén escritos en lenguaje informal siempre es fuente de imprecisiones y ambigüedades; por otro lado, en la práctica es inusual encontrarse con requerimientos escritos en lenguaje formal, por lo que es necesario partir de lo que se tiene, a pesar que sería mucho más fácil elaborar casos de prueba a partir de especificaciones formales que de especificaciones informales. (84)

Lo primero que hay que lograr es que los requisitos estén definidos de una forma formal con un lenguaje acorde con el tema a desarrollar, para evitar suposiciones, de que un requisito significa una cosa u otra.

El objetivo de las pruebas de verificación es buscar discrepancias entre los requerimientos y la ejecución del software.

3.8.2.1 Procedimiento.

El proceso de verificación de los requerimientos comienza con el análisis de esos requerimientos y una inspección en la cual se busca evaluar la consistencia, completitud y factibilidad de los requerimientos, tanto individualmente como juntos. Adicionalmente los requerimientos deben ser revisados y validados por los distintos actores involucrados con el sistema (*stakeholders),* acción que debe aclarar los compromisos al respecto, tanto en el sentido de prioridades y balance entre requerimientos como en el sentido de compromisos que asumen los actores. (85)

Para evitar sorpresas de variada índole a la hora de entregar el software, es conveniente especificar claramente qué se va a hacer para determinar que el sistema satisface sus requerimientos. Por ejemplo, no basta con decir que un sistema será amigable o fácil de usar, ¿cómo se medirá o verificará que el software es amigable?

El primer paso es Revisar la verificabilidad del requerimiento, para evitar lo arriba descrito y demostrar que los requerimientos que definen el sistema son lo que el cliente realmente quiere. Los costos de errores en los requerimientos son altos, por lo cual, la validación es muy importante**.** Luego especificar el criterio de verificación, es decir, aclarar bien la validez o invalidez de una entrada o función determinada, así como el tiempo de respuesta del sistema a la hora de solicitar un servicio determinado. Después hacer visible las propiedades o elementos del software para verificar el cumplimiento del requerimiento, en este caso se describe detalladamente como debe ser mostrado por la aplicación el servicio, especificando el lugar. Hacer controlable los elementos del software necesario para llevar a cabo las pruebas, donde se tendrán en cuenta la forma en que se mostraran los mensajes de error a la hora de solicitar un servicio determinado, teniendo en cuenta el criterio del cliente, además del lenguaje y palabras claves que serán empleadas en la aplicación.

De forma general se propone confeccionar una lista de chequeo donde se reflejen, paso a paso cada uno de los elementos anteriormente descritos, para una vez realizado el levantamiento de requisitos, que la misma sea aplicada por el grupo interno de calidad que existe en cada área temática de la facultad 7 de la UCI.

Este tipo de revisión puede ser formal con toda la documentación existente hasta el momento y con la participación de los clientes y el grupo de desarrollo, incluyendo analistas, desarrolladores y algunos miembros del grupo interno de calidad del área temática donde se desarrollara la aplicación. O de forma informal, solamente con miembros del grupo de calidad del área temática.

Para ambas formas de realizar la revisión debe ser llenada la planilla de no conformidades, vigente en ese momento, aprobado por la dirección de Calidad Central de la UCI.

En el caso que la revisión se efectúe de forma informal y las no conformidades detectadas afecten el futuro desarrollo de la aplicación Web, debe ser convocada una nueva reunión con los clientes para corregir las mismas y lograr que los requisitos cumplan las características que aquí se describen, pues una buena comunicación entre desarrolladores, clientes y usuarios puede resolver problemas en las primeras etapas.

Para la confección de la lista de chequeo pueden emplearse algunas preguntas que respondan a términos específicos como se muestra a continuación:

1. Validación. ¿Provee al sistema las funciones que mejor soporten las necesidades del cliente?
2. Consistencia. ¿Existe cualquier conflicto en los requerimientos?
3. Completo. ¿Están incluidas todas las funciones requeridas por el cliente?
4. Realismo. ¿Pueden los requerimientos ser implementados con la tecnología y el tiempo disponible?
5. Verificabilidad. ¿Es el Requerimiento realmente probable?
6. Entendibilidad. ¿Es el Requerimiento comprendido propiamente?
7. Probabilidad. ¿Es el origen de los requerimientos claramente establecido?
8. Adaptabilidad. ¿Puede el requerimiento ser cambiado sin causar un gran impacto en otros requerimientos?
9. Probabilidades ¿Está planificado que ocurran posibles cambios en los requerimientos cuando el sistema sea desarrollado y utilizado?

No obstante en el laboratorio de calidad de la facultad 7 existe una lista estándar para aplicar a los requisitos, la cual lleva por nombre “Lista de Comprobación para Requisitos”, siendo esta la que deben tener cada uno de los proyectos productivos de la facultad.

3.8.2.2 Plan de Pruebas de Aceptación.

Una vez concluido el proceso de revisión de los requerimientos, es sumamente importante que se comience a elaborar el plan de de pruebas de aceptación del cliente, las cuales verificaran que el sistema satisface los requerimientos propuestos por este. Estas pruebas son similares a las pruebas de sistema, pues se basan fundamentalmente en pruebas de funcionalidad.

A pesar de que es la ultima prueba que se aplica al producto, es de vital importancia su planificación en esta etapa del desarrollo de la aplicación, además es un trabajo que se tiene adelantado para la ultima fase del ciclo de vida.

3.8.2.2.1 Pruebas de aceptación.

Son básicamente pruebas funcionales realizadas por el cliente, sobre la aplicación completa, y buscan una cobertura de la especificación de requisitos y del manual del usuario. No se realizan durante el desarrollo, sino que se realizan sobre el producto terminado e integrado o pudiera ser una versión del producto o una iteración funcionad pactada previamente con el cliente y definido en el plan de pruebas de aceptación, al inicio del ciclo de vida.

La experiencia muestra que aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente comienza a usarlo, y en la actualidad todo el esfuerzo va encaminado en la satisfacción del cliente, por lo que al final este siempre tiene la razón.

No vale de nada comenzar a justificar lo mal hecho, cuando se pudo evitar en la revisión de requisitos, expresando, que estos no estaban claros, o que el manual es ambiguo, o simplemente culpar al cliente por no se todo lo exacto en la definición.

3.8.2.2.2 Procedimiento.

La primera actividad que se planifica en este plan, es la descripción de los tipos de pruebas de aceptación que el cliente realizará, así como el momento dentro del ciclo de vida en que será aplicado, pues puede acordarse de que en una primera versión completa del producto, el cliente realice un conjunto de pruebas de aceptación, para detectar fallas, que, podrán corregirse en una próxima iteración.

En cualquier caso debe planificarse el tiempo que llevará la realización de las pruebas de aceptación, en el caso de una primera versión no debe exceder una semana, no así en la ultima iteración donde puede llevar semanas o meses de pruebas, es decir, se define un tiempo de prueba como garantía de la aplicación que comienza a funcionar en un entorno real.

Por su parte en todo este período de prueba se tendrá constancia escrita de los defectos que se van detectando durante la planificación. (86)

Para este tipo de pruebas se emplean dos técnicas:

La prueba alfa.

Se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario. Las pruebas alfa se llevan a cabo en un entorno

controlado. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados. (87)

Este tipo de prueba será desarrollado en un laboratorio de la facultad 7, donde la aplicación esté publicada y simule un entorno real. En la misma estarán presentes a parte de los clientes, miembros del grupo de desarrollo y miembros del grupo de calidad interno del área temática, además de algún miembro del grupo de calidad de la facultad.

La prueba beta.

Se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación "en vivo" del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador. (88)

Como resultado de los problemas informados durante la prueba beta, el desarrollador del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda la clase de clientes.

En caso de que el cliente sea miembro de una entidad extranjera, o nacional pero que no pueda realizar las visitas a la Universidad de la Ciencias Informáticas, para desarrollar el tipo de prueba Alfa, se procederá a preparar con antelación a algunos miembros del grupo de desarrollo, o del área temática, que pueden ser del grupo interno de calidad a simular un cliente real. De lo contrario el grupo de calidad de la facultad debe asumir esta responsabilidad con un previo conocimiento del funcionamiento real de la aplicación, así como de todos sus servicios y funcionalidades. De esta forma el grupo de calidad de la facultad será el encargado de diseñar, guiar y dirigir las pruebas de aceptación, que en este caso pasarían a ser pruebas de liberación del producto, siendo estas pruebas un servicio que brinda el laboratorio de calidad de la facultad 7 a las aplicaciones Web que se desarrollen en dicha facultad.

Finalmente, antes de ser entregado al cliente final, dicha aplicación tendrá que ser revisada por el grupo central de calidad de la UCI, tomando esta como otra prueba de aceptación o de liberación en dependencia de las circunstancias y el tipo de aplicación que se encuentre en pruebas en ese momento, con toda la documentación y procesos, que se requieren.

3.8.3 Fase Especificación Formal (Análisis y Diseño).

En esta fase dentro del ciclo de vida de la aplicación es donde se comienza la puesta en marcha de todos los requisitos que el usuario solicitó y que serán convertidos en un producto, que sea capaz de cumplir cada una de estas especificaciones.

Además es donde surge un prototipo de interfaz de usuario y se genera toda la documentación y diagramas pertinentes, para dar paso a la fase de implantación. Es por ello que esta etapa debe ser revisada para evitar se cometan errores en etapas posteriores y por otra parte es donde se planifican las pruebas que se realizaran al sistema una vez este integrado.

3.8.3.1 Procedimiento.

Una vez realizada la descripción de todos los casos de uso, con sus flujos normales y alternos, estos deben ser revisados por un miembro del grupo interno de calidad que posea buenos conocimiento sobre la ingeniería de software. Donde se verificará que el caso de uso realiza la actividad que el cliente especificó a través de los requerimientos que se levantaron en la primera fase del ciclo de vida, así como su correspondencia con uno o varios de estos.

De igual forma se procede cuando se generen los diagramas correspondientes a los casos de uso, donde a parte de los posibles errores, y prevención de los mismos, se tendrá otro punto de vista a la hora de la puesta en marcha del sistema. Para estas pruebas se propone que sea un miembro del grupo de calidad del proyecto pues cada revisión siempre es prudente que la realiza alguien desde fuera, pues no cumple objetivo que los mismos analistas que diseñaron realicen la revisión, pues para ellos siempre estará bien lo que hicieron.

Otra actividad que se debe planificar en esta etapa es el diseño de casos de prueba, a través de la descripción de los casos de uso. Tarea que deben realizar los miembros del grupo interno de calidad del área temática al que pertenece la aplicación.

Este diseño será un adelanto para la etapa de pruebas de liberación que tendrán lugar al final del ciclo de vida de la aplicación por parte del equipo de calidad de la Facultad 7.

Por otro lado se debe chequear o revisar toda la documentación que se haya generado hasta el momento, incluyendo glosario de términos, documento visión y demás documentos generados.

Para este tipo de revisión existen un grupo de listas de chequeo que se encuentran en la dirección central de Calidad de la UCI y en el laboratorio de calidad de la Facultad 7. De este grupo de listas de chequeo, se propone aplicar las siguientes:

1. Lista de Comprobación para Modelo de caso de Uso de Negocio
2. Lista de Comprobación para Análisis y Diseño
3. Lista de Chequeo Casos de Uso
4. Lista de Chequeo para la especificación de casos de usos
5. Listas de Chequeo de Diagramas de Clases

Todas estas revisiones deben quedar registradas en un documento o expediente de proyecto que será actualizado en cada proceso de prueba que se desarrolle, además de ser la guía de los analistas y mas tarde de los desarrolladores para la depuración de los defectos encontrados.

Según se van creando los prototipos de interfaz de usuarios estos deben ser revisados, para verificar si cumplen con los estándares de diseño internacionales, además de los colores, fuentes, tipos de letras, etc. Para este tipo de revisión se propone la aplicación de una lista de chequeo que defina claramente que es lo que debe tener la aplicación y que sea estándar para todos los proyectos de la facultad 7, donde abarque todos estos términos de diseño.

Por ejemplo se pueden incluir algunas preguntas como las siguientes:

- ¿Está diseñada la interfaz para facilitar la realización eficiente de las tareas de la mejor forma posible?
- ¿Permite una cómoda navegación dentro del producto y una fácil salida de éste?
- ¿Permite distintos niveles de acceso en dependencia de las funcionalidades de la aplicacion?
- ¿Se corresponden los colores empleados con el destino de la aplicación?
- ¿Se emplea un solo idioma en toda la interfaz?

Estas son sólo algunas de las preguntas que pueden incluirse en la lista de chequeo, que se determine para esta fase dentro del ciclo de vida.

Una vez concluida la primera iteración con la aparición de un prototipo de interfaz, este debe ser mostrado al cliente, para de esta forma comprobar que se ajusta a sus requerimientos y evitar que luego quiera realizar cambios una vez esté aprobado por él mismo un diseño.

Si el usuario detecta inconformidades con este prototipo inicial los desarrolladores valoran los cambios y presentan una nueva versión hasta que el usuario quede satisfecho. La figura 1.3 muestra el ciclo de evaluación de la interfaz.

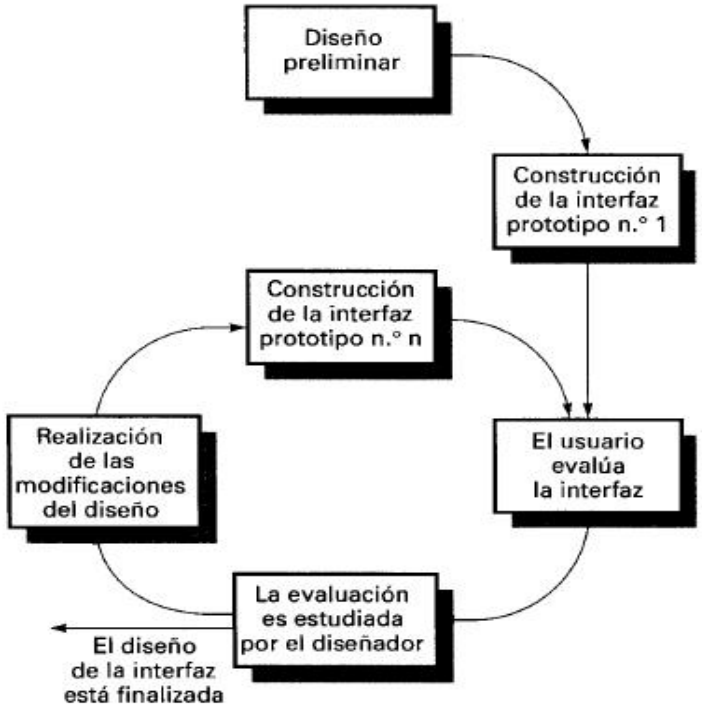

Figura 3.1 El ciclo de evaluación de diseño de la interfaz **(89)**

Al finalizar todo el proceso de revisión de la documentación, diagramas y prototipos de interfaz se procede a planificar las pruebas que se le realizaran al sistema una vez integrado todos los componentes.

3.8.3.2 Plan de pruebas del Sistema.

Los tipos de pruebas que se le aplicarán al sistema van enfocados a comprobar la correcta prestación de los servicios que debe brindar cualquier aplicación Web con una Arquitectura Orientada a Servicios.

Es el proceso de demostrar que la aplicación hace lo que debe de hacer de acuerdo a las especificaciones de los requerimientos. Asegurarse que el sistema cumple con las metas y objetivos propuestos.

Es en esta fase dentro del ciclo de vida donde se definen cuales serán las pruebas que se aplicarán al producto, y en que momento, con previo acuerdo con el equipo de calidad de la Facultad 7. Entrando esta actividad dentro de la planificación general del proyecto, para evitar improvisaciones a la hora de realizar un tipo de prueba. Por otro lado gana en organización del trabajo y en la prevención de errores, además de tener en cuenta la planificación del tiempo a la hora de ejecutar estas pruebas. De esta forma toda actividad que se realice, estará regida por un cronograma de trabajo.

3.8.3.2.1 Procedimiento.

La primera actividad dentro de la planificación de pruebas del sistema, es determinar los tipos de pruebas que se van a realizar. En este caso se proponen inicialmente 4 pruebas fundamentales, que son las más usadas internacionalmente para aplicaciones Web con Arquitectura Orientada a Servicios. Estas son:

- Pruebas de Carga
- Pruebas de Volumen
- Pruebas de Estabilidad
- Pruebas de Seguridad

Estas pruebas, cuyo objetivos y definiciones se abordaron en el capitulo 1, serán ejecutadas antes de realizar las pruebas de liberación del producto. Por lo que una vez concluida la primera iteración completa de la aplicación, debe coordinarse la primera entrada al laboratorio de calidad para la realización de las mismas, como parte de las primeras pruebas de caja negra que este equipo desarrollará.

Pero antes de ejecutar estas pruebas, los miembros del grupo interno de calidad de los proyectos productivos deben comprobar cada uno de los requisitos contra la aplicación, para verificar, que realmente el sistema cumple todas las condiciones puestas por el cliente.

De forma general se planificaría la ejecución de la primera fase del diseño de casos de pruebas, que fueron diseñadas previamente de conjunto con la descripción de cada caso de uso del sistema.

Se habla de primera fase pues, este tipo de prueba se realizará de forma oficial en una segunda fase por el equipo de calidad de la facultad en las pruebas de liberación, al final del ciclo de vida.

Para esta prueba debe llevarse un registro de los defectos encontrados que será adjuntado al expediente del proyecto que posteriormente será entregado al grupo de calidad de la facultad 7.

3.9 Fase Diseño de la Arquitectura (Análisis y Diseño).

En esta fase dentro del ciclo de vida de la aplicación Web se procederá a revisar el diseño de la arquitectura.

3.9.1 Procedimiento.

Para la revisión del diseño de la arquitectura hay que tener en cuenta varios elementos de peso, y que son los que se verificarán aquí para a la hora de integrar el sistema, todo marche bien.

Dentro de la arquitectura se revisa la navegabilidad de la aplicación, la división por módulos y la dependencia entre ellos. Estas revisiones son ejecutadas dentro del grupo de desarrollo por el grupo interno de calidad.

Se revisará además el número de páginas diseñadas, así como su acceso desde otras, definiendo un camino o ruta a seguir, o si desde una se puede llegar al resto. Además del acceso por niveles de restricción o permisos de usuarios.

Para ello se propone la aplicación de listas de chequeo que serán elaboradas en cada una de las áreas temáticas, en dependencia de las necesidades de cada una de ellas. Estas listas de chequeo pueden ser enriquecidas por otras listas estándares que existen en la dirección central de calidad de la Universidad.

Todas estas revisiones serán anexadas al expediente de proyecto, donde se lleva el control de todas las no conformidades detectadas en cada una de las fases por las que pasa el producto.

El equipo de desarrollo evaluará cada una de estas no conformidades y procederá a la depuración de las mismas.

3.9.2 Plan de pruebas del Subsistema.

En esta fase se planifican las pruebas que se le realizarán al subsistema, en este caso serían las pruebas de integración que son pruebas que se dividen en varias categorías, estas son: (90)

- Pruebas basadas en hilos: se Integra el conjunto de clases necesarias para responder a una entrada o evento del sistema (si se hizo un análisis de casos de uso, los datos para la prueba pueden recopilarse antes)
- Pruebas basadas en usos de clases: Se prueban primero las clases independientes (las que usan muy pocas o ninguna de las clases servidor) y luego se prueban las clases dependientes).
- Pruebas basadas en escenarios. Se concentran en lo que el usuario hace, no en lo que hace el sistema, es decir, se prueba la secuencia de eventos de un caso de uso.
- Pruebas aleatorias: Para cada clase cliente, usar la lista de operadores para generar una serie de secuencias de pruebas aleatorias, de modo que para cada mensaje que se genera, se determina la clase colaboradora y el operador correspondiente en el objeto servidor, se determinan los mensajes que el objeto servidor trasmite y se incorporan a la secuencia de prueba el próximo nivel de operadores.

Lo primero que hay que planificar es en que forma se llevara a cabo la integración del sistema. Las mismas pueden ser:

- Incremental
- No incremental
- Ascendente
- Descendente.

En la integración no incremental se combinan todos los módulos por anticipado. Lo que lleva al caos. Este tipo de integración no es muy recomendada, pues hay que esperar el momento adecuado, para realizar las mismas. (91)

Aquí se propone realizar la integración incremental, ya que se construye y se prueba en pequeños segmentos. Dentro de estas pruebas entran las formas ascendentes y descendentes, donde en la forma descendente se integran los módulos moviéndose hacia abajo por la jerarquía de control. Comienzo con el módulo principal e incorporo módulos en la forma: (92)

  - primero en profundidad

- primero en anchura.

Se utilizan módulos de resguardo. Estos módulos son de seguridad o garantía en cuanto a la dependencia entre ellos y el acceso a datos ó información dependiente de otro modulo específico, para que de esta forma fluya la información.

En la integración ascendente, se integran los módulos desde el fondo de la jerarquía hacia arriba. Se comienza con los módulos atómicos y se construyen constructores que luego se reemplazan.

En dependencia del tipo de aplicación y su complejidad se determinará que tipo de integración se empleará, para ello se muestran las ventajas y desventajas de cada una de estas. (93)

Integración descendente:

- (D) necesidad de resguardos
- (V) probar de antemano las principales funciones de control

Integración ascendente:

- (D) el programa como unidad no existe hasta que se haya añadido el último módulo.
- (V) mayor facilidad para el diseño de casos de prueba y falta de resguardos.

Teniendo definido el método de integración y la forma, se planificara en que momento dentro del ciclo de vida se integrarán cada uno de estos componentes.

Con previa coordinación con el grupo de calidad de la Facultad 7, se puede establecer la revisión por separado de los módulos, en dependencia de la funcionalidad y complejidad de la aplicación, antes de ser integrados, tomándose los mismos como un sistema global

Este proceso se realizaría, después de probar la unidad como tal, a través de las pruebas de caja blanca, de las que se hablarán en la próxima fase. Cuando esta unidad esté libre de errores se procedería entonces a realizar la integración.

Si la aplicación carece de sentido, o funcionalidad con la ausencia de uno de estos componentes o módulos, la integración se realiza en el equipo de desarrollo y se prueba si funciona correctamente con las pruebas antes descrita.

Luego de probar cada unidad por separado se comenzarán a integrar por la vía escogida, verificando que el sistema funciona correctamente según se le van incorporando módulos o componentes, hasta lograr tener el sistema como un todo.

3.10 Fase de Implementación.

3.10.1 Planificación y Ejecución de las Pruebas de Unidad.

Al inicio de esta fase se debe planificar una revisión completa del diseño de la aplicación, por parte de los integrantes del grupo de calidad interna del área temática a la que pertenece la aplicación.

Además de la planificación de las pruebas de Unidad que se desarrollarán al final de la fase como parte de las pruebas de caja blanca, las que serán desarrolladas según el cronograma y la planificación de entrega de cada módulo. Estas pruebas permiten determinar si un módulo del programa está listo y correctamente terminado, y no se deben confundir con las pruebas informales que realiza el programador mientras está desarrollando el módulo. El objetivo fundamental de las pruebas unitarias es asegurar el correcto funcionamiento de las interfaces, o flujo de datos entre componentes.

Las pruebas de caja blanca que se proponen, sean aplicadas en la facultad son las siguientes:

- Pruebas del camino Básico
- Pruebas de Complejidad Ciclomática
- Pruebas de Bucle

De forma general estos tipos de pruebas fueron abordados en el capitulo 1 de esta investigación. A continuación una descripción global de cómo realizar las pruebas unitarias.

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. (94)

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.

Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del modulo. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. Además de las estructuras de datos locales, durante la prueba de unidad se debe comprobar (en la medida de lo posible) el impacto de los datos globales sobre el módulo.

Durante la prueba de unidad, la comprobación selectiva de los caminos de ejecución es una tarea esencial. Se deben diseñar casos de prueba para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados. Las pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos. (95)

3.10.2 Procedimiento.

Debido a que un componente no es un programa independiente, se debe desarrollar para cada prueba de unidad un software que controle y/o resguarde. En la mayoría de las aplicaciones, un ***controlador*** no es más que un programa principal que acepta los datos del caso de prueba, pasa estos datos al módulo (a ser probado) e imprime los resultados importantes. Los resguardos sirven para reemplazar módulos que están subordinados (llamados por) el componente que hay que probar. Un resguardo o un subprograma simulado usa la interfaz del módulo subordinado, lleva a cabo una mínima manipulación de datos, imprime una verificación de entrada y devuelve control al módulo de prueba que lo invocó. (96)

Los controladores y los resguardos son una sobrecarga de trabajo. Es decir, ambos son software que deben desarrollarse (normalmente no se aplica un diseño formal) pero que no se entrega con el producto de software final. Si los controladores y resguardos son sencillos, el trabajo adicional es relativamente pequeño. (97)

Desgraciadamente, muchos componentes no pueden tener una adecuada prueba unitaria con un sencillo software adicional. En tales casos, la prueba completa se pospone hasta que se llegue al paso de prueba de integración (donde también se usan controladores o resguardos)
La prueba de unidad se simplifica cuando se diseña un módulo con un alto grado de cohesión Cuando un módulo solo realiza una función se reduce el número de casos de prueba y los errores se pueden predecir y descubrir mas fácilmente. (98)

El principal factor que se debe considerar al inicio de las pruebas es el tamaño del módulo a probar, se debe considerar si el tamaño del módulo permitirá probar adecuadamente toda su funcionalidad de manera sencilla y rápida. También es importante separar los módulos de acuerdo a su funcionalidad, si

los módulos son muy grandes y contienen muchas funcionalidades, estos se volverán más complejos de probar y al encontrar algún error será más difícil ubicar la funcionalidad defectuosa y corregirla. Al hacer esta labor el analista de pruebas o el designado a realizar este tipo de pruebas dentro del grupo de desarrollo, o del grupo interno de calidad del área temática podrá recomendar que un modulo muy complejo sea separado en 2 o 3 módulos más sencillos.

Este tipo de pruebas debe ser realizado por personal especializado en pruebas de este tipo, para lo cual se necesita de una capacitación previa, antes de comenzar con la ejecución de las mismas, pues este personal debe estar familiarizado en el uso de herramientas de depuración y pruebas, así mismo deben conocer el lenguaje de programación en el que se está desarrollando la aplicación, en la actualidad existen una gran cantidad de herramientas que apoyan la labor del analista de pruebas, inclusive se pueden conseguir herramientas para cada tipo de lenguaje, estas herramientas pueden facilitar el desarrollo de pruebas, elaboración de casos de pruebas, seguimiento de errores, etc. En el capitulo 1 de esta investigación se mencionan algunas de ellas, proponiendo su uso en la facultad.

En la imagen que se muestra a continuación es una de las mejores formas de representar el procedimiento de pruebas de caja blanca, en este tipo de pruebas el cubo representaría un sistema en donde se pueden observar los diversos componentes que forman parte del mismo, cada uno de estos componentes debe ser probado en su totalidad (óvalos), y también sus interfaces o comunicaciones con los demás componentes (flechas), este tipo de pruebas también son llamadas pruebas de caja de cristal ya que este último termino representa mejor el tipo de pruebas.

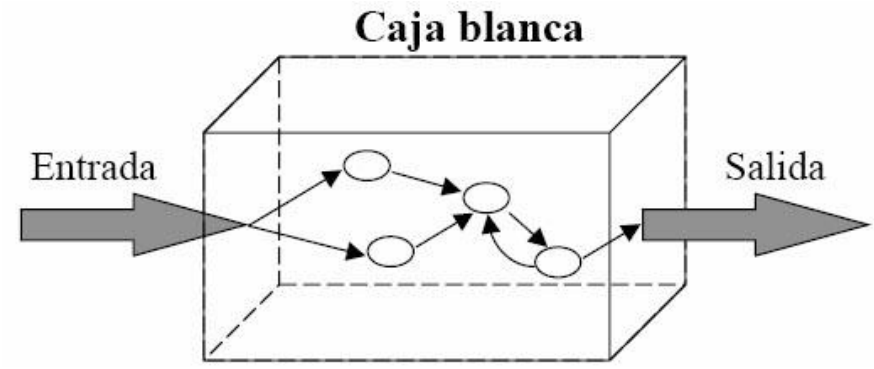

Figura 3.2 Pruebas de Caja Blanca

El procedimiento para obtener buenos casos de pruebas y aplicar las pruebas antes mencionadas es el siguiente:

1. **Usando el diseño o el código como base, se dibuja el correspondiente grafo de flujo.** Se crea un grafo usando los símbolos y las normas de construcción establecidas para tal efecto, numerando las sentencias de código y haciendo coincidir estos números con sus correspondientes nodos del grafo de flujo. (99)

2. **Se determina la complejidad Ciclomática del grafo de flujo resultante.** La complejidad Ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto del método de prueba del camino básico, el valor calculado como complejidad Ciclomática define el número de caminos independientes del conjunto *básico* de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. (100)

   Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición. En términos del grafo de flujo, un camino independiente está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino. (101)

   La complejidad se puede calcular de tres formas: (102)

   1. El número de regiones del grafo de flujo coincide con la complejidad Ciclomática.

   2. La complejidad Ciclomática, V (G), de un grafo de flujo G se define como:

   V (G)=A-N+2 donde **A** es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

   3**.** La complejidad Ciclomática, V (G), de un grafo de flujo G también se define como V (G) =P+ 1 donde P es el número de nodos predicado contenidos en el grafo de flujo G.

3. **Se determina un conjunto básico de caminos linealmente independientes.** El valor de V (G) da el número de caminos linealmente independientes de la estructura de control del programa. Normalmente merece la pena identificar los nodos predicados para que sea más fácil obtener los casos de prueba. (103)

4. **Se preparan los casos de prueba que forzarán la ejecución de cada camino del conjunto básico.** Se debe escoger los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino. Para cada caso de prueba se

prueban con los valores límites, medios, mínimos y superiores a los establecidos. Co los tres primeros se garantiza que funciona con los valores establecidos, y con el último se verifica la validación, en caso de no resistir un valor superior. Además de esta forma se realizan menos casos de pruebas. (104)

Se ejecutan cada caso de prueba y se comparan los resultados obtenidos con los esperados. Una vez terminados todos los casos de prueba, el responsable de la prueba podrá estar seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez. Es importante darse cuenta de que algunos caminos independientes no se pueden probar de forma aislada. Es decir, la combinación de datos requerida para recorrer el camino no se puede conseguir con el flujo normal del programa. En tales casos, estos caminos se han de probar como parte de otra prueba de camino.

De igual forma se diseñan los casos de pruebas, para las pruebas de bucle, como se explica en el capitulo 1 de esta investigación, pues el procedimiento es el mismo, solo que varia el numero de veces que se debe ejecutar un bucle, así como el tipo de bucle que se esté probando. (105)

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

- Los datos de entrada son conocidos por el probador o Analista de Pruebas y estos deben ser preparados con minuciosidad, ya que el resultado de las pruebas dependen de estos.
- Se debe conocer que componentes interactúan en cada caso de prueba.
- Se debe conocer de antemano que 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 y se comienza con la siguiente.

Luego de tener una buena cantidad de módulos independientes probados y encontrados Conformes, el siguiente paso es integrarlos en dependencia del tipo de integración que lleve ya sea Incremental o no incremental.

Estrategias de pruebas del software:

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. (106)

Las características generales son: (107)

- La prueba comienza en el nivel de módulo y trabaja “hacia afuera”.

- En diferentes puntos son adecuadas a la vez distintas técnicas de prueba.
- La prueba la realiza la persona que desarrolla el software y (para grandes proyectos) un grupo de pruebas independiente.
- La prueba y la depuración son actividades diferentes.

. Más concretamente, los objetivos de la estrategia de prueba son:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de unidad, integración y las pruebas de sistema. Las pruebas de unidad y de integración son necesarias dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Los productos de desarrollo de software en los que se detectan defectos son probadas de nuevo y posiblemente devueltas a otra etapa, como diseño o implementación, de forma que los defectos puedan ser arreglados.

3.11 Fase Depuración y Cambios; Ejecución de las pruebas de Integración.

Al concluir la fase de pruebas de unidad se procede a ejecutar las pruebas de integración según lo planificado en la fase de revisión del diseño de la arquitectura**.**

Las pruebas de integración en los proyectos de desarrollo de software, no solo se presentan en la integración entre módulos de un mismo producto sino que se están planteando proyectos que ofrecen soluciones conformadas por varios productos de software, lo cual le da una nueva dimensión al proceso de pruebas de integración. (108) Estas corresponden a las pruebas de integración entre productos, en el cual se podría considerar los productos como componentes más grandes, sin embargo existen características que las hacen particulares considerando que son productos construidos por áreas temáticas diferentes, o dentro de la misma área temática pero no por los mismos proyectos y que obedecen a estándares de construcción distintos, en las más diversas plataformas

Estas pruebas son necesarias, pues aunque todos los módulos funcionan bien por separado, se pone en duda de que funcionen todos juntos y funcionen bien. El problema es ponerlos juntos en interacción, pues los datos se pueden perder en una interfaz; un módulo puede tener un efecto adverso e inadvertido sobre otro; las subfunciones, cuando se combinan, pueden no producir la función

principal deseada; la imprecisión aceptada individualmente puede crecer hasta niveles inaceptables; y las estructuras de datos globales pueden presentar problemas. (109) La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es coger los módulos probados mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

3.11.1 Procedimiento.

Estas pruebas se pueden plantear desde un punto de vista estructural o funcional.

Las pruebas estructurales de integración son similares a las pruebas de caja blanca; pero trabajan a un nivel conceptual superior. En lugar de referirse a sentencias del lenguaje, se hará referencia a llamadas entre módulos. Se trata de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura de segmentos o de ramas. (110)

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

Las pruebas finales de integración cubren todo el sistema y pretenden cubrir plenamente la especificación de requisitos del usuario. Además, a estas alturas ya suele estar disponible el manual de usuario, que también se utiliza para realizar pruebas hasta lograr una cobertura aceptable.

Para integrar los módulos de forma ascendente se desarrollan mediante los siguientes pasos: (111)

1. Se combinan los módulos de bajo nivel en grupos (a veces denominados construcciones) que realicen una subfunción específica del software.
2. Se escribe un controlador (un programa de control de la prueba) para coordinar la entrada y la salida de los casos de prueba.
3. Se prueba el grupo.
4. Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

Según se van ejecutando estos pasos se van probando todas las funcionalidades de uno y otro módulo, comprobando que no varían los servicios y que no funcionan bien solamente separados sino que una vez integrados se mantienen los servicios y se inicializan otros.

Se propone además como tipo de prueba en la integración las siguientes:

- **Autenticación Única de Usuarios**, que se desarrolla de la siguiente forma:

Se valida el registro de los usuarios a la solución, la conexión se realiza una vez y este debe ser reconocido en cada una de las aplicaciones de la solución, sin que se solicite nuevamente login y usuario al invocar la ejecución de alguna de ellas. Se debe verificar que los accesos del usuario en cada aplicación corresponden a los definidos en cada producto de la solución.

- **Administración de Usuarios**

Se validan los requerimientos de administración de usuarios (creación de usuarios, modificación de datos, cambio de contraseña, eliminación de usuarios, activación y desactivación de usuarios). Esta función normalmente se implementa a través del mediador. (112)

- **Búsqueda Unificada**

Se validan que a partir de un buscador central se tenga acceso a los buscadores de las demás aplicaciones. Se debe establecer cuales de las aplicaciones tienen buscador y se valida que exista compatibilidad entre los parámetros de búsqueda y que los resultados sean correctos y completos.

Este tipo de prueba se realiza en el caso que la aplicación posea un buscador interno, y que en alguno de los módulos, exista de la misma forma un buscador, que a la hora de integrarse, debe incluirse uno dentro del otro, posibilitando la realización de cualquier búsqueda dentro de la aplicación. (113)

- **Integración Funcional**

Se validan la sincronización a nivel de procesos, es decir la interacción entre los productos a partir de la ejecución de procesos y la transferencia de información a otros procesos. Se deben identificar los procesos de las aplicaciones que interactúan y el mecanismo de intercambio de información. Este puede realizarse a través de interfaces directas entre aplicaciones o mediante el uso de herramientas altamente configurables que actúan como mediadores. (114)

En la integración se revisa además el redireccionamiento de la aplicación, así como la funcionalidad de los botones para retroceder y adelantar en las páginas. Puesto que una vez integrados los componentes, las direcciones cambian, así como su camino de acceso.

Todos los defectos detectados durante la integración se registrarán en una planilla de no conformidades que se anexará al expediente del proyecto y que tendrán que ser analizadas por el equipo de desarrollo, para valorar y procesar los defectos detectados en el proceso de integración.

Al concluir este proceso de integración con todo su correcto funcionamiento, la aplicación se encuentra en condiciones de pasar a la etapa de pruebas del sistema.

3.12 Fase Depuración y Cambios; Ejecución de las pruebas del Sistema

3.12.1 Procedimiento.

Según la planificación de la aplicación en cada una de las fases del ciclo de vida, una vez concluidas todas las pruebas de caja blanca, los miembros del grupo interno de calidad proceden a ejecutar los diseños de casos de pruebas previamente diseñados en la etapa de análisis y diseño.

Estas pruebas serían el comienzo de la primera fase de ejecución del diseño de casos de prueba, para ello se debe llevar el control de no conformidades detectadas en este proceso. De igual forma un control con las clases válidas e inválidas usadas en la realización de las pruebas.

Al concluir esta etapa de pruebas, las no conformidades serán entregadas al grupo de desarrollo para su depuración y cambios, para luego volver a ser procesados los mismos diseños de casos de pruebas para verificar su correcta implementación o detección de nuevos fallos. Este ciclo será repetido mientras que el grupo interno de calidad esté detectando no conformidades.

Para la ejecución de las pruebas de carga, volumen, estabilidad y seguridad, el grupo de desarrollo en coordinación con el equipo de calidad de la facultad deberá entregar en el expediente del proyecto, un documento donde se especifique el alcance de la aplicación, es decir se refleje de una forma clara la cantidad aproximada de usuarios para la que está diseñada la aplicación. Además se debe especificar en situaciones extras, el máximo de usuarios que pueden conectarse a la misma, así como la tendencia real de conexiones simultáneas. Por otro lado debe estar especificado también un amplio juego de datos que permitan a los probadores moverse por toda la aplicación. Lo más importante en esta documentación es la entrega de los usuarios y contraseñas para los distintos niveles de acceso en el sistema.

Antes de someter cualquier aplicación a este tipo de pruebas se necesita capacitar al personal que será encargado de realizar las mismas. En este caso seria a través de talleres o cursos optativos dirigidos específicamente a como realizar cada una de estas pruebas.

En el caso de las pruebas de seguridad, primeramente el equipo de calidad de la facultad debe estar preparado para las mismas, pues como se aborda en el capitulo 1 de esta investigación, este tipo de pruebas constituyen ataques en todos los sentidos a la aplicación. Por lo menos un grupo del equipo de calidad de la facultad debe especializarse en este tipo de pruebas, mediante investigaciones, cursos optativos orientados a nivel central y explotar todas las vías que existen para atacar a los sistemas y apoderarse de la información. Para fortalecer de esta forma la seguridad que hasta ahora se ha venido implementando en todas las aplicaciones de la facultad y que poco se ha profundizado en este tema.

Como segunda actividad fundamental es condicionar el laboratorio de calidad con las herramientas que simularán las pruebas y que en esta investigación se proponen algunas de ellas en el capitulo 1.

Como tercera actividad, se tiene capacitar al personal en el uso de las herramientas, que se aprueben a nivel de facultad o a nivel de la Dirección Central de Calidad.

Con las pruebas de carga se busca identificar las condiciones de carga pico en las cuales el programa fallará en manejar las cargas de procesamiento requeridas dentro del periodo de tiempo requerido. Para este tipo de pruebas se propone aplicar el JMeter, herramienta de la cual se habla en el capitulo 1 de esta investigación.

Esta herramienta debe estar instalada en una de las máquinas de este laboratorio como amo o controlador de las peticiones, y como esclavo o receptor de peticiones en las demás máquinas del laboratorio de calidad, para poder simular la carga de conexiones que debe soportar la aplicación en tiempo real una vez este montada en su destino final.

Esta versión de máquina esclavo debe estar montada un una partición con sistema operativo Linux, no así con la máquina amo que puede estar en Windows.

Para comenzar la ejecución de las pruebas de carga, se planifica según la cantidad máxima promedio de usuarios para la que está diseñada la aplicación y se realizan las peticiones a la herramienta de realizar la simulación de conexiones de forma incremental, pues es cierto que en el mismo momento nunca se van a conectar tantos usuarios simultáneamente.

Cada máquina esclavo puede mandar un porciento determinado de peticiones que se repetirán cada cierto tiempo establecido, hasta llegar a la carga que se defina previamente antes de ejecutar la prueba.

Con esta carga de conexiones se procede a ejecutar las pruebas de estabilidad, que se realizarán observando el comportamiento de los servicios con la carga de conexiones por un tiempo determinado, al final el tiempo de respuesta no debe variar, y cada servicio debe mantenerse sin sufrir cambios.

Para la ejecución de las pruebas de volumen se planifica mediante el tipo de servicio que brinde la aplicación, donde planificaría una demanda que sature la aplicación en cuanto a datos. Por ejemplo se solicitaría desde varias conexiones la generación de un reporte determinado (n) veces y al mismo tiempo la búsqueda de algunos datos específicos para ser mostrados en pantalla, de forma tal que la aplicación tenga que usar un gran Volumen de datos para satisfacer la petición del cliente.

Todas estas pruebas se planifican en dependencia de los datos que entreguen los desarrolladores en el expediente del producto y para todas se debe llevar el registro de no conformidades, así como el diseño de los casos de pruebas, con las clases Validas e invalidas.

Al finalizar esta etapa de pruebas, se entrega el registro de no conformidades al equipo de desarrollo, para su posterior depuración y cambios, los cuales una vez estén solucionados deben pasar nuevamente por este tipo de pruebas para comprobar su correcta depuración y detectar otras fallas que puedan haber surgido con los cambios, o que se hallan pasado por alto en una primera revisión.

Esta entrega puede ser coordinada previamente con el grupo de desarrollo, y entregarse diario un registro, con las no conformidades detectadas en el día, para que el equipo de desarrollo aproveche el tiempo y trabaje sobre los defectos encontrados, y no este parado mientras se estén desarrollando las pruebas. Todo este proceso debe ser previamente consultado y planificado, aunque puede ser flexible, en dependencia de la complejidad de la aplicación.

Al concluir el ciclo de pruebas del sistema, la aplicación se encuentra en condiciones de pasar a la etapa de pruebas de liberación, por parte del equipo de calidad de la Facultad.

3.13 Fase Depuración y Cambios; Ejecución de las pruebas de Aceptación y/o Liberación.

Esta es la última fase del ciclo de vida de una aplicación Web y es donde comienza todo el proceso de pruebas de aceptación, que se inicia por solicitud de las distintas áreas temáticas al laboratorio de calidad de la facultad 7.

3.13.1 Procedimiento.

Una vez realizada la solicitud, el equipo de desarrollo y el equipo de calidad se reúnen y planifican los cronogramas de entrega y liberación de la aplicación, de esta reunión, debe quedar un acta firmada por ambas partes como constancia de la entrega oficial del producto.

Luego de acordar la entrega de la aplicación, el equipo de desarrollo entrega al equipo de calidad un expediente, el cual se crea según la norma cubana NC ISO/IEC 12119, y que debe contener la aplicación Web que se probará, el documento de especificación de casos de uso, el manual de usuario, un glosario de términos, y en caso de no estar incluidos en la especificación de casos de uso, un documento que contiene los requerimientos y a que caso de uso corresponde cada uno de ellos. En este expediente además deberán entregar un documento con los usuarios, permisos, nombres y demás detalles necesarios que se deban conocer para la instalación del software y no estén incluidos en los manuales.

Por otro lado, de conjunto con el manual antes mencionado el equipo de desarrollo deberá entregar al equipo de calidad con antelación los requerimientos mínimos de hardware y software para que la aplicación funcione.

Otro aspecto fundamental, que hay que tener en cuenta es, que para la realización de las pruebas, la base de datos debe tener por lo menos algunos juego de datos para cada servicio, para que el probador cuente con todo lo necesario y utilice datos específicos.

Cuando la aplicación se encuentra en el laboratorio de calidad, se crea una copia del expediente entregado por el grupo de desarrollo, en el cual se irán incluyendo todo lo que se genere durante el proceso de pruebas.

Antes de comenzar con las pruebas a la documentación y la aplicación se considera necesario aplicar como técnica de aceptación, el uso de listas de chequeo para medir atributos de las interfaces y de calidad con los que debe contar la aplicación. Existen algunas listas de chequeo definidas en el laboratorio de calidad de la facultad 7, que pueden ser ajustadas a este tipo de aplicaciones, mostrando al final de esta revisión un resultado cuantitativo de las características del producto, así como de su documentación.

Este tipo de revisión puede dar una medida de la situación actual del producto, pudiendo el laboratorio de calidad de la facultad 7, rechazar, el producto desde este momento, o aceptarlo y proceder al resto de las pruebas que a continuación se describen.

Las primeras pruebas que se realizan serán las pruebas al manual de usuario, donde los probadores deben ser capaces de moverse por una aplicación que no conocen y comprenderla así como saber hacia donde ir y que hacer.

Dentro de esta revisión, entran detalles como la ortografía y la redacción, así como la forma en que se detallan las funciones, para los usuarios, donde a veces los desarrolladores asumen que el usuario es un experto en el tema y se omiten detalles importantes.

Las próximas pruebas a aplicar son las pruebas funcionales, las cuales mediante la técnica de caja negra verifican si el software cumple con los requerimientos definidos por el usuario. Para esto se revisa la primera fase de la ejecución de los casos de pruebas diseñados por el grupo interno de calidad del área temática a donde pertenece la aplicación.

En dependencia de la calidad que tengan estos diseños se refinan y proponen un nuevo diseño de casos de pruebas, a partir de los casos de uso, de forma tal que al final del diseño se tendrá un caso de prueba por cada caso de uso existente. Para los casos de uso incluidos no se diseñarán casos de pruebas sino que estos se agregarán en el caso de uso que los contenga y se probarán cuantas veces aparezcan en el software. A medida que se diseñan los casos de pruebas se probará el documento de especificación de caso de uso, se comprobará que lo que se encuentra en la aplicación corresponde a lo descrito en el documento así como la ortografía y la redacción.

De esta forma se pone en ejecución la segunda fase del diseño de casos de prueba que en dependencia de la saturación de productos que se estén revisando en ese momento en el laboratorio de calidad, se podrán llevar a cabo estas dos pruebas a la vez, designando un grupo para hacer las pruebas a los manuales, mientras que otro grupo diseña los casos de pruebas y llevan a la practica sus diseños, cada uno llevando a la vez el registro de no conformidades. De esta forma se ahorra tiempo y se proporciona una mayor cantidad de pruebas permitiendo una mayor detección de errores y defectos.

En caso de que la aplicación, sea muy compleja, y esté compuesta por varios módulos que se integrarán al final para lograr el producto completo, se probarán primero los módulos por separados y luego se diseñará un caso de prueba de integración para probar el software como un todo, en este caso en el expediente que entrega el grupo de desarrollo debe aparecer además un documento donde se explique la dependencia entre módulos, así como de sus datos con respecto a otros. Es bueno aclarar que si la aplicación es más sencilla esta descripción no es necesaria.

Este tipo de pruebas deben diseñarse de forma tal que cumplan un objetivo trazado, y no realizar pruebas a lo loco, para determinar errores que puedan surgir al azar. Es por ello que se propone sean diseñadas teniendo en cuenta que respondan las siguientes preguntas: (115)

¿Cómo se prueba la validez funcional?

¿Qué clase de entradas compondrán buenos casos de prueba?

¿Es el sistema sensible a ciertos valores de entrada?

¿De qué forma están aislados los límites de una clase de datos?

¿Qué volúmenes y niveles de datos tolerará el sistema?

¿Qué efectos sobre la operación del sistema tendrán combinaciones específicas de datos?

Además se diseña y aplica un caso de prueba del sistema, que no es más que un caso de prueba que sigue un flujo básico, ya que con las pruebas modulares se puede perder la visión general del producto, pues probará todos los flujos alternativos. Con el mismo se puede simular el proceso en vivo como mismo debe ocurrir cuando el usuario final lo utilice.

Por otro lado se realizan las pruebas exploratorias, donde los probadores intentan utilizar todos los servicios de la aplicación, según lo descrito en el manual de usuario, o guiado muchas veces por lo que la aplicación por lógica permite hacer. En este tipo de prueba se trata de modificar valores, se verifican las validaciones de cada una de las entradas posibles, se realizan búsquedas, entradas de datos incorrectos, pruebas que de alguna forma traten de desequilibrar la aplicación, para comprobar su correcto funcionamiento.

Para este tipo de pruebas también se lleva el registro de no conformidades, donde se documenta todo los señalamientos, defectos y sugerencias que los probadores detecten en el transcurso de estas pruebas.

En la realización de estas pruebas es recomendable seguir como método los que a continuación se proponen

Métodos:

Partición equivalente: divide el campo de entrada en clases de datos, para descubrir clases de errores, reduciendo la cantidad de casos de pruebas a desarrollar. Se planifica cuales serán las entradas, para

eliminar redundancias en las pruebas. Para ello se delimitan los tipos de datos que serán probados. (116)

Análisis de los valores límite: elige casos de prueba que ejerciten los valores límites. En este caso se realizaran pruebas con valores próximos a los límites, muy por debajo y fuera de estos, garantizando de esta forma el correcto funcionamiento de la aplicación con los valores planificados. (117)

Prueba de comparación: se utiliza SW y HW redundante. Se prueba cada versión con los mismos datos y se comparan los resultados. Este tipo de prueba debe realizarse en condiciones similares a donde estará la aplicación finalmente. Es decir medir el acceso desde máquinas con diferentes niveles de acceso, diferentes navegadores Web, y así en dependencia de los requisitos especificados al comienzo del ciclo de vida. (118)

Otro aspecto que hay que tener en cuenta es la forma en que se realizaran estas pruebas, que por ser exploratorias, no implica que no tengan una fundamentación o forma de proceder. Para ello se consideran necesario tener presente los siguientes consejos: (119)

- Use siempre datos de entrada bien definidos para los que se conozcan los resultados correctos que deben obtenerse.
- Detecte primero los defectos obvios (usando datos de prueba muy simples) y luego sí realice pruebas más complejas.
- Cuando modifique algo mientras prueba realice un solo cambio cada vez y utilice los mismos datos con los que detectó el defecto.
- Pruebe el programa para verificar si detecta entradas incorrectas.

El grupo de calidad, en previo acuerdo con el grupo de desarrollo realiza la entrega de las no conformidades detectadas durante la revisión, las que pueden ser al finalizar cada jornada de pruebas, para que el grupo de desarrollo aproveche el tiempo y trabaje en las deficiencias detectadas, o puede ser un informe final donde se recojan todas las no conformidades del ciclo de pruebas de aceptación.

De esta forma se producirá una secuencia de entrega donde el grupo de calidad chequeará en cada entrega el avance en cuanto a las deficiencias anteriormente señaladas, por lo que es de vital importancia una buena comunicación entre ambos equipos de trabajo, para lograr el desarrollo exitoso del producto que llegará al usuario final.

No obstante, en cada iteración de la entrega, el equipo de desarrollo deberá entregar un documento adjunto a la nueva versión, que especifique exactamente que parte se cambió, y si aún se trabaja en alguna parte de la aplicación, por lo que algún servicio se verá afectado. Facilitando de esta forma el trabajo, de ambos equipos y evitando que se repitan los señalamientos. Además de esta forma se podrán detectar nuevos errores que en revisiones anteriores pueden haber pasado inadvertidas.

Finalmente se elaborará un acta, donde se certifique, que la aplicación fue procesada por el grupo de calidad de la facultad 7 de la Universidad de Ciencias Informáticas y que está libre de errores.

Una vez liberado el producto, se procederá a crear las condiciones necesarias para que el usuario final pruebe la aplicación, pues este es el único que puede detectar ciertos detalles y que así se describe en el plan de pruebas de aceptación, en la primera parte de pruebas denominadas de tipo alfa, al inicio de este capítulo.

Para esta etapa del proceso de pruebas los ingenieros de prueba refinan una vez más los casos de pruebas con la última versión del software, que es el resultado de todas las pruebas anteriores.

Para probar, el usuario final se guiará por los casos de pruebas ya refinados, así como necesitará el documento según las características del software con los datos que contiene la Base de datos, aplicará el caso de prueba del sistema y las listas de chequeo.

En un primer encuentro se explicará al usuario el objetivo y propósito de las pruebas, así como la manera en que serán realizadas, a medida que avancen las pruebas se recogerán todos los señalamientos y problemas detectados, que los miembros del equipo de calidad redactarán en el "Sumario de Evaluación de Pruebas", al final de cada día se realiza una reunión de conclusión del día, donde se expondrán los resultados del día, así como cualquier dificultad o inconveniente que haya podido surgir.

El "Sumario de Evaluación de Pruebas" puede hacerse después del ultimo día de las pruebas después de tener todos los señalamientos hechos por el usuario, pero realmente esto no es recomendable, es mejor que se vaya conformando según se termina cada día, pues en ocasiones no se entienden bien los señalamientos realizados por el usuario o no lo redactaron correctamente, y así el ingeniero de pruebas puede aclarar cualquier duda al respecto al siguiente día.

Luego de terminados todos los días previstos para las pruebas los ingenieros de pruebas realizan las solicitudes de cambio a partir del "Sumario de Evaluación de Pruebas", estas solicitudes se entregan a los desarrolladores los cuales redactarán un documento en que expondrán cuales proceden a

realizarse y cuales no así como la causa de estas decisiones. También se realizará el sumario de las listas de chequeo, donde se obtiene un resultado cuantitativo de las mismas.

Después de esto los desarrolladores trabajarán sobre las solicitudes de cambio realizadas hasta una segunda presentación del software al usuario final y así este proceso se repetirá las veces que sea necesario con la diferencia de que ya no se diseñaran nuevos casos de pruebas sino el equipo de calidad refina los ya realizados y los vuelve aplicar así como verificará si estas solicitudes fueron llevadas a cabo y redacta un documento resumen de estas.

También para cada solicitud de cambio los desarrolladores deben incluir que solución fue dada y que partes de la aplicación fueron afectadas con los cambios lo que incluye requerimientos y caso de uso, esto se hace en busca de nuevos errores incluidos.

Al terminar el proceso se tendrá el expediente actualizado con todo lo correspondiente al proceso de pruebas, así quedará archivado un historial sobre las pruebas realizadas, este incluirá un informe sobre las pruebas realizadas, este informe recoge todo lo sucedido en las diferentes etapas del proceso de pruebas.

Una vez concluido todo este proceso de pruebas, se liberará el producto totalmente, con una garantía definida previamente por el grupo de desarrollo, donde los desarrolladores serán los máximos encargados de corregir cualquier falla que presente la aplicación durante este intervalo de pruebas en el ambiente real que se mantendrá la aplicación.

Al concluir la elaboración del procedimiento se puede concluir que:

- La puesta en marcha del presente procedimiento debe resolver o mejorar la situación problemica existente en la Facultad 7 de la UCI, la cual es abordada en la introducción de esta investigación.
- La normalización o estandarización de los tipos de pruebas, forma y momento en que serán aplicadas, teniendo en cuenta el responsable de su aplicación, permitirán darle una mejor organización a la producción de la Facultad 7.
- La inserción de nuevas pruebas y el uso de herramientas que se proponen mejorarán el éxito en el desempeño de las aplicaciones Web que se implementen en la Facultad 7.

- La satisfacción del cliente es el aspecto principal a tener en cuenta en el proceso de pruebas y revisión del producto que se desarrolla.

Conclusiones.

Con la presente investigación, luego de un estudio realizado sobre las estrategias y métodos de pruebas existentes a nivel mundial y una caracterización de la producción de la Facultad 7, así como el cumplimiento de los objetivos y tareas trazados, se arribó a las siguientes conclusiones:

- Se realizó un análisis sobre los procesos, metodologías y modelos de pruebas existentes, definiéndose como guía para la elaboración del procedimiento el *Modelo W* de pruebas de software.
- Se valoró el uso e importancia de herramientas en la realización y simulación de pruebas en aplicaciones Web, recomendándose el uso de algunas de ellas que cumplen con los estándares de producción de la Universidad.
- Se aplicaron entrevistas a desarrolladores de las distintas Áreas Temáticas del polo Informática para la Salud, las que aportaron datos reales que contribuyeron a reafirmar la necesidad de un procedimiento que estandarice los procesos de pruebas de forma general en la Facultad 7.
- Se definió un procedimiento general que garantice el éxito en el desarrollo de cada fase del ciclo de vida de las aplicaciones Web, el cual proporcionará una mejor organización a la producción de la Facultad 7, mediante la inserción de nuevas pruebas y el uso de herramientas, así como un mayor control de los procesos.

Recomendaciones.

- Aplicar y dar seguimiento al procedimiento propuesto en los proyectos productivos de la Facultad 7 que desarrollen aplicaciones Web.
- Extender el uso del procedimiento a la Universidad de las Ciencias Informáticas en general y no solo para las aplicaciones Web, sino que cada proyecto pueda adaptarlo a las características de sus sistemas o productos.
- Realizar un procedimiento en cada una de las fases que aquí se abordan mediante el desarrollo de otros temas de tesis que den cumplimiento a cada uno de los pasos y actividades descritos globalmente. Proponiendo todas las pruebas que se deben hacer en cada fase, así como responsables de las mismas.
- Realizar un estudio profundo en la aplicación de pruebas a través de herramientas que simulen su desarrollo, teniendo en cuenta los lenguajes de programación, licencias y tipos de pruebas.
- Profundizar en el estudio de la aplicación de pruebas de Seguridad, Carga, Volumen y Estabilidad para una posterior capacitación del personal que debe ejecutar las mismas.

Referencia Bibliográfica

1. Introducción a Patrones. [En línea] [Citado el: 12 de diciembre de 2007.] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html.

2. Arquitecturas: Algunos Fundamentos. [En línea] [Citado el: 24 de Enero de 2008.] http://homepage.mac.com/imaz/iblog/C612772037/E20050907214355/Media/Arquitecturas,%20algunos%20fundamentos.pdf..

3. Conferencia_Implementacion.*pdf.* Habana : UCI:Teleformacion, 2007.

4. ídem a Referencia 3.

5. ídem a Referencia 3.

6. **Vignaga Andrés Perovich Daniel .** Enfoque Metodológico para el Desarrollo Basado en Componentes. [En línea] [Citado el: 19 de Noviembre de 2007.] http://dis.um.es/~jsaez/dbc/curso0708/docs/art01_vp03.pdf.

7. Universidad Nacional Mayor De San Marcos. *Glosario de Términos.* [En línea] [Citado el: 13 de Diciembre de 2007.] http://www.unmsm.edu.pe/ogp/ARCHIVOS/Glosario/inds.htm.

8. **Brey, Gustavo A. (2006). SOA (Arquitectura Orientada a Servicios).** Arquitectura Orientada a Servicios. [En línea] [Citado el: 16 de Noviembre de 2007.] http://www.ibm.com/es/.

9. Idem a Referncia 8. [En línea]

10. Idem a Referncia 8. [En línea]

11. Idem a Referencia 8. [En línea]

12. **Velasco Elizondo Perla Inés .** PRUEBA DE COMPONENTES DE SOFTWARE BASADAS EN EL MODELO DE JAVABEANS. [En línea] Abril de 2001. [Citado el: 12 de Enero de 2008.] http://www.cs.man.ac.uk/~velascop/publ/Tesis.pdf.

13. Departamento de Lenguajes y Sistemas Informáticos. [En línea] Enero de 2008. [Citado el: 26 de Enero de 2008.] http://lsi.ugr.es/~arroyo/inndoc/doc/pruebas/pruebas_d.php.

14. Idem a Referencia 13. [En línea]

15. Idem a Referencia 13. [En línea]

16. Idem a Referncia 13. [En línea]

17. **Blog de Mauricio.** Carrera de Ingenieria Informatica. [En línea] 14 de Noviembre de 2006. [Citado el: 9 de Febrero de 2008.] http://my.opera.com/pelican0/blog/show.dml/564829.

18. Idem a Referencia 17.

19. Idem a Referncia 17.

20. Idem a Referncia 17.

21. Idem a Referencia 17.

22. Idem a Referencia 17.

23. Guia de Tecnicas de Pruebas y Metricas. [En línea] [Citado el: 21 de Febrero de 2008.] http://www.lpsi.eui.upm.es/MDes/TfcMetrica/GTPrueb.htm.

24. **profesores, Colectivo de.** Conferencia de flujo de prueba. [En línea] 2006. [Citado el: 18 de Noviembre de 2008.] http://teleformacion.uci.cu/mod/resource/view.php?id=10817&subdir=/Conferencias_IS2_05-06.

25. Idem a la Referncia 13.

26. Idem a la Referencia 13.

27. Idem a la Referncia 13.

28. Idem a la Referencia 13.

29. **Pressman, Roger S.** *Ingeniería de software un enfoque practico.* Quinta Edición. pág. 316.

30. *I*dem a la Referencia 29.

31. **Cejas(Venezuela), Julio.** Pruebas en SOA(Arquitecturas orientadas en servicios). [En línea] Mijao Blog, 21 de Mayo de 2007. [Citado el: 16 de Noviembre de 2007.] http://mijao.blogspot.com/2007/05/pruebas-en-soaarquitecturas-orientadas.html.

32. Idem a la Referencia 31. [En línea]

33. Idem a la Referencia 31. [En línea]

34. Entorno de Pruebas. Herramientas. [En línea] 2007. [Citado el: 10 de Noviembre de 2007.] http://www.als-es.com/home.php?location=herramientas/entorno-pruebas.

35. Idem a la Referencia 34. [En línea]

36. Idem a la Referencia 34. [En línea]

37. Idem a la Referencia 34. [En línea]

38. Idem a la Referencia 34. [En línea]

39. Idem a la Referencia 34. [En línea]

40. Idem a la Referencia 34. [En línea]

41. Idem a la Referencia 34. [En línea]

42. Idem a la Referencia 34. [En línea]

43. Idem a la Referencia 34. [En línea]

44. Idem a la Referencia 34. [En línea]

45. Idem a la Referencia 34. [En línea]

46. Idem a la Referencia 34. [En línea]

47. Software Quality Assurance Network. [En línea] 10 de Octubre de 2005. [Citado el: 14 de Marzo de 2008.] http://softqanetwork.com/?p=9.

48. Idem a la refrencia 47. [En línea]
49. Idem a la Referencia 47. [En línea]
50. ALS Software Lifecicle Optimization. [En línea] 2007. [Citado el: 1 de Marzo de 2008.] http://www.als-es.com/home.php?location=herramientas/entorno-pruebas/soatest.
51. Idem a la Referencia 50. [En línea]
52. Idem a la Referencia 50. [En línea]
53. Idem a la Referencia 50. [En línea]
54. Idem a la Referencia 50. [En línea]
55. Idem a la Referencia 50. [En línea]
56. ALS Software Lifecicle Optimization. [En línea] 2007. [Citado el: 1 de Marzo de 2008.] http://www.als-es.com/home.php?location=herramientas%2Fentorno-desarrollo%2Fjtest.
57. Idem a la Referencia 56. [En línea]
58. Idem a la Referencia 56. [En línea]
59. Idem a la Referencia 56. [En línea]
60. Idem a la Referencia 56. [En línea]
61. Idem a la Referencia 56. [En línea]
62. Idem a la referencia 56. [En línea]
63. Idem a la Referencia 56. [En línea]
64. Idem a la Referencia 13. [En línea]
65. Idem a la Referencia 13. [En línea]
66. Idem a la Referencia 13. [En línea]
67. Idem a la Referencia 13. [En línea]
68. Idem a la Referencia 13. [En línea]
69. **Presuman, Roger S.** *Ingeniería del Software: Un enfoque practico.* 3ra Edición. págs. Pag. 26-30.
70. Idem a la Referencia 69.
71. Idem a la referencia 69.
72. Idem a la Referencia 69.
73. Idem a la Referencia 69.
74. Idem a la Referencia 69.
75. Intro Ingenieria de Software. *MODELO CASCADA Y MODELO V.* [En línea] Noviembre de 2007. [Citado el: 24 de Febrero de 2008.] http://caraujo334.blogspot.es/1192584300/.
76. Idem a la Referencia 75. [En línea]

77. Kynetia Software for Business Solutions. *Metodología de Pruebas .* [En línea] [Citado el: 3 de Abril de 2008.] http://www.kynetia.es/calidad/metodologia-de-pruebas.html.

78. Idem a la Referencia 77. [En línea]

79. Idem a la Referencia 77. [En línea]

80. Idem a la Referencia 77. [En línea]

81. Idem a la Referencia 77. [En línea]

82. **Domínguez Fortún Jandrich, Bonal Cáceres Rolando, Pérez Valdés Ignacio.** *Infraestructura productiva. Manual de procedimientos.* s.l. : UCI, 2008.

83. Implantacion y Aceptacion del Sistema. [En línea] [Citado el: 20 de Marzo de 2008.] www.csi.map.es/csi/metrica3/iasproc.pdf.

84. **Teruel., Prof. Alejandro.** Requerimientos de Pruebas. [En línea] 5 de Enero de 2001. [Citado el: 11 de Enero de 2008.] http://www.ldc.usb.ve/~teruel/ci4713/clases2001/testSpecs.html.

85. —. Las Pruebas de Verificación de Requerimientos. [En línea] Marzo de 2001. [Citado el: 14 de Abril de 2008.] http://www.ldc.usb.ve/~teruel/ci4713/clases2001/testReqs.html.

86. **Presman, Roger S.** *"Ingeniería de software un enfoque practico".* Quinta edición. pág. pág. 316.

87. Idem a la Referencia 86.

88. Idem a la Referencia 86.

89. **Presman, Roger S.** *"Ingeniería de software un enfoque practico".* Quinta edición. pág. pág. 269.

90. **Pressman, Roger S.** *"Ingeniería de software un enfoque practico".* Quinta edición. pág. pág. 312.

91. Idem a la Referencia 90.

92. Idem a la Referencia 90.

93. Idem a la Referencia 90.

94. Software Quality Management . [En línea] 2007. [Citado el: 21 de Abril de 2008.] http://softqm.blogspot.com/2006/11/gestin-de-la-calidad-del-software.html.

95. **Lycos Inc.** Estrategias De Prueba Del Software. [En línea] 2 de Noviembre de 2001. [Citado el: 9 de Abril de 2008.] http://www.angelfire.com/my/jimena/ingsoft/guia9.htm.

96. Idem a la Referencia 95. [En línea]

97. Idem a la Referencia 95. [En línea]

98. **Pressman, Roger S.** *"Ingeniería de software un enfoque practico".* Quinta edición. pág. pág. 310 .

99. —. *"Ingeniería de software un enfoque practico".* Quinta edición. pág. pág. 288 .

100. Idem a la Referencia 99.

101. Idem a la Referencia 99.

102. Idema la referencia 99.

103. Idem a la referencia 99.
104. Idem a la Referencia 99.
105. Idem a la Referencia 99.
106. Idem a la Referencia 13. [En línea]
107. Idem a la Referencia 13. [En línea]
108. **V., María José Roca.** Pruebas de Integración de Productos: Un enfoque práctico. [En línea] Julio de 2005. [Citado el: 19 de Febrero de 2008.] http://www.greensqa.com/archivos/Art01-PruebasIntegracionv1.pdf.
109. Idem a la Referencia 95. [En línea]
110. **Mañas, José A.** Prueba de Programas. [En línea] 16 de Marzo de 2004. [Citado el: 31 de Enero de 2008.] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm.
111. **Pressman, Roger S.** *"Ingeniería de software un enfoque practico".* Quinta edición. pág. pág. 313.
112. Idem a la Referencia 108.
113. Idem a la referencia 108.
114. Idem a la Referencia 108.
115. Tecnicas de Pruebas de Software. *Estrategias de pruebas de Software.* [En línea] [Citado el: 18 de Febrero de 2008.] ftp:/.../053_dissistema/contenido/teo_clase_0813_pruebadesistema_2003.ppt .
116. Idem a la Referencia 115. [En línea]
117. Idem a la Referencia 115. [En línea]
118. Idem a la Referencia 115. [En línea]
119. **Valencia, María Eugenia.** PRUEBA DE SOFTWARE. [En línea] 2007. [Citado el: 26 de Enero de 2008.] http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/PRUEBASoftware.pdf.

Bibliografía.

1. **ALS**. 2007. ALS Software Lifecycle Optimization. [Online] 2007. [Cited: Marzo 1, 2008.] http://www.als-es.com/home.php?location=herramientas/entorno-pruebas/soatest.

2. **—.** 2007. Entorno de Pruebas. Herramientas. [Online] 2007. [Cited: Noviembre 10, 2007.] http://www.als-es.com/home.php?location=herramientas/entorno-pruebas.

3. **Álvarez, Mauricio**. 2006. Carrera de Ingeniería Informática. [Online] Noviembre 14, 2006. [Cited: Febrero 9, 2008.] http://my.opera.com/pelican0/blog/show.dml/564829.

4. **Assurance Network**. 2005. Software Quality Assurance Network. [Online] Octubre 10, 2005. [Cited: Marzo 14, 2008.] http://softqanetwork.com/?p=9.

5. **Brey, Gustavo A**. (2006). Arquitectura Orientada a Servicios. SOA (Arquitectura Orientada a Servicios). [Online] [Cited: Noviembre 16, 2007.] http://www.ibm.com/es/.

6. **Cejas, Julio**. 2007. Pruebas en SOA (Arquitecturas orientadas en servicios). [Online] Mijao Blog, Mayo 21, 2007. [Cited: Noviembre 16, 2007.] http://mijao.blogspot.com/2007/05/pruebas-en-soaarquitecturas-orientadas.html.

7. **Domínguez Fortún Jandrich, Bonal Cáceres Rolando, Pérez Valdés Ignacio**. 2008. Infraestructura productiva. Manual de procedimientos. S.l.: UCI, 2008.

8. **Dpto. Lenguajes, Proyectos y Sistemas Informáticos Madrid.** Guía de Técnicas de Pruebas y Métricas. [Online] [Cited: Febrero 21, 2008.] http://www.lpsi.eui.upm.es/MDes/TfcMetrica/GTPrueb.html.

9. **Elizondo, Perla Inés Velasco**. 2001. Prueba De Componentes De Software Basadas En El Modelo De JavaBeans. [Online] Abril 2001. [Cited: Enero 12, 2008.] http://www.cs.man.ac.uk/~velascop/publ/Tesis.pdf.

10. **Facultad de Ciencias, UNAM**. Introducción a Patrones. [En línea] [Citado el: 12 de diciembre de 2007.] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html.

11. **Heredia, Carlos Alberto.** 2007. Escuela de Ingeniería de Sistemas y Computación. [Online] 2007. [Cited: Febrero 5, 2008.] http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/Pruebas.pdf.

12. **Inc, Lycos**. 2001. ESTRATEGIAS DE PRUEBA DEL SOFTWARE. [Online] Noviembre 2, 2001. [Cited: Abril 9, 2008.] http://www.angelfire.com/my/jimena/ingsoft/guia9.htm.

13. **Kynetia**. Kynetia Software for Business Solutions. Metodología de Pruebas. [Online] [Cited: Abril 3, 2008.] http://www.kynetia.es/calidad/metodologia-de-pruebas.html.

14. **Labs, Lucasian**. Herramientas PARASOFT. [Online] [Cited: Noviembre 15, 2007.] http://www.lucasian.com/lucasian_partners.htm.

15. **LSI Universidad de Granada**. 2008. Departamento de Lenguajes y Sistemas Informáticos. [Online] Enero 2008. [Cited: Enero 26, 2008.] http://lsi.ugr.es/~arroyo/inndoc/doc/pruebas/pruebas_d.php.

16. **Mac.com**. ARQUITECTURAS: ALGUNOS FUNDAMENTOS. [Online] [Cited: Enero 24, 2008.] http://homepage.mac.com/imaz/iblog/C612772037/E20050907214355/Media/Arquitecturas,%20algunos%20fundamentos.pdf.

17. **Mañas, José A**. 2004. Prueba de Programas. [Online] Marzo 16, 2004. [Cited: Enero 31, 2008.] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm.

18. **Ministerio de Administraciones Públicas**. Implantación y Aceptación del Sistema. [Online] [Cited: Marzo 20, 2008.] www.csi.map.es/csi/metrica3/iasproc.pdf.

19. **Moreno, Ana M. Sánchez-Segura, Maribel**. Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento de Arquitectónico. [Online] [Cited: Enero 28, 2008.] http://www.willydev.net/descargas/prev/PatronesUsa.pdf.

20. **MSF Intro Ingeniería de Software**. 2007. Intro Ingeniería de Software. MODELO CASCADA Y MODELO V. [Online] Noviembre 2007. [Cited: Febrero 24, 2008.] http://caraujo334.blogspot.es/1192584300/.

21. **Oktaba, Hanna**. 2004. Introducción a Patrones. [Online] 2004. [Cited: Noviembre 19, 2007.] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html.

22. **Pressman, Roger S**. Ingeniería de software un enfoque practico. Quinta edición. p. pág. 269.

23. **—.** Ingeniería de software un enfoque practico. Quinta edición. p. pág. 316.

24. **Pressman**, **Roger S**. "Ingeniería de software un enfoque practico". Quinta edición. p. pág. 312.

25. **—.** "Ingeniería de software un enfoque practico". Quinta edición. p. pág. 310.

26. **—**. "Ingeniería de software un enfoque practico". Quinta edición. p. pág. 288.

27. **—**. "Ingeniería de software un enfoque practico". Quinta edición. p. pág. 313.

28. **—**. Ingeniería de software un enfoque practico. Quinta Edición. p. 316.

29. **Pressman, Roger S**. Ingeniería del Software: Un enfoque práctico. 3ra Edición. pp. Pág. 26-30.

30. **Profesores, Colectivo de**. 2006. Conferencia de flujo de prueba. [Online] 2006. [Cited: Noviembre 18, 2008.]http://teleformacion.uci.cu/mod/resource/view.php?id=10817&subdir=/Conferencias_IS2_05-06.

31. **Quality, Rational Tester for SOA**. 2007. IBM Software Demos. Rational Tester for SOA Quality. [Online] 2007.[Cited: Enero 21, 2008.] http://demos.dfw.ibm.com/on_demand/Download/es/IBM_Demo_Rational_Tester_for_SOA_Quality-1-Mar07.pdf.

32. **Regueira, Pablo**. Técnicas de Pruebas de Software. Estrategias de pruebas de Software. [Online] [Cited: Febrero18, 2008.]ftp:/../053_dissistema/contenido/teo_clase_0813_pruebadesistema_203.ppt.

33. **Softqm.blogspot.com**. 2007. Software Quality Management. [Online] 2007. [Cited: Abril 21, 2008.] http://softqm.blogspot.com/2006/11/gestin-de-la-calidad-del-software.html.

34. **Teruel., Prof. Alejandro**. 2001. Las Pruebas de Verificación de Requerimientos. [Online] Marzo 2001. [Cited: Abril 14, 2008.] http://www.ldc.usb.ve/~teruel/ci4713/clases2001/testReqs.html.

35. —. 2001. Requerimientos de Pruebas. [Online] Enero 5, 2001. [Cited: Enero 11, 2008.] http://www.ldc.usb.ve/~teruel/ci4713/clases2001/testSpecs.html.

36. **UCI, Colectivo de Profesores**. 2007. Conferencia_Implementacion.pdf. Habana: Tele formación-UCI Tevé, 2007.

37. **Universidad Nacional Mayor De San Marcos**. Glosario de Términos. [Online] [Cited: Diciembre 13, 2007.] http://www.unmsm.edu.pe/ogp/ARCHIVOS/Glosario/inds.htm.

38. **V., María José Roca**. 2005. Pruebas de Integración de Productos: Un enfoque práctico. [Online] Julio 2005. [Cited: Febrero 19, 2008.] http://www.greensqa.com/archivos/Art01-PruebasIntegracionv1.pdf.

39. **Valencia, María Eugenia**. 2007. PRUEBA DE SOFTWARE. [Online] 2007. [Cited: Enero 26, 2008.] http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/PRUEBASoftware.pdf.

40. **Valentín Estrada, Kamilo and Valdivia Mola, Karelia**. 2007. Proceso de desarrollo basado en la Arquitectura Orientada a Servicios en el Proyecto APS. Habana: UCI, 2007.

41. **Vicente Toribio, Jose María**. 2005. Adictos al Trabajo. [Online] Abril 17, 2005. [Cited: Febrero 5, 2008.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jmeter.

42. **Vignaga, Daniel, Perovich Andrés**. Enfoque Metodológico para el Desarrollo Basado en Componentes. [Online] [Cited: Noviembre 19, 2007.] http://dis.um.es/~jsaez/dbc/curso0708/docs/art01_vp03.pdf.

Glosario de Términos:

Área Temática: Forma en que se agrupan varios proyectos productivos que poseen características comunes en cuanto al tema a desarrollar EJ (SAS) Sistemas de Apoyo a la Salud. Esta estructuración organiza mejor la producción en la Facultad 7.

Calidad: La capacidad de cumplir con las características inherentes de un producto, componente de producto, o proceso para satisfacer las exigencias de los clientes.

Complejidad Ciclomática: Es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa, da el límite superior del número de pruebas que se deben realzar.

Listas de Chequeo: son para chequear características generales del producto que no se verifican a través de las pruebas.

Polo Productivo*:* División Estructural de la Facultad 7 encaminado a organizar mejor la producción en cuanto a los temas a desarrollar. En estos momentos existen dos Polos productivos.

Procedimiento*:* Sucesión cronológica de operaciones concatenadas entre sí, que se constituyen en una unidad de función para la realización de una actividad o tarea específica dentro de un ámbito predeterminado de aplicación. Todo procedimiento involucra actividades y tareas del personal, determinación de tiempos de métodos de trabajo y de control para lograr el cabal, oportuno y eficiente desarrollo de las operaciones.

Proyecto productivo: todo proceso (o acción estratégica), encaminado a conseguir un objetivo previamente fijado, con unos recursos económicos y temporales limitados en los que la finalidad es desarrollar una actividad de tipo económico fundamentalmente caracterizada por la creación de productos (bienes y/o servicios).

Pruebas de Liberación: Conjunto de pruebas y revisiones desarrolladas por el grupo de calidad de la facultad y posteriormente por el equipo de calidad central de la universidad para garantizar la satisfacción del cliente

Requerimiento*:* Funcionalidad requerida por un usuario para resolver un problema o satisfacer uno o varios objetivos.

SW*: Software.* Es un término genérico que designa al conjunto de programas de distinto tipo (sistema operativo y aplicaciones diversas) que hacen posible operar con la computadora.

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

TIC*:* Tecnologías de la Información y las Comunicaciones.

UCI*:* Universidad de las Ciencias Informáticas.

Anexos

Anexo 1 Encuesta aplicada a los integrantes de los proyectos productivos de la Facultad 7 donde se desarrollan aplicaciones Web.

Encuesta a Realizar en los distintos Proyectos Productivos de la Facultad 7

Esta encuesta se realiza con el objetivo de conocer el comportamiento actual de la realización de pruebas durante el ciclo de vida en los distintos proyectos productivos de la facultad 7.

Nombre del proyecto_____ Área Temática_____

Rol del encuestado: _____

1. ¿Utiliza su proyecto alguna metodología de desarrollo? Si___ No___
2. Si la respuesta anterior es si, mencione cual_____
3. ¿Qué Estilo de Arquitectura utilizan?_____
4. ¿Es su aplicación Orientada a Servicios? Si___ No___
   a. ¿Porqué?_____
5. ¿Es su aplicación Basada en Componentes? Si___ No___
   a) ¿Porqué?_____
6. ¿Como considera la calidad de dicha aplicación?: E___ B___ R___ M___
7. ¿Existe en su proyecto un Grupo interno de Calidad? Si___ No___
8. En el proyecto se realizan pruebas de: ____ Caja negra _____ Caja blanca
9. Si se realizan pruebas de caja blanca diga que métodos utiliza: ___ Prueba del camino básico___ Prueba de condición ___Prueba de flujo de dato ___Prueba de bucles ___Otro ¿Cuál?_____
10. ¿Si se realizan pruebas de Caja negra, Cómo lo hacen? ____Validando entradas _____Diseñando casos de pruebas _____Otra Vía ¿Cuál?_____
11. ¿En su proyecto se aplica algún tipo de Listas de chequeo? Si___ No___
12. ¿En qué momento (Fase) realizan las pruebas en su proyecto? _____Inicio _____Elaboración ____Construcción _____Transición
13. ¿Cuántas veces? __Una __Dos__ En cada iteración __Las veces Necesarias

14. ¿Conoce alguna herramienta para la realización de pruebas? Si___ No___

15. Si la Respuesta es si

    ¿Se aplica Alguna en su proyecto? Si___ No____ _
    ¿Cuál(es)?_____

17. ¿El proyecto ha sido revisado por el equipo de calidad del software de la facultad o de la universidad? Si___ No___ No sé____

    Si su respuesta es Si

    ¿Considera que estas revisiones son necesarias?: Si___ No__

    ¿Por Qué?_____

18. ¿Sabe usted lo que es un plan de pruebas? Si___ No___ No sé____

19. Según sus conocimientos, las pruebas de aceptación las realiza:

    ___El desarrollador ___El grupo de calidad interno ___El usuario ___El grupo de calidad de la facultad ___El cliente final.

20. ¿En su proyecto se lleva algún tipo de registro de defectos detectados durante las pruebas realizadas? Si___ No__

Anexo 2 Lista de Chequeo para la revisión de los Requisitos Funcionales**.** Ejemplo de preguntas.

Nivel	Evaluación	Eval.	NP	Comentario
	Claridad ¿Los requisitos se escriben en lengua comprensible para el usuario/cliente?			
	¿Hay requisitos que tienen más de una interpretación?			
	¿Cada requisito característico del producto final se describe con una terminología única?			
	¿Hay un glosario en el cual los requisitos significativos y específicos están en términos definidos y claros?			
	¿Se entienden los requisitos para ponerlos en ejecución por un grupo independiente?			
	Completo ¿El requisito tiene un contenido específico?			
	Están especificados los cambios posibles a los requisitos.			

	La probabilidad de cambios está especificada para cada requisito.			
	Del contraste contra diversas fuentes no surgen errores u omisiones.			
	¿Se etiquetan todas las figuras, tablas y diagramas?			
	¿Se hace una remisión de todas las figuras, tablas y diagramas?			
	¿Se definen todos los términos en cada uno de los requisitos determinados?			
	¿Se ponen en un índice todos los términos identificados?			
	¿Todas las unidades de medida están bien definidas?			
	¿Existen áreas donde está incompleta la información debido a que el desarrollo aún no ha comenzado a especificarse?			
	¿La información que falta se define en el requisito?			
	¿Algún requisito necesita una especificación detallada?			
	¿Algún requisito necesita ser menos especificado?			
	¿Todos los requisitos se describen ellos mismos?			
	¿Están incluidos todos los requisitos relacionados con la funcionalidad?			
	¿Hay requisitos que produzcan inquietud?			
	¿Están incluidos todos los requisitos relacionados con el funcionamiento?			
	¿Están incluidos todos los requisitos relacionados con los apremios del diseño?			
	¿Están incluidos todos los requisitos relacionados con sus cualidades?			
	¿Están incluidos todos los requisitos relacionados con las interfaces externas?			
	¿Están incluidos todos los requisitos relacionados con las bases de datos?			
	¿Están incluidos todos los requisitos relacionados con el software?			
	¿Están incluidos todos los requisitos relacionados con la comunicación?			
	¿Están incluidos todos los requisitos relacionados con el hardware?			
	¿Están incluidos todos los requisitos relacionados con las entradas?			
	¿Están incluidos todos los requisitos relacionados con las salidas?			

	¿Están incluidos todos los requisitos relacionados con la divulgación?			
	¿Están incluidos todos los requisitos relacionados con la seguridad?			
	¿Están incluidos todos los requisitos relacionados con la capacidad de mantenimiento?			
	¿Están incluidos todos los requisitos relacionados con la instalación?			
	¿Están incluidos todos los requisitos relacionados con la criticabilidad?			
	¿Están incluidos todos los requisitos relacionados con las limitaciones de permanencia?			
	¿Se especifican los posibles cambios de los requisitos?			
	¿Se especifica para cada requisito la probabilidad del cambio?			
	Consistencia ¿Hay requisitos que describen el mismo objeto que estén en conflicto con otros requisitos con respecto a la terminología?			
	¿Hay requisitos que describen al mismo objeto que estén en conflicto con respecto a las características?			
	¿Hay requisitos que describan dos o más acciones que estén en conflicto temporalmente?			
	Cada frase aporta a la especificación.			
	Rastreabilidad ¿Los requisitos que fueron especificados por los usuarios fueron rastreados varias veces?			
	¿Los requisitos que están registrados en el documento o que fueron registrados por alguna persona fueron rastreados varias veces?			
	¿Todos los requisitos son detectados en el documento del diseño?			
	¿Los requisitos fueron rastreados en todos los módulos?			
	Comprobabilidad ¿Existen requisitos que sean imposibles de cumplir?			
	¿Para cada requisito hay un proceso que es ejecutado por un ser humano o una máquina para verificar el requisito?			
	¿Hay requisitos que sean expresados en términos comprobables en la última fase?			
	Modificabilidad ¿En el documento los requisitos se organizan de manera clara y lógica?			

	¿La organización está adherida a un estándar aceptado?			
	¿La redundancia es mínima y sólo se debe a distintos niveles de abstracción o detalle?			
	Contenido General ¿Cada requisito es relevante al problema y a su solución?			
	¿Está definido cualquier requisito en el diseño realmente?			
	¿Está definido cualquier requisito con detalle en la verificación?			
	¿Está definido cualquier requisito con detalle en la gerencia del proyecto?			
	¿Tiene una sección de introducción?			
	¿Tiene una sección de la descripción general?			
	¿Tiene una sección del alcance?			
	¿Tiene las definiciones, siglas, y una sección de las abreviaturas?			
	¿Tiene una sección específica de los requisitos?			
	¿Tiene una sección de la perspectiva del producto?			
	¿Tiene una sección de las funciones del producto?			
	¿Tiene una sección de las características del usuario?			
	¿Tiene una sección general de los apremios?			
	¿Tiene una sección de las dependencias?			
	¿Están todos los apéndices necesarios?			
	¿Están todas las tablas necesarias?			
	¿Están todos los diagramas necesarios?			
	Específico Entradas ¿Se especifican todas las fuentes de la entrada?			
	¿Se especifican todos los requisitos de la exactitud de la entrada?			
	¿Se especifican todas las frecuencias de la entrada?			
	¿Se especifican todos los formatos de la entrada?			

	¿Se especifican todos los tipos de valores en las salidas?			
	¿Se especifican todos los formatos de la salida?			
	¿Se especifican las frecuencias de las salidas?			
	Informes ¿Se especifican todos los formatos de informes?			
	¿Se especifican todos los informes que utilizan cálculos y fórmulas?			
	¿Se especifican todos los requisitos para filtrar los datos en el informe?			
	¿Se especifica todo para clasificar el informe?			
	¿Se especifica totalmente los requisitos del informe?			
	¿Se especifica todos los requisitos para el formato del informe?			
	Funciones ¿Son todas las funciones del software especificadas?			
	¿Todas las entradas se especifican para cada función?			
	¿Todos los aspectos del producto se especifican para cada función?			
	¿Todas las salidas se especifican para cada función?			
	¿Todos los requisitos de funcionamiento se especifican para cada función?			
	¿Está todo el diseño especificado para cada función?			
	¿Todas las cualidades se especifican para cada función?			
	¿Todos los requisitos de la seguridad especificados para cada función?			
	¿Todos los requisitos de la capacidad de mantenimiento se especifican para cada función?			
	¿Todos los requisitos de la base de datos se especifican para cada función?			
	¿Todos los requisitos operacionales se especifican para cada función?			
	¿Todos los requisitos de la instalación se especifican para cada función?			
	Interfaces Externas ¿Se especifican todas las interfaces utilizadas?			

	¿Se especifican todas las interfaces del hardware?			
	¿Se especifican todas las interfaces del software?			
	¿Se especifican todas las interfaces de comunicaciones?			
	¿Se especifican todos los requisitos del diseño de interfaz?			
	¿Se especifican todos los requisitos de la seguridad de interfaz?			
	¿Se especifican todos los requisitos de la capacidad de mantenimiento de la interfaz?			
	¿Se especifican todas las interfaces de interacción de humano – computador?			
	Interfaces Internas ¿Se han identificado todas las interfaces internas?			
	¿Se han especificado todas las características internas de las interfaces?			
	Sincronización ¿Se especifican todos los tiempos de transformación?			
	¿Se especifican todas las tarifas de transferencia de datos?			
	¿Se especifica todo el sistema con tarifas?			
	Confiabilidad ¿Las consecuencias de la falta de software se especifican para cada requisito?			
	¿Se especifica la información a proteger contra fallas?			
	¿Se especifica una estrategia para la detección de errores?			
	¿Se especifica una estrategia para la corrección?			
	Hardware ¿Se especifica la memoria mínima?			
	¿Se especifica el almacenamiento mínimo?			
	¿Se especifica la memoria máxima?			
	¿Se especifica el almacenamiento mínimo?			
	Software ¿Se especifica el software requerido y el sistema operativo?			

	¿Se especifican todas las unidades requeridas del software?			
	¿Se especifican todos los productos de software comparados que deben ser utilizados con el sistema?			
	Comunicaciones ¿Se especifica la red?			
	¿Se especifican los protocolos requeridos?			
	¿Se especifica la capacidad requerida de la red?			
	¿Se especifica la tarifa del rendimiento de procesamiento de la red, requerido y estimado?			
	¿Se especifica el mínimo estimado de las conexiones de red?			
	¿Se especifica los requisitos de funcionamiento mínimo de la red?			
	¿Se especifican los requerimientos de funcionamiento máximo de la red?			
	¿Se especifican los requerimientos de funcionamiento óptimos para la red?			
	Trazabilidad Cada requisito obedece a una necesidad específica de usuario.			
	Cada requisito tiene su origen en una fuente (documento o persona) específica.			
	Cada requisito se puede rastrear hacia delante su incorporación al diseño.			
	Cada requisito se puede rastrear hacia delante su incorporación en determinados módulos.			
	Verificabilidad Cada requisito es implementable.			
	Para cada requisito existe un procedimiento que, ejecutado por una persona o máquina, permite verificar si se cumple			
	Hay algún requisito que se va a expresar en términos verificables más adelante.			
	Priorización Cada requisito tiene asignado un nivel de prioridad asignado por el Cliente/Usuario para guiar las negociaciones/compromisos.			