UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 9

TÍTULO: Diseño y aplicación de pruebas al subsistema de Seguridad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICA

AUTOR(ES): Erieilis Reyes González

TUTOR(ES): Ing. Aliosmi López Velázquez

Ciudad de la Habana, julio, 1 2010.

"Año 52 de la Revolución".

"El futuro de Cuba tiene que ser necesariamente un futuro de hombres de ciencia, de hombres de pensamiento"

DEDICATORIA

Dedico mi tesis en especial a mi Mamá y mi Papá por darme la vida y siempre brindarme mucho amor, cariño, apoyo y confianza a pesar de todas las situaciones difíciles por las que pasé a lo largo de la carrera.

A mi abuela por el inagotable manantial de amor que me ha brindado siempre y por sus consejos y dedicación.

A mi hermana que desde que está en la escuela no ha dejado de estar a mi lado cuando la he necesitado y además por ser unas de las personas que más quiero en esta vida.

A mis tíos y abuelos por brindarme su apoyo.

A toda mi familia por confiar mucho en mí y por quererme tanto.

También a todos mis amigos, que por suerte son muchos, y a todos con los que he trabajado o compartido una tarea o un sueño.

AGRADECIMIENTOS

Agradezco a la Revolución y en especial a Fidel por darme la oportunidad de convertirme en una profesional.

A mis padres por su apoyo incondicional.

A mi hermana por alimentar mis aspiraciones de culminación de mis estudios.

A mis abuelitos por los consejos que me dieron.

A mi tía por estar pendiente de mí aunque esté lejos.

A mi amiga Yuneikys que a pesar de que siempre fui yo la que tuve que darle fuerzas cuando me toco a mí ablandarme ella estuvo allí para darme fuerzas.

A mis amigas: Katisleydis, Mariem y Yenisel, por estar junto a mí, pendiente de mis problemas.

A mi tutora ya que sin su ayuda no hubiera sido posible el desarrollo de este trabajo.

A todas aquellas personas que no menciono porque sería muy larga la lista pero que no por esto dejan de ser importantes para mí y les agradezco con el alma toda la ayuda brindada.

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Erieilis Reyes González

_____

Ing. Aliosmi López Velázquez

_____

DATOS DE CONTACTO

Ing. Aliosmi López Velázquez: Graduada en la especialidad de Ingeniera en Ciencias Informáticas. Actualmente ejerce como profesora de Ingeniería de Software en la Universidad de las Ciencias Informáticas.

RESUMEN

El propósito del presente trabajo de diploma es ejecutar el plan de prueba del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria, concretamente al módulo de Seguridad donde se gestionan todos los usuarios y sus roles.

Dentro de esas actividades estuvo la aplicación de las pruebas de Caja Blanca y Caja Negra para lo cual se diseñaron los casos de prueba utilizando las técnicas del Camino Básico y Partición Equivalente respectivamente. Además, se realizaron pruebas de Integración y de Sistema, con las técnicas de Integración Incremental Ascendente y Prueba de Stress respectivamente. Se archivaron los resultados obtenidos de la ejecución del proceso y se realizó una evaluación de cada parte probada.

Todo lo antes planteado garantiza un producto de software competitivo, con la calidad requerida y con un alto nivel técnico. Motiva un proceso de Gestión de Cambio paralelo al de prueba, mediante el cual se van mitigando los defectos resultantes del periodo de desarrollo. También sirve de guía para extender estas pruebas al resto de los módulos del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria.

PALABRAS CLAVES

Casos de prueba, plan de prueba, componente de prueba, calidad, estrategia.

ABSTRACT

The purpose of this dissertation is to implement the testing process of the Process Management System of the Department of University Television, specifically the Security module which handle all users and their roles. Are performed each workflow activities generated test artifacts resulting from each activity.

The process was carried out using test cases of white box and black box, designed using the techniques of basic roads and Equivalent Partition respectively, which formed the input for execution through a component interface implemented and tested for the module tested. Also being tested and system integration, with the techniques Incremental Integration and Testing Ascending Stress respectively. It filed the results of the implementation of process and an evaluation of each test part. The strategy and resources necessary to carry out this process planned in the Test Plan, establishing a basis for control and monitoring during implementation.

This process will ensure a competitive software product, with the required quality and high technical level. Motivate a change management process parallel to the testing process, through which will mitigate the resulting defects in the development process. Also serve as a guide to extend this process to the rest of the modules of Process Management System of the Department of University Television.

TABLAS Y FIGURAS

ÍNDICE

INTRODUCCIÓN

El desarrollo de la informática a nivel mundial es significativo. Desde los primeros pasos que llevaron al surgimiento de ese tema años atrás hasta el momento, han sido innumerables los avances que se han logrado en ese ámbito. Un ejemplo de ello es el surgimiento de los ordenadores, de Internet, y de disímiles dispositivos. Actualmente debido al logro que ha tenido la sociedad, ya casi todo se encuentra automatizado y se hace mediante computadoras.

Cuba no está exenta de este desarrollo tecnológico, se pueden apreciar pasos de avance en este sentido, con los múltiples programas de la Batalla de Ideas como por ejemplo los televisores (TV) y videos en las escuelas, las teles clases, los Joven Club de Computación y Electrónica y la Universidad de las Ciencias Informáticas (UCI). Además de la creación de algunas empresas para producir software entre muchos otros adelantos que demuestran que se va avanzando en la informatización de la sociedad cubana.

En el país se están efectuando inversiones para convertir la producción de software en uno de los renglones más importantes de la economía, por lo que la necesidad de obtener productos de software que sean desarrollados con profesionalidad va en aumento cada día. Lograr que tengan un alto nivel de calidad, que sean productos realmente competitivos, es la prioridad de los productores de software. Esto se puede lograr aplicando las mejores prácticas definidas profesionalmente, durante todo el proceso de desarrollo del software.

Estas buenas prácticas suponen la aplicación de las guías de procesos marcadas por las disposiciones que al nivel de organización, se han establecido, ya sea como un sistema de calidad bien definido o mediante una serie de procedimientos y estándares reglamentados, que es conocido como aseguramiento de la calidad del software. En todo caso, la medición del cumplimiento de las definiciones expresadas admite, las actividades de verificación y validación (básicamente, pruebas de software y actividades de revisión y auditoría). La medición puede favorecer tanto en el control de los procesos y actividades como el de los productos, para entender la situación de los mismos o para examinar si cumplen los requisitos pedidos o un cierto nivel de calidad.

Producir software con calidad, a un costo razonable, produce beneficios tanto para los clientes como para los desarrolladores. Para producir software con eficacia debe definirse y usarse un modelo, que no es más que un conjunto de buenas prácticas para el ciclo de vida del software, enfocado en los procesos de gestión y desarrollo de proyectos. Un cliente satisfecho con seguridad continuará necesitando más funciones y se le venderán más productos. En todo proceso de desarrollo de software los principales componentes son las

actividades que se llevan a cabo para certificar la calidad del software que se produce. A este conjunto de actividades y sus productos se le conoce como el Aseguramiento de la Calidad del Software (ACS). **(Luis Fernández Sanz)**

La Universidad de las Ciencias Informáticas no puede quedar fuera de esta premisa, ya que uno de sus propósitos es formar parte de la producción de software del país, y que éstos tengan la mejor calidad posible. Una de las opciones que brinda la UCI para la formación de sus profesionales es la utilización de teleclases o videoconferencias de las asignaturas que se imparten en la misma, esto marcó el origen de la Dirección de Televisión Universitaria (DTU), la cual fue creada en los inicios de la UCI debido a la necesidad de un sistema de comunicación que abarcara el campus universitario y llevara a los miembros de la comunidad universitaria en principio un conjunto de informaciones importantes para el desarrollo de la vida y la convivencia dentro de la Institución; inmediatamente esta idea fue desarrollándose y abarcando otras ramas de suma importancia como la docencia.

Actualmente en la DTU todo el trabajo se hace manualmente. Esto es un problema pues se debe trabajar con cierta cantidad de información en copia dura: dígase papeles, películas en casetes de video, además que se requiere de una buena comunicación entre los trabajadores. Esta y muchas otras problemáticas que fueron surgiendo originaron la idea de crear un sistema informático que brinde respuesta a las insuficiencias en los procesos de gestión que se llevan a cabo en la DTU, el mismo se denomina Sistema de Gestión de Procesos para la Dirección de Televisión Universitaria (SGP-DTU).

Para la elaboración de este producto software se creó un proyecto conformado por estudiantes y profesores de la UCI. El principal objetivo de los miembros del proyecto es la construcción de esa solución con eficiencia y calidad. Los proyectos de desarrollo de software tradicionalmente han padecido de problemas para garantizar la calidad, tanto en el propio proceso de desarrollo como en los productos que entregan. Esta problemática tiene su origen en las habituales desviaciones de plazos y esfuerzo sobre los valores previstos y en la frecuente aparición de fallos durante la implantación y operación de los productos resultantes.

Lo anteriormente planteado da lugar a ciertas dificultades, como es la escasa adherencia a los plazos y esfuerzos previstos, lo que denota o pone de manifiesto la baja acción de la calidad en el proceso de gestión, y la falta de calidad de los productos desarrollados: cuanto menor es ésta, mayor es el número de defectos y, consecuentemente, mayor será el número de fallos que aparecerán durante la ejecución del software. Cuanto más se logre evitar estos conflictos más se ganará en calidad del software. Un componente importante del Aseguramiento de la Calidad del Software son las actividades del proceso de verificación y validación (V&V) de

software que se realizan durante las diferentes fases que componen el ciclo de desarrollo de los sistemas. Algunos de los objetivos de este proceso son:

1. Verificar que los productos obtenidos en cada fase del ciclo de vida:
   - ✓ Cumplan con los requisitos de la fase anterior.
   - ✓ Satisfagan los estándares, prácticas y convenciones de la fase actual.
   - ✓ Establezcan las bases apropiadas para iniciar la siguiente fase del ciclo de vida.

2. Validar que el producto final cumpla con los requisitos del software establecidos. **(2010)**

La Validación y Verificación es un proceso de control que asegura que el software cumple con su especificación y satisface las necesidades del usuario. Muchas veces se confunde "verificación" con "validación". Boehm (1979) puso en claro con pocas palabras la diferencia:

- ✓ Validación: ¿estamos construyendo el producto correcto? Se ocupa de controlar si el producto satisface los requisitos.
- ✓ Verificación: ¿estamos construyendo correctamente el producto? Implica controlar que el producto se está desarrollando conforme a su especificación inicial.

Entre las técnicas del proceso de V&V se encuentran las pruebas de software, un conjunto de herramientas, técnicas y métodos que hacen a la excelencia del desempeño de un programa, así como también la mejor publicidad que una empresa dedicada a la producción de software pueda tener. Las técnicas para descubrir problemas en un programa son numerosas y van desde el descubrimiento por uso del ingenio del personal de prueba hasta novedosas herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad.

Lo más importante es tener un conocimiento amplio sobre las técnicas de pruebas del software que se pueden aplicar. Estas técnicas proporcionan una guía sistemática para diseñar los casos de pruebas. Es necesario saber cuál es la metodología idónea a aplicar, además de conocer las herramientas que se requieren para esta tarea.

Teniendo en consideración que el proyecto Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria tiene gran importancia para la DTU y es necesario que el mismo cuente con la calidad requerida en todo software, se decide aplicarle una estrategia de prueba para garantizar que se está teniendo como resultado un software con la calidad y nivel técnico requerido.

Luego de todo lo antes expuesto se plantea que el **problema** de la investigación consiste en que no se cuenta con un plan de pruebas para aplicar al proyecto SGP-DTU impidiendo la verificación y validación de los requisitos planteados por el cliente. El **objeto de estudio** de esta investigación está enmarcado en las pruebas para proyectos de software. El **campo de acción** es la aplicación de casos de prueba del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria. Se **defiende la idea** de que si se desarrolla correctamente el plan de pruebas al subsistema de Seguridad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria entonces se podrá garantizar que el sistema tendrá sus módulos correctamente implementados.

Se concibe como **objetivo general** aplicar el plan de pruebas al subsistema de Seguridad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria que permita certificar la calidad del producto.

Para dar cumplimiento a este objetivo se plantean las siguientes **tareas**:

1. Realizar una revisión bibliográfica acerca de los diferentes tipos de prueba.
2. Seleccionar de estas pruebas cuáles se le van a realizar al sistema durante todo el proceso de desarrollo.
3. Definir los procedimientos para cada una de las pruebas seleccionadas tanto las de caja blanca como las de caja negra.
4. Diseñar los casos de prueba para cada uno de los casos de uso del subsistema.
5. Confeccionar un cronograma con todas las pruebas que se van a aplicar.
6. Realizar las pruebas planificadas al subsistema.
7. Recolectar el resultado de las pruebas.

Resultados esperados:

1. Resultados de las pruebas realizadas al subsistema de seguridad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria.
2. Evaluación de los resultados.

3. Evaluación del flujo de prueba implementado.

Métodos Científicos Utilizados

Analítico Sintético: Está dado por el análisis de los documentos generados en el levantamiento y captura de requisitos, extrayendo y analizando los principales elementos relacionados con el objeto de estudio.

Técnicas de Investigación

Entrevista y/o Encuesta: Entrevista a la analista del módulo con el objetivo de obtener a través de una conversación planificada, información cualitativa de los eventos y fenómenos que puedan ocurrir en el sistema.

Análisis documental: Basado en la exploración de toda la documentación generada por los analistas, dígase especificación de casos de usos, glosario de términos, grafos de entidades, así como también las especificaciones de la Arquitectura establecidas para la realización de este sistema.

El contenido de este trabajo se encuentra estructurado de la siguiente manera:

En el capítulo 1, “Fundamentos Teóricos” se realiza un estudio de todos los aspectos teóricos relacionados con las pruebas que se aplican al software y se abordan los conceptos asociados a este proceso.

En el capítulo 2, “Plan de prueba” se define el Plan de prueba para el subsistema de seguridad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria en el cual se refleja la estrategia a aplicar a este sistema.

En el capítulo 3, “Diseño, ejecución y evaluación de las pruebas” se diseñan y ejecutan los casos de prueba para el subsistema de seguridad del Sistema de gestión de Procesos de la Dirección de Televisión Universitaria y finalmente se evalúan los resultados obtenidos durante la aplicación de los casos de prueba diseñados.

1

Capítulo

Fundamentación Teórica

Introducción

En el capítulo se describe el entorno que define la metodología de desarrollo de software definida en el proyecto para llevar a cabo las pruebas que garanticen la calidad del producto. Se introducen algunos conceptos y se profundiza en los orígenes del trabajo a realizar así como los tipos de pruebas y sus técnicas para verificar y validar el producto desarrollado.

1.1. Metodología de desarrollo de software

Las metodologías de desarrollo de software especifican un conjunto de criterios que rigen la forma en que se emplea la ingeniería del software en un proyecto productivo; o sea, es la base para la realización de un proyecto de software, la principal etapa para obtener los objetivos buscados con dicho proyecto. Que no se utilice la metodología adecuada en un proceso de desarrollo de un proyecto también garantiza con seguridad la poca o nula calidad del mismo.

La producción de un software con calidad involucra el uso de metodologías o estándares para el análisis, diseño, programación y prueba de este, que permitan equilibrar la estética de trabajo, para así lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, que a la vez realcen la productividad, tanto para la labor de desarrollo como para el control de la calidad del software. La idea es controlar de forma transparente el proceso de desarrollo completo, esencialmente permite producir sistemas en el tiempo planificado para este y con el costo estimado, esto es importantísimo pues en la mayoría de las veces se planea hacer algo que finalmente toma más tiempo de lo planeado. **(McCall.J, 1997)**

La metodología de desarrollo que se utilizó durante todo el proceso de construcción del proyecto Sistema de Gestión de Procesos de la Dirección de Televisión (SGP-DTU) fue la metodología RUP.

1.1.1. Proceso Unificado de Desarrollo (RUP)

Dentro de las metodologías fuertes la que más se enfatiza es el Proceso Unificado de Desarrollo (RUP). Esta es una metodología para la ingeniería de software que va más allá del solo análisis y diseño orientado a objetos para proveer una familia de técnicas que toleran el ciclo completo de desarrollo de software. El resultado es un proceso basado en componentes, dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental. **(2009)**

RUP se basa en casos de uso (Use Case) para figurar lo que se tiene y lo que se espera del software, está orientado a la arquitectura del sistema a implementar, justificándose de la mejor manera, apoyándose en UML. Para poder utilizar RUP antes hay que adecuarlo a las características de la empresa, y medir de manera exacta el tiempo, costos y todos los demás recursos involucrados en el proceso. **(2009)**

La estructura dinámica de RUP es la que permite que éste sea un proceso de desarrollo fundamentalmente iterativo, y en esta parte se ven inmersas las 4 fases por las que está compuesta: **(Scribd)**

- Inicio
- Elaboración
- Construcción
- Transición

Fase de Inicio: Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones posteriores. **(Scribd)**

Fase de elaboración: En la fase de elaboración se seleccionan los casos de uso que permiten definir la arquitectura base del sistema y se desarrollaran en esta fase, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar. **(Scribd)**

Fase de Desarrollo: El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requisitos pendientes, administrar los cambios de acuerdo con las evaluaciones realizadas por los usuarios. **(Scribd)**

Fase de Cierre: El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el

soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto. **(Scribd)**

Además RUP también cuenta con nueve disciplinas:

• Modelamiento del negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
• Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
• Análisis y diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
• Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
• Prueba (Testeo): Busca los defectos a los largo del ciclo de vida.
• Instalación: Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
• Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
• Administración de configuración y cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.
• Ambiente: Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización. **(Scribd)**

RUP brinda un escenario formidable para la elaboración del diseño de los casos de prueba ya que por ser una metodología guiada por casos de usos, en ellos se almacena detalladamente toda la información de las funcionalidades del sistema en general. Esto certifica que a la hora de confeccionar los diseños de casos de prueba no se quede ninguna funcionalidad sin probar, esta ventaja no la garantiza ninguna otra metodología, ya que carecen de documentación en gran escala. **(2009)**

RUP como metodología de desarrollo al recorrer cada fase, ya antes descrita, especifica una serie de disciplinas dentro de las cuales están las pruebas. Disciplina que está presente en cada una de las fases, alcanzando mayor peso en la fase de construcción. Esta disciplina tiene una gran importancia, ya que es la que va a probar la calidad del producto final. **(2009)**

1.2. Calidad del software. Conceptos y definiciones

Al conjunto de propiedades inherentes a un objeto que le conceden capacidad para satisfacer necesidades implícitas o explícitas se le denomina calidad. La calidad de un producto o servicio es la percepción que el cliente tiene del mismo, es una fijación mental del consumidor que asume conformidad con dicho producto o servicio y la capacidad del mismo para satisfacer sus necesidades. Por tanto, debe definirse en el contexto que se esté considerando, por ejemplo, la calidad del servicio postal, del servicio dental, del producto, de vida, etc. **(Ricardo, 2010)**

Se puede decir que en el mundo de la informática la calidad no es más que la "Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" según planteó R. S. Pressman (1992), además se pudiera tener en cuenta que puede ser "El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas" según ISO 8402 (UNE 66-001-92). Analizando ambos planteamientos, si un software es desarrollado por alguien que estudió y se preparó para eso, o sea, un profesional en esa rama, el mismo debe tener una calidad excelente, además para obtener esta calidad, el producto debe brindar a sus usuarios todas las posibilidades que estos pidieron al solicitar la creación del software. Siguiendo con el tema de la calidad, se debe tener en cuenta que para lograr esta calidad de la que se habla, se debe comprobar que el software en cuestión cumpla con los factores de calidad requeridos y definidos por la misma.

1.3. Factores en la calidad del software

El término calidad del software se interpreta de diferentes maneras. Una de las definiciones más difundidas de calidad es la planteada por McCall (1977), que especifica una serie de factores. Cada uno de esos factores los subdivide en criterios, teniendo asociado cada uno de ellos una métrica.

- ✓ Funcionalidad

Funcionalidad se refiere a la habilidad del sistema de realizar el trabajo para el cual fue concebido. Este factor de calidad tiene características asociadas como las que se refieren a las capacidades de un programa, la generalidad de las funciones y la seguridad del sistema. Tiene subcaracterísticas asociadas como la adecuación, quien se refiere al refinamiento de los grafos de secuencia. **(McCall.J, 1997)**

- ✓ Confiabilidad

Confiabilidad hace referencia a consistencia. Tiene subcaracterísticas, una de ellas es la exactitud, la cual representa la precisión de los cálculos y el control; tolerancia a fallos, estos son los atributos del software que proporcionan una estructura de módulos altamente independientes; correctitud; recuperabilidad, la que constituye la existencia de mecanismos o dispositivos de software para restablecer el nivel de desempeño y recuperar datos; exactitud de las salidas; tiempo medio de fallos; capacidad de recuperación ante fallas y capacidad de predicción entre otras. **(McCall.J, 1997)**

- ✓ Mantenibilidad

Mantenibilidad es la capacidad de someter a un sistema a reparaciones y evolución. Hace referencia a las propiedades internas de un producto. Presenta simplicidad, que se refiere a los atributos del software que posibilitan la implementación de funciones de la forma más comprensible posible. Concreción, quien se caracteriza por presentar atributos del software que posibilitan la implementación de una función con la menor cantidad de códigos posibles. Los atributos del software proporcionan explicaciones sobre la implementación de las funciones y proporcionan uniformidad en las técnicas y notaciones de diseño e implementación. **(McCall.J, 1997)**

- ✓ Portabilidad

Portabilidad es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos. Se va a caracterizar por atributos del software que proporcionan explicaciones sobre la implementación de las funciones; atributos del software que proporcionan módulos altamente independientes; características del software que determinan su independencia del entorno operativo y del hardware. **(McCall.J, 1997)**

- ✓ Eficiencia

Eficiencia se caracteriza por un conjunto de atributos con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas. Eficiencia hace alusión al comportamiento en el tiempo del software y al comportamiento de recursos, además de atributos que minimizan el espacio de almacenamiento necesario. **(McCall.J, 1997)**

Siempre se debe tener en cuenta, que en dependencia del sistema que se esté probando, va a estar determinada la prioridad de los factores a evaluar y la importancia de los mismos para determinar la calidad

final. Después de haber visto y analizado estos factores, se debe trazar la estrategia con la que se va a trabajar para comprobar dicha calidad.

1.4. 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.
Las características generales son:

- 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 y la depuración son actividades diferentes.
- La prueba la realiza la persona que desarrolla el software y (para grandes proyectos), un grupo de pruebas independiente. **(Granada.)**

Objetivos de la estrategia de prueba:

- 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 probados de nuevo y posiblemente devueltos a otra etapa, como diseño o implementación, de forma que los defectos puedan ser arreglados. **(Granada.)**

Luego de ver cómo se debe y cuál es la importancia de que se trace una estrategia de prueba para cada software, es preciso estudiar y profundizar en; qué son las pruebas y cuáles son los diferentes tipos de estas que se pueden aplicar según el proyecto que se está probando.

1.5. Pruebas de software

Las pruebas constituyen una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requisitos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. **(Booch, 2004)**

Una buena prueba según Kaner[1], Falk y Nguyen posee los siguientes atributos:

1. Tiene una alta probabilidad de encontrar un error.
2. No debe ser redundante.
3. Debería ser "la mejor de la cosecha".
4. No debería ser ni demasiado sencilla ni demasiado compleja.

El objetivo de la fase de pruebas de un programa es el de detectar todo posible malfuncionamiento antes de que entre en producción. Un error detectado en el laboratorio puede ser costoso de reparar; pero siempre es peor que el error le aparezca al usuario final. En esta idea, las pruebas serán de mayor calidad cuantos menos errores queden por descubrir tras haberla pasado. Y, viceversa, si un programa aún tiene muchos fallos tras haber superado un conjunto de pruebas, se dirá que las mismas son de poca calidad. Si se pudiera probar un programa con todos los posibles datos de entrada, se tendrían las pruebas perfectas, pues no hay lugar para las sorpresas. Lamentablemente, casi nunca es posible probar con todos los casos. En consecuencia, se necesita un criterio para elegir qué casos se prueban. **(Ricardo, 2010)**

1.6. Tipos de pruebas

Hay dos formas de probar cualquier producto construido (y casi cualquier cosa): 1) Si se conoce la función específica para la que se diseñó el producto, se aplican pruebas, que demuestren que cada función es plenamente operacional, mientras se buscan los errores de cada función; 2) si se conoce el funcionamiento interno del producto, se aplican pruebas para asegurarse de que todas las piezas encajan; es decir, que las operaciones internas se realizan de acuerdo con las especificaciones y que se han probado todos los componentes internos de manera adecuada. El primer término se refiere a las pruebas de caja negra y el segundo a las de caja blanca. A continuación se hace un análisis más detallado de cada una de estas pruebas. **(Pressman, 2005)**

[1] Cem Kaner J.D Profesor de Tecnología de dotación lógica en el Instituto de la Florida de la tecnología, y director del centro del Tech de la Florida para la educación y la investigación de prueba (CSTER) del software desde 2004. Trabajó en la industria del software que comenzaba en 1983 en Silicon Valley.

o **Pruebas de caja blanca:** se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado. Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. **(Pressman, 2005)**

Métodos de prueba basados en caja blanca

- **La prueba del camino básico:** Esta prueba permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

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

- **La prueba de flujo de datos**: Se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

- **La prueba de bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles. **(Pressman, 2005)**

o **Prueba de caja negra:** se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. **(Pressman, 2005)**

Técnicas para desarrollar la prueba de caja negra:

- **Técnica de Partición de Equivalencia:** esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

- **Técnica del Análisis de Valores Límites:** esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

- **Técnica de Grafos de Causa-Efecto:** es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones. **(Pressman, 2005)**

- **Pruebas 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. **(Granada.)**

Más adelante, cuando el módulo parece presentable, se entra en una fase de prueba sistemática. En esta etapa se empieza a buscar fallos siguiendo algún criterio para que "no se escape nada". Los criterios más habituales son los denominados de caja negra y de caja blanca.

- **Prueba de integración:** El objetivo es seleccionar los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. El objetivo principal es detectar las fallas de interacción entre las distintas clases que componen al sistema.

Se llama integración incremental cuando el programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y corregir, es más probable que se pueda probar completamente las interfaces y se pueda aplicar un enfoque de prueba sistemática. Hay dos estrategias de integración incremental:

1. **Integración Descendente (Top-Down):**

Se integran los módulos moviéndose hacia abajo por la jerarquía de control. Comenzando por el módulo principal, los módulos subordinados se van incorporando a la estructura, primero en profundidad, que integra todos los módulos de un camino de control principal de la estructura, o primero en anchura, que incorpora todos los módulos directamente subordinados a cada nivel, moviéndose por la estructura de forma horizontal. **(Pressman, 2005)**

Este proceso se realiza en una serie de cinco pasos:

1. Se usa el módulo de control principal como controlador de la prueba, disponiendo de resguardos para todos los módulos directamente subordinados al módulo de control principal.
2. Dependiendo del enfoque de integración elegido se van sustituyendo los resguardos subordinados uno a uno por los módulos reales.
3. Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.
4. Tras terminar cada conjunto de pruebas, se reemplaza otro resguardo con el módulo real.
5. Se hace la prueba de regresión para asegurarse de que no se han introducido errores nuevos.

El programa continúa desde el paso 2 hasta que se haya construido la estructura del programa entero. Al aplicar esta estrategia pueden surgir algunos problemas, el más común se da cuando se requiere un proceso de los niveles más bajos de la jerarquía para poder probar adecuadamente los niveles superiores. Al principio

de la prueba descendente, los módulos de bajo nivel se reemplazan por resguardos; por tanto, no pueden fluir datos significativos hacia arriba por la estructura del programa. Para solucionar esto se tienen tres opciones:

- Retrasar muchas de las pruebas hasta que los resguardos sean reemplazados por los módulos reales.
- Desarrollar resguardos que realicen funciones limitadas que simulen los módulos reales.
- Integrar el software desde el fondo de la jerarquía hacia arriba.

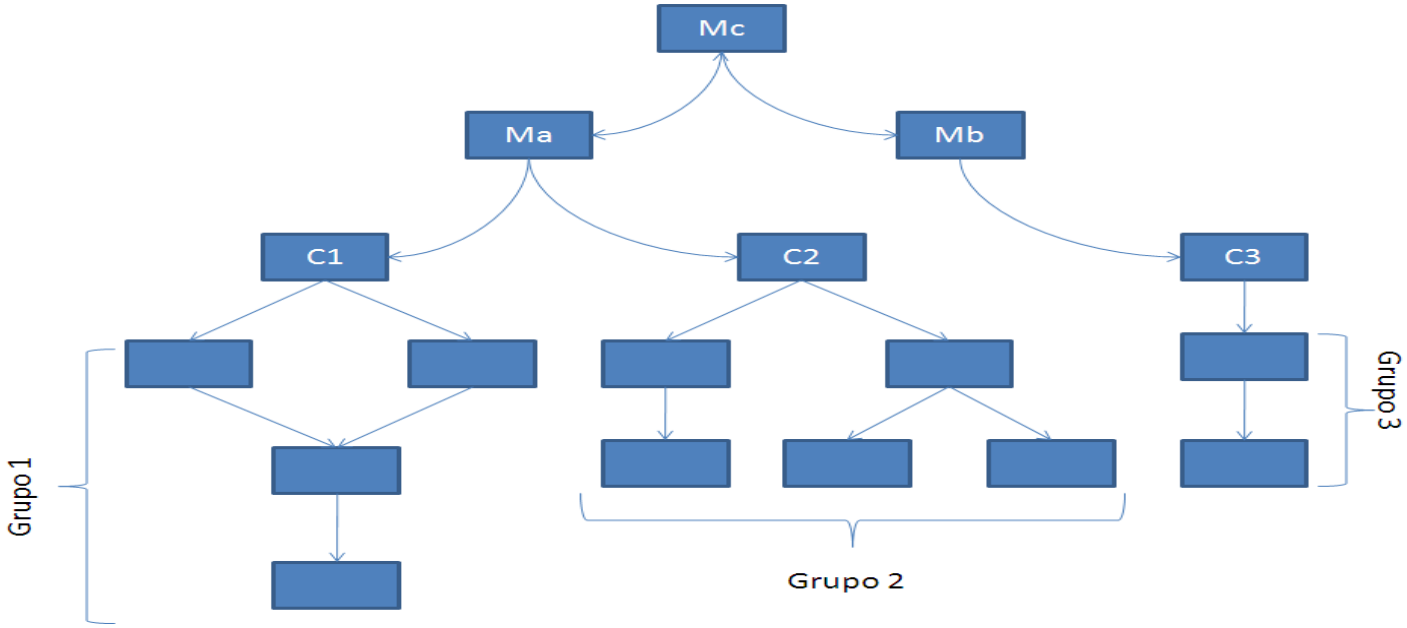

Figura 1. Integración Descendente

2. Integración Ascendente (Bottom-Up):

Empieza la construcción y la prueba con los módulos de los niveles más bajos de la estructura del programa. Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados a un nivel dado siempre está disponible y se elimina la necesidad de resguardos. **(Pressman, 2005)**

Se puede implementar una estrategia de integración ascendente mediante los siguientes pasos:

1. Se combinan los módulos de bajo nivel en grupos que realicen una subfunción específica del software.
2. Se escribe un controlador 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.

A medida que la integración progresa disminuye la necesidad de controladores de prueba diferentes.
La selección de una estrategia de integración depende de las características del software y de la planificación del proyecto.

Una buena alternativa es usar una mezcla de las dos estrategias (Ascendente y Descendente) que use la descendente para los niveles superiores de la estructura, junto con la ascendente para los niveles subordinados.

A medida que progresa la prueba de integración, se deben identificar los módulos críticos. Un módulo crítico es aquel que tiene una o más de las siguientes características:

- Está dirigido a varios requisitos del software.
- Tiene un mayor nivel de control.
- Es complejo o propenso a errores.
- Tiene unos requisitos de rendimiento muy definidos.

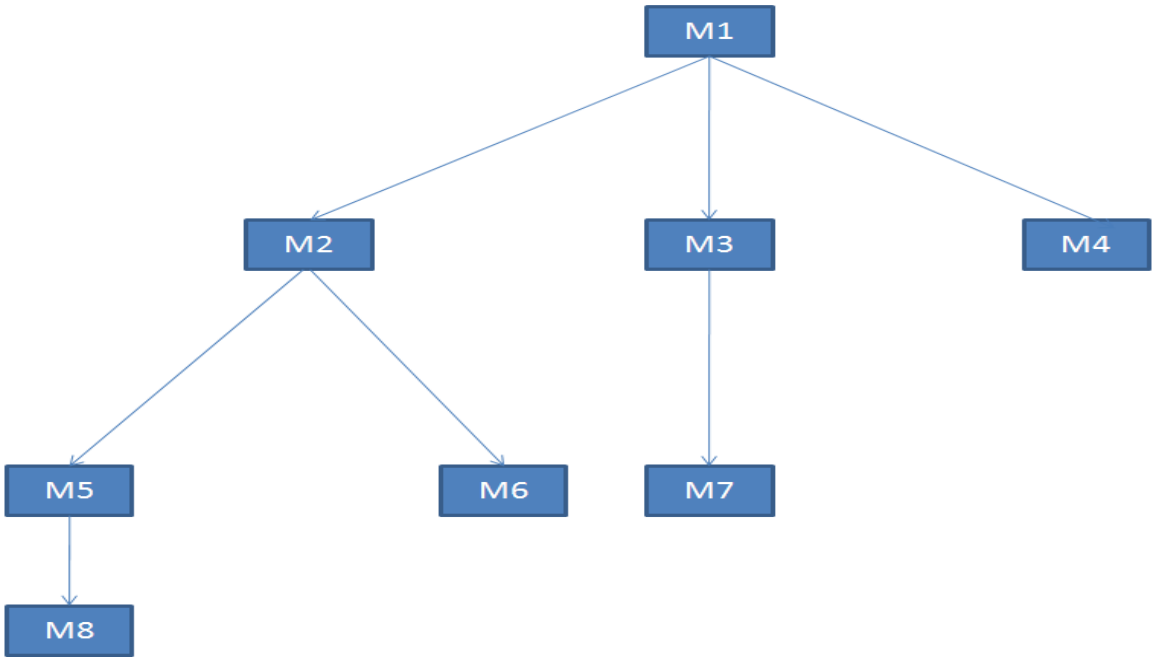

Figura 2: Integración Ascendente

Una estrategia de prueba para el software debe constar de pruebas de bajo nivel, así como de pruebas de alto nivel.

Pruebas de Alto Nivel

- **Prueba 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:

  - **Prueba de recuperación:** Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.

  - **Prueba de seguridad:** se determinan los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a datos.

  - **Prueba de resistencia:** determinan hasta donde puede soportar el programa determinadas condiciones extremas.

  - **Prueba de rendimiento:** determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones, etc.

  - **Prueba de instalación:** Se centra en asegurar que el sistema software desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepcionales, por ejemplo con espacio de disco insuficiente o continúas interrupciones. **(Granada.)**

- **Prueba de validación:** El software totalmente ensamblado se prueba como un todo para comprobar si cumple los requisitos funcionales y de rendimiento, facilidad de mantenimiento, recuperación de errores, etc.

Las pruebas de validación en la ingeniería de software son el proceso de revisión que el sistema de software producido cumple con las especificaciones y que cumple su cometido. Es normalmente una parte del proceso de pruebas de software de un proyecto, que también utiliza técnicas tales como evaluaciones, inspecciones, y tutoriales. La validación es el proceso de comprobar lo que se ha especificado es lo que el usuario realmente quiere, o sea, son los requisitos del sistema.

Existen dos tipos de requisitos que deben tomarse en cuenta en la planificación de las pruebas de validación. Los primeros son los requisitos funcionales, tomados a partir del modelo de casos de uso. Los segundos son los llamados no-funcionales, entre los cuales se puede mencionar: (**Scribd)**

- Accesibilidad
- Disponibilidad
- Eficacia (funcionamiento que resulta en lo referente a esfuerzo)

- Capacidad de mantenimiento
- Funcionamiento /Tiempo de reacción
- Confiabilidad
- Resistencia
- Robustez
- Seguridad
- Compatibilidad
- Estabilidad
- Soportabilidad

La validación del software se lleva a cabo a través de un proceso denominado pruebas alfa y beta, para descubrir errores que surgen con mayor facilidad bajo la operación del usuario final.

Prueba alfa: es realizada por un usuario en el lugar de desarrollo del software. De manera que es el cliente el que utiliza el software en la forma más natural posible, y el implementador sólo observa y registra cualquier error o problema de uso, todo ello en un entorno controlado.

Prueba beta: se lleva a cabo con los usuarios en el sitio real donde será destinado el software como producto final; estos sitios se conocen como beta sites. En estas pruebas, normalmente, el implementador no está presente, es decir, el software trabaja en un ambiente que no puede ser controlado por el equipo de desarrollo. El cliente registra todo lo que considere como problemas del software y lo reporta a intervalos regulares al equipo de desarrollo.

Los beta sites por lo general son clientes dispuestos a correr los riesgos de aceptar un sistema que todavía no ha sido fielmente comprobado, a cambio de ventajas estratégicas para la empresa

1.7. Otros tipos de pruebas:

~ **Recorridos (walkthroughs):** Consiste en sentar alrededor de una mesa a los desarrolladores y a una serie de críticos, bajo las órdenes de un moderador que impida un recalentamiento de los ánimos. El método consiste en que los revisores se leen el programa línea a línea y piden explicaciones de todo lo que no está meridianamente claro. Puede que simplemente falte un comentario explicativo, o que detecten un error auténtico o que simplemente el código sea tan complejo de entender/explicar que más vale que se rehaga de forma más simple. Para un sistema complejo pueden hacer falta muchas sesiones. Esta técnica es muy eficaz

localizando errores de naturaleza local; pero falla estrepitosamente cuando el error deriva de la interacción entre dos partes alejadas del programa. Nótese que no se está ejecutando el programa, sólo mirándolo con lupa, y de esta forma sólo se ve en cada instante un trocito del listado. **(Mañas, 1994)**

~ **Aleatorias (random testing):** Ciertos autores consideran injustificada una aproximación sistemática a las pruebas. Alegan que la probabilidad de descubrir un error es prácticamente la misma si se hacen una serie de pruebas aleatoriamente elegidas, que si se hacen siguiendo las instrucciones dictadas por criterios de cobertura (caja negra o blanca). Como esto es muy cierto, probablemente sea muy razonable comenzar la fase de pruebas con una serie de casos elegidos al azar. Esto pondrá de manifiesto los errores más patentes. No obstante, pueden permanecer ocultos errores más sibilinos que sólo se muestran ante entradas muy precisas. Si el programa es poco crítico (una aplicación personal, un juego,...) puede que esto sea suficiente. Pero si se trata de una aplicación militar o con riesgo para vidas humanas, es de todo punto insuficiente. **(Mañas, 1994)**

~ **Solidez (robustness testing):** Se prueba la capacidad del sistema para salir de situaciones embarazosas provocadas por errores en el suministro de datos. Estas pruebas son importantes en sistemas con una interfaz al exterior, en particular cuando la interfaz es humana.

Por ejemplo, en un sistema que admite una serie de órdenes (commands) se deben probar los siguientes extremos:

- Órdenes correctas, todas y cada una.
- Órdenes con defectos de sintaxis, tanto pequeñas desviaciones como errores de bulto.
- Órdenes correctas, pero en orden incorrecto, o fuera de lugar.
- La orden nula (línea vacía, una o más).
- Órdenes correctas, pero con datos de más.
- Provocar una interrupción (BREAK, ^C, o lo que corresponda al sistema soporte) justo después de introducir una orden.
- Órdenes con delimitadores inapropiados (comas, puntos,...).
- Órdenes con delimitadores incongruentes consigo mismos (por ejemplo, esto]. **(Mañas, 1994)**

~ **Aguante (stress testing):** En ciertos sistemas es conveniente saber hasta dónde soportan, bien por razones internas (¿hasta cuántos datos podrá procesar?), bien externas (¿es capaz de trabajar con un disco al 90%?, ¿aguanta una carga de la CPU del 90 %?, etc.) **(Mañas, 1994)**

~ **Conformidad u Homologación (conformance testing):** En programas de comunicaciones es muy frecuente que, además de los requisitos específicos del programa que estamos construyendo, aparezca alguna norma más amplia a la que el programa deba atenerse. Es frecuente que organismos internacionales como ISO[2] y el CCITT[3] elaboren especificaciones de referencia a las que los diversos fabricantes deben atenerse para que sus ordenadores sean capaces de entenderse entre sí.

Las pruebas, de caja negra, que se le pasan a un producto para detectar discrepancias respecto a una norma de las descritas en el párrafo anterior se denominan de conformidad u homologación. Suelen realizarse en un centro especialmente acreditado y, si se pasan satisfactoriamente, el producto recibe un sello oficial que dice: "homologado". **(Mañas, 1994)**

~ **Interoperabilidad (interoperability tesing):** En el mismo escenario del punto anterior, programas de comunicaciones que deben permitir que dos ordenadores se entiendan, aparte de las pruebas de conformidad se suelen correr una serie de pruebas, también de caja negra, que involucran 2 o más productos, y buscan problemas de comunicación entre ellos. **(Mañas, 1994)**

~ **Regresión (regression testing):** Todos los sistemas sufren una evolución a lo largo de su vida activa. En cada nueva versión se supone que o bien se corrigen defectos, o se añaden nuevas funciones, o ambas cosas. En cualquier caso, una nueva versión exige una nueva pasada por las pruebas. Si éstas se han sistematizado en una fase anterior, ahora pueden volver a pasarse automáticamente, simplemente para comprobar que las modificaciones no provocan errores donde antes no los había.

El mínimo necesario para usar unas pruebas en una futura revisión del programa es una documentación muy clara.

Las pruebas de regresión son particularmente espectaculares cuando se trata de probar la interacción con un agente externo. Existen empresas que viven de comercializar productos que "graban" la ejecución de una prueba con operadores humanos para luego repetirla cuantas veces haga falta "reproduciendo la grabación". Y,

[2] ISO: Organización Internacional para la Estandarización. Organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica.

obviamente, deben monitorear la respuesta del sistema en ambos casos, compararla, y avisar de cualquier discrepancia significativa. **(Mañas, 1994)**

~ **Mutación (mutation testing):** Es una técnica curiosa, consistente en alterar ligeramente el sistema bajo pruebas (introduciendo errores) para averiguar si la batería de pruebas es capaz de detectarlo. Si no, más vale introducir nuevas pruebas. Todo esto es muy laborioso y francamente artesano. **(Mañas, 1994)**

Todas estas pruebas al aplicarlas proporcionan la información que se necesita para saber si se ha terminado o no con el software que se está construyendo. Por supuesto cada una de ellas está destinada a una parte específica del sistema que se va a probar, y esto va a devolver como resultado si existen errores o no en el mismo.

Conclusiones parciales

Se puede afirmar que para que un producto software cumpla con las especificaciones para las que fue creado debe ser evaluado siguiendo determinados atributos de calidad. Estos atributos son determinados en función de las características de la solución de software a desarrollar. Para validar el cumplimiento con las especificaciones y los atributos antes mencionados es necesario conocer los diferentes tipos de pruebas y técnicas para asegurar la calidad del software. En este capítulo se han mencionado los factores a tener en cuenta para medir la calidad de un sistema, también se han mencionado los diferentes tipos de pruebas existentes y como deben ser usadas para obtener un resultado óptimo y favorable.

[3] CCITT**:** Comité Consultivo de Telegrafía y Telefonía. Organización que establece estándares internacionales sobre telecomunicaciones. Actualmente es conocido como ITU.

Capítulo

Tendencias y tecnologías actuales a aplicar.

Introducción

La proyección de los procesos de prueba es una fase de trascendental importancia, ya que además de precisar la estrategia y determinar los recursos necesarios, permite ejecutar apreciaciones que constituyen una base para el control y seguimiento mientras el proceso es ejecutado. Implanta un referente para próximos proyectos y además brinda información para efectuar el análisis de los procesos para identificar aspectos de perfección. La construcción de un buen plan de prueba es la piedra angular y en efecto uno de los principales factores críticos de un proceso de prueba que permite entregar un software de mejor nivel. Es importante no hacer los planes de prueba suponiendo que prácticamente no hay defectos en los programas, y por lo tanto dedicando pocos recursos a las pruebas, pues siempre hay defectos.

2.1 Pruebas a aplicar

- **Pruebas de Unidad**
  - ✓ **De Caja blanca**
    - **Camino básico**

La prueba del camino básico es una técnica de prueba de la Caja Blanca propuesta por Tom McCabe. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de

flujo asociado al código y se calcula su complejidad ciclomática. Se identifican los nodos y según las instrucciones serán las aristas que conecten a esos nodos que son los que permiten dibujar el grafo de flujo. Los pasos que se siguen para aplicar esta técnica son los siguientes:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo de flujo.
3. Se determina el conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

- **Notación de Grafo de Flujo.**

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo.
Cada nodo del grafo corresponde a una o más sentencias de código fuente. Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.
Para construir el grafo se debe tener en cuenta la notación para las instrucciones. Figura 3 y Figura 4.

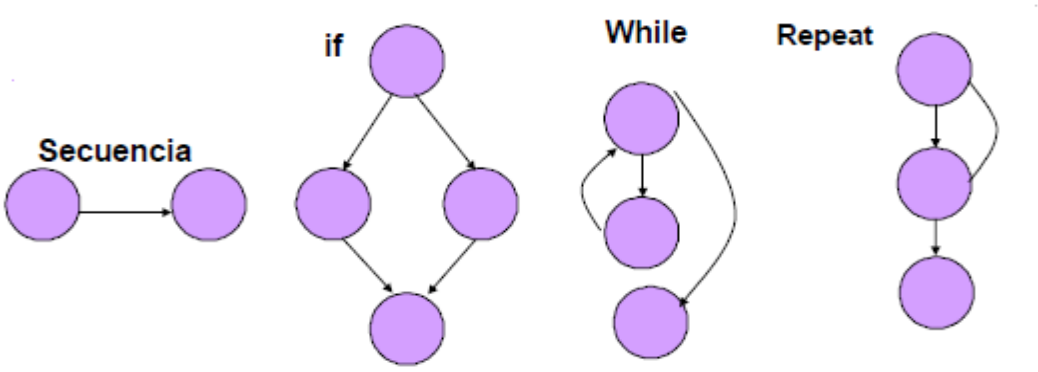

Figura 3: Notación de grafos de flujo para las instrucciones: Secuenciales, If, While

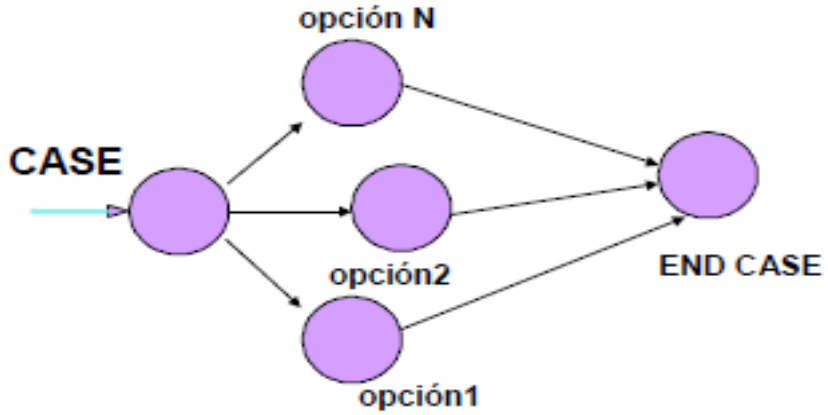

Figura 4: Notación de grafos de flujo para la instrucción Case

Un ejemplo de representación de Grafo de Flujo es el mostrado en la Figura 5 en el cual aparecen sus componentes:

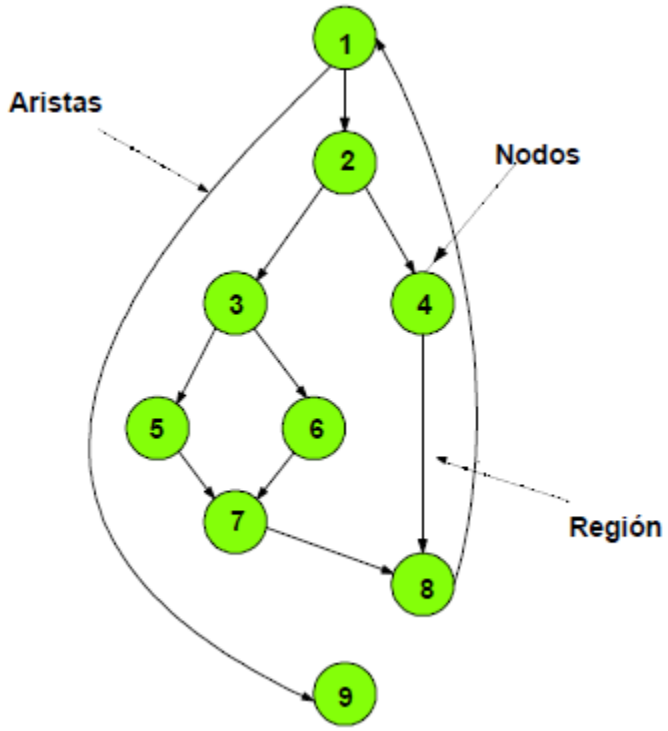

Figura 5: Características de los grafos

Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo. Cuando en un diseño se encuentran condiciones compuestas (uno o más operadores AND, NAND, NOR lógicos en una sentencia condicional), la generación del grafo de flujo se hace un poco más complicada.

Un Grafo de Flujo está formado por 3 componentes fundamentales que ayudan a su elaboración, comprensión y nos brinda información para confirmar que el trabajo se está haciendo adecuadamente.

Los componentes son:

- ✓ Nodo: Un nodo en el Grafo de Flujo es representado con un círculo y el mismo representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser también que haya nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

- ✓ Aristas: Las flechas del grafo se denominan aristas y representan el flujo de control, son análogas a las representadas en un grafo de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

- ✓ Regiones: Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran y la cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa esta es una de las formas de calcular la complejidad ciclomática.

  - **Complejidad Ciclomática.**

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica del programa. 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 ejecute cada sentencia al menos una vez.
Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente.
Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo:
1. El número de regiones del grafo coincide con la complejidad ciclomática, V (G).
2. La complejidad ciclomática, V (G), de un grafo de flujo G se define como V (G) = Aristas – Nodos + 2
3. La complejidad ciclomática, V (G), de un grafo de flujo G se define como V (G) = Nodos Predicado + 1
La Figura 6 representa, por ejemplo, las cuatro regiones del grafo de flujo.

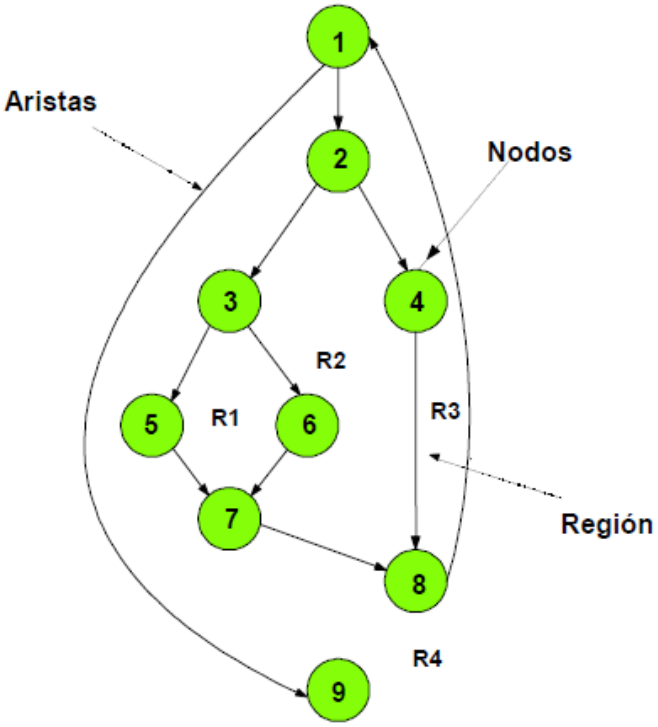

Figura 6: Número de regiones del grafo de flujo

- **Determinar el conjunto básico de caminos independientes**

En términos de 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. En la identificación de los distintos caminos de un programa para probar se debe tener en cuenta que cada nuevo camino debe tener el mínimo número de sentencias nuevas o condiciones nuevas respecto a los que ya existen. De esta manera, se intenta que el proceso de depuración sea más sencillo.

El conjunto de caminos independientes de un grafo no es único. No obstante, a continuación, se muestran algunas heurísticas para identificar dichos caminos:

(a) Elegir un camino principal que represente una función válida que no sea un tratamiento de error. Debe intentar elegirse el camino que atraviese el máximo número de decisiones en el grafo.

(b) Identificar el segundo camino mediante la localización de la primera decisión en el camino de la línea básica alternando su resultado mientras se mantiene el máximo número de decisiones originales del camino inicial.

(c) Identificar un tercer camino, colocando la primera decisión en su valor original a la vez que se altera la segunda decisión del camino básico, mientras se intenta mantener el resto de decisiones originales.

(d) Continuar el proceso hasta haber conseguido tratar todas las decisiones, intentando mantener como en su origen el resto de ellas.

Este método permite obtener V (G) caminos independientes cubriendo el criterio de cobertura de decisión y sentencia.

- **Derivación de casos de prueba.**

Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Se selecciona el método de camino básico como prueba de caja blanca ya que este permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizarán que durante la prueba se ejecute por lo menos una vez cada sentencia del programa, a diferencia de la prueba del flujo de datos que selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y el uso de las variables del programa, y la Prueba de bucles que se centra exclusivamente en la validez de las construcciones de bucles (simples, anidados, concatenados y no estructurados). O sea, que la prueba más completa y la que mejor resultados brinda es la de camino básico.

- **Pruebas de Aceptación (validación)**
  - ✓ **De caja negra**
    - **Partición equivalente**

Una partición equivalente es una técnica de prueba de Caja Negra que divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba para la partición equivalente se basa en la evaluación de las clases de equivalencia.

El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices:

- ✓ Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.
- ✓ Si una entrada requiere un valor concreto, aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.

- ✓ Si una entrada requiere un valor de entre los de un conjunto, aparecen 2 clases de equivalencia: en el conjunto o fuera de él.
- ✓ Si una entrada es booleana, hay 2 clases: sí o no.

Los mismos criterios se aplican a las salidas esperadas: hay que intentar generar resultados en todas y cada una de las clases.

Aplicando estas directrices se ejecutan casos de pruebas para cada elemento de datos del campo de entrada a desarrollar. Los casos se seleccionan de forma que ejerciten el mayor número de atributos de cada clase de equivalencia a la vez.

Para aplicar esta técnica de prueba se tienen en cuenta los siguientes pasos:

Primeramente se deben identificar las clases de equivalencia lo cual se hace tomando cada condición de entrada y aplicándole las directrices antes expuestas, Tabla 2.

Condición externa	Clases de equivalencia válidas (CEV)	Clases de equivalencia Inválidas (CEI)
Condición de entrada	Clases válidas para esa condición de entrada	Clases inválidas para esa condición de entrada

Tabla 1: Características de los grafos

Para definir las clases de equivalencia hace falta tener en cuenta un conjunto de reglas:

- ✓ Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y 2 inválidas.
- ✓ Si una condición de entrada especifica la cantidad de valores, identificar una clase de equivalencia válida y dos inválidas.
- ✓ Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, identificar una clase válida para cada uno de ellos y una clase inválida.
- ✓ Si una condición de entrada especifica una situación de tipo “debe ser”, identificar una clase válida y una inválida.
- ✓ Si existe una razón para creer que el programa no trata de forma idéntica ciertos elementos pertenecientes a una clase, dividirla en clases de equivalencia menores.

Luego de tener las clases válidas e inválidas definidas, se procede a definir los casos de pruebas, pero para ello antes se debe haber asignado un identificador único a cada clase de equivalencia. Luego entonces se pueden definir los casos teniendo en cuenta lo siguiente:

a. Escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.

b. Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de pruebas.

Con la aplicación de esa técnica se obtiene un conjunto de pruebas que reduce el número de casos de pruebas y dice algo sobre la presencia o ausencia de errores. A menudo, se plantea que las pruebas a los software nunca terminan, simplemente se transfiere del desarrollador al cliente. Cada vez que el cliente usa el programa está llevando a cabo una prueba. Aplicando el diseño de casos de pruebas al software en cuestión se puede conseguir una prueba más completa y descubrir y corregir el mayor número de posibles.

Se escoge como técnica de caja negra a aplicar la de partición de equivalencia pues esta divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. Sin embargo, la técnica de análisis de valores límite lleva a una elección de casos de prueba que ejerciten los valores límites y la Gráfica Causa-efecto representa una ayuda gráfica en seleccionar, de una manera sistemática, un gran conjunto de casos de prueba. Tiene un efecto secundario beneficioso en precisar estados incompletos y ambigüedades en la especificación.

- **Pruebas de Integración**

o Integración incremental ascendente: es donde la construcción del diseño empieza desde los módulos más bajos hacia arriba (módulo principal), el procesamiento requerido de los módulos subordinados siempre está disponible y elimina la necesidad de resguardo. La sección de una estrategia de integración depende de las características del software y, a veces, del plan del proyecto, en algunos de los casos se puede combinar ambas estrategias.

Ventajas de la integración incremental ascendente:

- Las entradas para las pruebas son más fáciles de crear ya que los módulos inferiores suelen tener funciones más específicas.

- Es más fácil la observación de los resultados de las pruebas puesto que es en los módulos inferiores donde se elaboran.
- Resuelve primero los errores de los módulos inferiores que son los que acostumbran a tener el procesamiento más complejo, para luego nutrir de datos al resto del sistema.

Luego de ver todas las pruebas que serán aplicadas es necesario conocer también los procedimientos a seguir en cada una de ellas.

➢ **Pruebas de Sistema**

o Prueba de stress: El "Stress Test" es una prueba que mide el comportamiento del sistema bajo una cierta demanda concurrente de conexiones. Esta es una de las pruebas clave que se deben realizar durante el ciclo de vida del software para garantizar que nuestro sistema va a cumplir con las expectativas previstas cuando sea implementado en producción. Es aquella que exige al sistema al máximo punto para poder medir sus capacidades y las condiciones en las cuales trabaja realizando una cantidad definida de peticiones y procesos. Están enfocadas a evaluar como el sistema responde bajo condiciones anormales, (extrema carga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponibles.)

2.2. Procedimientos para cada una de las pruebas seleccionadas

➢ **Pruebas de Unidad**

✓ **De Caja blanca**

o **Camino básico**

Procedimiento:

1. Identificar dentro del código a revisar los nodos.
2. Elaborar el grafo de flujo.
3. Calcular la complejidad ciclomática del grafo.
4. Determinar un conjunto básico de caminos independientes.
5. Preparar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

➢ **Pruebas de Aceptación (validación)**

✓ **De caja negra**

o **Partición equivalente**

Procedimiento:

1. Se identifican clases de equivalencia válida (CEV) e inválida (CEI).

2. Asignar un número único a cada clase de equivalencia.

3. Escribir casos de prueba hasta que sean cubiertas todas las CEV, intentando cubrir en cada caso tantas CEV como sea posible.

4. Para cada CEI, escribir un caso de prueba, cubriendo en cada caso una CEI.

- **Pruebas de integración**

✓ **Incremental ascendente**

Procedimiento:

1. Se combinan los módulos de bajo nivel en grupos que realicen una subfunción específica.

2. Se escribe un controlador (un programa de control de la prueba) para coordinar la entrada y 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.

- **Pruebas de sistema**

✓ **Prueba de stress**

Procedimiento:

1. Se determina el número de usuarios que se desea intervengan a la vez en un/os camino/s crítico/s.

2. Se van simulando incorporaciones de usuarios que ejecutan el mismo camino, todos al mismo tiempo.

3. Incorporar más usuarios que los necesarios hasta conseguir que el sistema colapse.

Conclusiones parciales.

Una vez estudiados los diferentes tipos de prueba así como sus técnicas y procedimientos se pudo definir la estructura de los casos de prueba para cada uno de los tipos de prueba definido lo que permite que se tenga una organización de cómo aplicar cada una de ellas. Además, quedó definida la importancia de tener un plan de prueba correctamente estructurado y que abarque tanto pruebas de alto nivel como de bajo nivel.

3

Capítulo

Diseño, ejecución y evaluación de las pruebas

Introducción

Este capítulo contiene todo lo referente al diseño de los casos de prueba ya elaborados para cada caso de uso y para cada tipo de prueba a utilizar según los niveles antes definidos, además de la planificación para la aplicación de las mismas. Se evalúan los resultados obtenidos de la ejecución de los casos de pruebas y se realiza un resumen de los mismos.

3.1. Diseño de los casos de pruebas.

- **Pruebas de Unidad**
- ✓ **De Caja Blanca**
  - **Camino básico**

Nombre CU: Gestionar Usuario

Para el Camino 1: 1-2-3-4-2-5

Caso de Prueba: Probando la función UpdRol

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado:

Comprobar que la instrucción foreach se cumple y se puedan actualizar los datos del rol.

Para el Camino 2: 1-2-5

Caso de Prueba: Probando la función UpdRol

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado:

Comprobar que la instrucción foreach no se cumple y no ocurre ningún cambio en los datos del rol.

Nombre CU: Adicionar Usuario

Para el Camino 1: 1-2-3-4-2-5

Caso de Prueba: Probando la función AddRol

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado:

Comprobar que la instrucción foreach se cumple y se puede añadir un nuevo rol a la Base de Datos.

Para el Camino 2: 1-2-5

Caso de Prueba: Probando la función AddRol

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado:

Comprobar que la instrucción foreach no se cumple y no permite añadir un nuevo rol a la Base de Datos.

Nombre CU: Gestionar Rol

Para el Camino 1: 1-2-3-4-5

Caso de Prueba: Probando la función ProcessForm

Entrada:

Valor de sfWebRequest $request: por definir.

Valor de sfForm $form: por definir.

Resultado:

Comprobar que la instrucción if se cumple y se compruebe que el formulario es válido.

Para el Camino 2: 1-2-5

Caso de Prueba: Probando la función ProcessForm

Entrada:

Valor de sfWebRequest $request: por definir.

Valor de sfForm $form: por definir.

Resultado:

Comprobar que la instrucción if no se cumple y se compruebe que el formulario no es válido.

Nombre CU: Gestionar Rol

Para el Camino 1: 1-2-3-4-5-6-7-9-11-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir.

Resultado:

Comprobar que la instrucción if se cumple y si existe el rol con esos permisos no se crea.

Para el Camino 2: 1-2-3-4-5-6-8-9-11-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir.

Resultado:

Comprobar que la instrucción if se cumple y si no existe el rol con esos permisos se crea.

Para el Camino 3: 1-2-3-4-10-11-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir.

Resultado:

Comprobar que la instrucción if no se cumple y no se pueda crear un rol con ese nombre.

Para el Camino 4: 1-2-12-13-14-16-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir**.**

Resultado:

Comprobar que la instrucción if se cumple y se pueda modificar el rol.

Para el Camino 5: 1-2-12-13-15-16-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir.

Resultado:

Comprobar que la instrucción if no se cumple y no se pueda modificar el rol porque no existe.

Para el Camino 6: 1-2-17-18-19-21-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir.

Resultado:

Comprobar que la instrucción if se cumple y se pueda eliminar el rol.

Para el Camino 7: 1-2-17-18-20-21-22

Caso de Prueba: Probando la función ExecuteRolGestioner

Entrada:

Valor de sfWebRequest $request: por definir.

Resultado:

Comprobar que la instrucción if no se cumpla y no se pueda eliminar el rol porque no existe.

- **Pruebas de Aceptación (validación)**
- ✓ **De Caja Negra**
  - **Partición equivalente**

Nombre del Caso de Uso: Gestionar Rol

Descripción General: El CU comienza cuando el Administrador desea adicionar un rol y le asigna permisos, lo modifica, lo elimina o lo busca. El sistema brinda las opciones pertinentes. El Administrador selecciona la opción deseada, realiza la operación y termina el CU con la actualización de los datos de dicho rol.

Condiciones de Ejecución: El usuario tiene que estar autenticado con privilegios de administración para realizar cualquiera de las acciones antes mencionadas.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC1:Gestionar Rol	EC 1.1: El administrador solicita visualizar interfaz para adicionar un rol.	El sistema muestra la interfaz con los campos a llenar.
SC 2:Insertar Rol	EC 2.1: El administrador entra los datos del rol especificando los permisos a los que tendrá acceso.	El sistema comprueba la validez de la información. Busca el rol en la Base de Datos para verificar que no exista.
	EC 2.2: El usuario no introduce datos en alguno de los campos o en caso de hacerlo incorrectamente.	El sistema marca en rojo los campos que no han sido introducidos o que presentan datos incorrectos.

SC 3: Modificar Rol	EC 3.1: El administrador selecciona el rol que desea modificar.	El sistema verifica la existencia del rol especificado. Muestra información del rol.
	EC 3.2: El administrador modifica los parámetros deseados y solicita modificar el rol.	Verifica que esté correcta la información. Se actualiza el rol en la Base de Datos.
	EC 3.3: El usuario modifica datos en los campos pero incorrectamente.	El sistema emite un mensaje de error en la información que se desea modificar.
SC 4: Eliminar Rol	EC 4.1: El administrador selecciona la opción eliminar.	El sistema muestra la información del rol.
	EC 4.2: El administrador selecciona el rol a eliminar.	El sistema pide la confirmación de la acción. Se elimina el rol, se actualiza la Base de Datos.
SC 5: Buscar Rol	EC 5.1: El administrador selecciona la opción buscar e inserta el criterio de búsqueda.	El sistema muestra los datos del rol buscado.
	EC 5.2: El administrador introduce el texto por el cual desea encontrar el rol incorrectamente o el rol no existe.	El sistema muestra un mensaje de error que indica que no se encontró el rol.

Tabla 2: Caso de prueba Gestionar Rol

No	Nombre de campo	Clasificación	Valor Nulo	Descripción

1	Gestionar Rol	Link	Si	El campo es un link a la interfaz donde se mostrarán todas las opciones a realizar
2	Insertar Rol	Es un campo Seleccionar, de tipo combo box	Si	El campo utilizado, se marca si se desea adicionar un Rol al sistema.
3	Rol	Es un campo de Texto.	Si	En el campo se escribe el Rol que se desea adicionar.
4	Permisos	Son campos de tipo Checkbox	Si	Este campo va a contener, todo los permisos a los que tendrá acceso el rol escrito.
5	Aceptar	Button	No	Este campo es el que me permitirá insertar el rol.
	Campo vacío	Texto	Si	Si no se introduce ningún rol, muestra en el campo. campo Vacío.
9	Cancelar	Button	No	El campo va a cancelar la operación de insertar Rol.
10	Modificar Rol	Es un campo Seleccionar, de tipo combo box	Si	El campo utilizado, se marca si se desea modificar un Rol al sistema.
11	Rol	Es un campo despegable.	No	En el campo se muestra todos los roles insertados en la base datos, y que se quieren modificar.
12	Permisos	Son campos de tipo Checkbox	Si	Se encuentran marcados los permisos que se le habían dado al rol, que se desea modificar anteriormente.
13	Aceptar	Button	No	El campo permitirá aceptar la opción de modificar el rol, escogido.

15	Eliminar Rol	Es un campo Seleccionar, de tipo combo box	Si	Este campo se marca si se desea eliminar un Rol
16	Rol	Es un campo despegable.	No	En este campo, se muestran todo los roles, que existen, para marcar el que se desea eliminar
17	Aceptar	Button	No	Este campo es para aceptar la opción de eliminar rol.
19	Cancelar	Button	No	En este campo se cancela la opción de eliminar.
20	Buscar Rol	Campo de Texto	Si	En este campo se escribe el criterio de búsqueda.
21	Icono	Button	No	Este campo me va a realizar la búsqueda.
22	Ventana	HTML	Si	Se muestra todo los criterios de búsqueda

Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante

Tabla 3: Descripción de variables

Escenario	Var 1 *(Insertar Rol)*	Var 2 *(Permisos)*	Var 3 *(Rol)*	Var 4 (Aceptar)	Var 5 (Cancelar)	Respuesta del Sistema	Resultado de la Prueba	Flujo Central

Insertar correctamente	V	V	V	N/A	N/A	Que se marque la opción de insertar Rol, Que se pueda escribir en el campo rol, un rol determinado, que se marque los permisos para ese rol, y que se pueda insertar mediante el campo aceptar o cancelar la operación, por el campo cancelar. Que en caso de que los datos sean correctos se inserten, satisfactoriamente en la base de datos, en caso de haber un rol con los mismos permisos, este debe informar al usuario, mediante un mensaje, y en caso de haber ya insertado ese rol anteriormente, comunicarlo mediante un mensaje.		Se marca en el checkbox la opción insertar rol. Se escribe un rol determinado en el campo rol. Se dan los permisos al Rol. Se acepta la opción insertar. Se verifica mensaje de inserción exitosa Se escribe un rol insertado anteriormente, Se acepta la opción insertar Se verifica mensaje de rol insertado anteriormen

Campos Vacios						En caso de que no se escriba nada en el campo rol, muestra el campo en rojo, y con un mensaje de campo vacio			te Se escribe rol determinado Darle permisos de otro rol ya insertado anteriormente Aceptar la opción insertar

Tabla 4: Matriz de datos SC 2 Insertar Rol

Escenario	Var 1 (Modificar Rol)	Var 2 (Permisos)	Var3 (Rol)	Var 4 (Aceptar)	Var 5 (Cancelar)	Var 6 (Modificación exitosa)	Var 7 No se Puede modificar	Respuesta del Sistema	Resultado de la Prueba	Flujo Central

Seleccionar rol	V		V					Que se marque la opción de Modificar Rol, Que muestre los roles existente,		Se marca en el checkbox la opción modificar rol. Se selecciona el rol que deseo modificar Se marcan los permisos que voy a modificar en el rol. Se acepta la opción modificar. Se verifica mensaje de modificación exitosa Se marcan los mismos permisos de otro Rol ya insertado. Se verifica mensaje de modificación incorrecta,

Modificación exitosa		V		V	V	I		Que muestre los permisos, y que se pueda modificar mediante el campo aceptar o cancelar la operación, por el campo cancelar, En caso de que los datos sean correctos, modificar el rol, y mostrar un mensaje informándo, del éxito de la		
Problemas en la modificación							I	Muestra un mensaje con el problema de por qué no se pudo modificar		

Tabla 5: Matriz de datos SC 3 Modificar Rol

Escenario	Variable 1 (Eliminar Rol)	Variable 3 (Rol)	Variable 4 (Aceptar)	Variable 5 (Cancelar)	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
Seleccionar Opción Eliminar	V				Que se marque la opción de Eliminar Rol,		Se marca en el checkbox la opción eliminar rol. Se selecciona el rol que deseo eliminar Se acepta eliminar o cancelar.
Seleccionar Rol y eliminar		v	v	v	Que muestre los roles existente, y permita eliminar mediante el botón aceptar o cancelar mediante el botón cancelar		,

Tabla 6: Matriz de datos SC 4 Eliminar Rol

Escenario	Variable 1 (Buscar Rol)	Variable 2 (icono)	Variable 3 ventana	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
Buscar Rol	V	V	v	El sistema muestra el campo de texto donde se va a especificar el criterio de búsqueda, y el icono, donde se va a dar clic para realizar la búsqueda, luego sale una ventana mostrando los criterios de búsqueda encontrados.		Se escribe un criterio de búsqueda Se da clic en el icono Se verifica que la búsqueda realizada sea correcta. Se pone otro criterio de búsqueda

Datos Incorrectos o criterio no encontrado				El sistema muestra en la ventana un mensaje de error, informando sobre el criterio de búsqueda o es incorrecto, o no se encuentra nada sobre él.		incorrecto Se verifica mensaje de información, sobre este error.

Tabla 7: Matriz de datos SC 5 Buscar Rol

Nombre del Caso de Uso: Autenticar usuario

Descripción General: El usuario del sistema se autentica para obtener permisos sobre determinadas opciones del sistema.

Condiciones de Ejecución: Para realizar esta operación debe haberse registrado anteriormente.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
Autenticar Usuario	EC 1.1: “Autenticación exitosa”.	El usuario del sistema se autentica para obtener permisos sobre determinadas opciones del sistema.
	EC1.2: “Faltan datos obligatorios”.	El usuario debe completar todos los datos obligatorios.

	EC 1.3: “Usuario o contraseña no existen en la BD”.	El usuario debe poner bien su usuario y contraseña si no está registrado debe solicitar que lo hagan.

Tabla 8: Caso de prueba Autenticar Usuario

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
Usuario	Usuario	Campo de texto	No	El usuario debe contener solo letras.
Contraseña	Contraseña	Campo de texto	No	La contraseña puede tener tanto números como letras y otros símbolos.

Tabla 9: Descripción de variables

Escenario	*Usuario*	*Contraseña*	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1: “Autenticación exitosa”.	*desampedro*	*(vacía)*	El usuario queda autenticado y obtiene permiso para entrar al sistema		Se entra el usuario Se entra la contraseña

	ereyesg	*(vacía)*	El sistema muestra un mensaje de error alertando que el usuario no es		Se pulsa el botón.
	svazquez	*admin*	El sistema muestra un mensaje de error alertando que la contraseña no es		
EC1.2: "Faltan datos obligatorios".	ereyesg	(vacía)	El sistema muestra un mensaje de error alertando que debe llenar todos		Se entra el usuario Se entra la contraseña Se pulsa el botón.
	dsampedro	admin	El sistema muestra un mensaje de error alertando que debe llenar todos		
EC 1.n: 3 "Usuario o contraseña no existen en la BD".	svazquez	(vacía)	El sistema da acceso al usuario. Después de buscarlo en la base de datos y haberlo		Se entra el usuario Se entra la contraseña

	ereyesg	(vacía)	El sistema muestra un mensaje de error alertando que ese usuario no		Se pulsa el botón.
	dsampedro	admin	El sistema muestra un mensaje de error alertando que esa contraseña		

Tabla 10: Matriz de datos SC 1 Autenticar Usuario

Nombre del Caso de Uso: Gestionar usuario

Descripción General: El CU comienza cuando el Administrador desea adicionar un usuario y le asigna permisos, lo modifica, lo elimina o lo busca. El sistema brinda las opciones pertinentes. El Administrador selecciona la opción deseada, realiza la operación y termina el CU con la actualización de los datos de dicho usuario.

Condiciones de Ejecución: El usuario tiene que estar autenticado con privilegios de administración para realizar cualquiera de las acciones antes mencionadas.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC1:Gestionar Usuario	EC 1.1: El administrador solicita visualizar interfaz para adicionar un usuario	El sistema muestra la interfaz con los campos a llenar.
SC 2:Insertar Usuario	EC 2.1: El administrador entra los datos del usuario especificando los permisos a los que tendrá acceso.	El sistema comprueba la validez de la información. Busca el usuario en la Base de Datos para verificar que no exista.

	EC 2.2: El usuario no introduce datos en alguno de los campos o en caso de hacerlo incorrectamente.	El sistema marca en rojo los campos que no han sido introducidos o que presentan datos incorrectos.
SC 3:Modificar Usuario	EC 3.1: El administrador selecciona el usuario que desea modificar.	El sistema verifica la existencia del usuario especificado. Muestra información del usuario.
	EC 3.2: El administrador modifica los parámetros deseados y solicita modificar el usuario.	Verifica que esté correcta la información. Se actualiza el usuario en la Base de Datos.
	EC 3.3: El usuario modifica datos en los campos pero incorrectamente.	El sistema emite un mensaje de error en la información que se desea modificar.
SC 4: Eliminar Usuario	EC 4.1: El administrador selecciona la opción eliminar.	El sistema muestra la información del usuario.
	EC 4.2: El administrador selecciona el usuario a eliminar.	El sistema pide la confirmación de la acción. Se elimina el usuario, se actualiza la Base de Datos.
SC 5: Buscar Usuario	EC 5.1: El administrador selecciona la opción buscar e inserta el criterio de búsqueda.	El sistema muestra los datos del usuario buscado.
	EC 5.2: El administrador introduce el texto por el cual desea encontrar el usuario incorrectamente o el usuario no existe.	El sistema muestra un mensaje de error que indica que no se encontró el usuario.

Tabla 11: Caso de prueba Gestionar Usuario

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
Usuario	Usuario	Campo de texto	No	El usuario debe contener solo letras.
Contraseña	Contraseña	Campo de texto	No	La contraseña puede tener tanto letras como números y otros símbolos.

Tabla 12: Descripción de variables

Escenario	***Usuario***	***Contraseña***	**Respuesta del Sistema**	**Resultado de la Prueba**	**Flujo Central**
Adicionar Usuario	*V*	*V*	El usuario queda almacenado en la Base de Datos y obtiene permiso para entrar al sistema		Se introduce el usuario Se pulsa el botón.
	I	*N/A*	El sistema muestra un mensaje de error alertando que el usuario no es correcto.		

	I	*N/A*	El sistema muestra un mensaje de error alertando que ya existe en la Base de Datos		
Modificar Usuario	I	*N/A*	El sistema muestra un mensaje de error alertando que el usuario no existe en la Base de Datos		Se introduce el usuario Se pulsa el botón.
Eliminar Usuario	V	V	El usuario es eliminado de la Base de Datos		Se introduce el usuario Se pulsa el botón.
	I	*N/A*	El sistema muestra un mensaje de error alertando que ese usuario no esta en la Base de Datos.		

Tabla 13: Matriz de datos SC 1 Gestionar Usuario

- **Pruebas de Integración**

✓ **Integración Incremental Ascendente**

A medida que se van terminando los módulos se integran hasta tener el sistema construido totalmente y ver que funciona como un todo. Se prueban las funcionalidades de los demás subsistemas para asegurar que los datos que manipula el subsistema de seguridad correspondan con los necesarios para el correcto funcionamiento del sistema de manera general, se comprueba que al darle los permisos a cada rol en específico, el usuario al cual sea asignado este rol tenga acceso a los módulos establecidos en los mismos, además de que cuando se cree un usuario este sea adicionado a la base de datos. Teniendo

todos los diseños de prueba listos se debe planificar el período en que se van a aplicar cada uno para luego documentar los resultados.

- **Pruebas de Sistema**

✓ **Pruebas de Stress**

Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. Dentro de las pruebas de sistema la más útil en este caso es la de Stress ya que está enfocada a evaluar cómo el sistema responde bajo las condiciones anormales, dígase extrema sobrecarga, incipiente memoria entre otras. En este caso se va a probar cuantas conexiones es capaz de soportar la aplicación y se va a realizar con la herramienta JMeter la cual se utiliza para simular un número de conexiones a la misma, ya que utilizando una herramienta para la automatización de este proceso se logra reducir los costos en tiempo y recursos destinados a su realización, además que utilizar JMeter en aplicaciones web, presume una mayor confianza en el proceso y en la fiabilidad de los resultados. **(Almenares, 2008)**

Durante la aplicación de las pruebas utilizando la herramienta se obtienen informaciones como las siguientes:

✓ Muestras: Cantidad de páginas (Hilos) que simulan la cantidad de usuarios que están interactuando con el sistema desde la misma URL.

✓ Media: Media de páginas que se cargaron de manera satisfactoria.

✓ Mediana: Tiempo promedio que han tardado en cargarse las páginas.

✓ Min: Tiempo mínimo que ha demorado en cargarse una página.

✓ Max: Tiempo máximo que ha tardado en cargarse una página.

✓ Línea 90 %: 90 por ciento de las páginas que se cargaron de manera satisfactoria.

✓ %Error: Por ciento de error de las páginas que no se llegaron a cargar de manera satisfactoria.

✓ Kb/Seg: Velocidad de carga de las paginas.

✓ Tiempos de Respuestas: Total del tiempo que demoró en cargarse la cantidad de hilos de esa prueba. **(Almenares, 2008)**

Luego de tener los parámetros necesarios para trabajar con el sistema, es necesaria la utilización de una Aserción de Respuesta (Anexo 1) en la se especificará el patrón que se necesita validar en la prueba. **(Almenares, 2008)**

Por último solo se necesitan algunos de los elementos en los cuales se registran los resultados de las pruebas, los cuales son los Listeners, cada uno especializado en mostrar los resultados de las pruebas por aspectos y características diferentes según las necesidades del test. En este caso los utilizados son el Informe agregado (Anexo 2) y Ver Árbol de Resultados (Anexo 3).

Cantidad de conexiones	Resultado esperado	Resultado obtenido

Tabla 14: Modelo para recolectar los resultados de las pruebas de Stress

Cuando ya el sistema esté terminado se conectan al mismo múltiples usuarios al mismo tiempo para probar si resiste o no.

3.2. Cronograma de aplicación de las pruebas

Nro.	Período de aplicación (abril)	Prueba
1	**1ro - 6**	**Caja Blanca(Camino Básico)**
2	**7 - 13**	**Caja Negra(Partición Equivalente)**
3	**14**	**Integración (Incremental Ascendente)**
4	**15**	**Sistema (Stress)**

Tabla 15: Cronograma de aplicación de las pruebas

Luego de haber pasado esta etapa de pruebas, se observan los resultados esperados en el momento en que se diseñó cada caso de prueba para cada uno de los casos de uso.

3.3. Resultados obtenidos

- **Pruebas de Unidad**
- ✓ **De Caja blanca**
  - **Camino básico**

Nro. del camino	Caso de Prueba	Objetivo	Resultado
1	Probando la función UpdRol	Comprobar que la instrucción foreach se cumple y se puedan actualizar los datos del rol.	Satisfactorio
2	Probando la función UpdRol	Comprobar que la instrucción foreach no se cumple y no ocurre ningún cambio en los datos del rol.	Satisfactorio
1	Probando la función AddRol	Comprobar que la instrucción foreach se cumple y se puede añadir un nuevo rol a la Base de Datos.	Satisfactorio
2	Probando la función AddRol	Comprobar que la instrucción foreach no se cumple y no permite añadir un nuevo rol a la Base de Datos.	Satisfactorio
1	Probando la función ProcessForm	Comprobar que la instrucción if se cumple y se compruebe que el formulario es válido.	Satisfactorio

2	Probando la función ProcessForm	Comprobar que la instrucción if no se cumple y se compruebe que el formulario no es válido.	Satisfactorio
1	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if se cumple y si existe el rol con esos permisos no se crea.	Satisfactorio
2	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if se cumple y si no existe el rol con esos permisos se crea.	Satisfactorio
3	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if no se cumple y no se pueda crear un rol con ese nombre.	Satisfactorio
4	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if se cumple y se pueda modificar el rol.	Satisfactorio
5	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if no se cumple y no se pueda modificar el rol porque no existe.	Satisfactorio
6	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if se cumple y se pueda eliminar el rol.	Satisfactorio
7	Probando la función ExecuteRolGestioner	Comprobar que la instrucción if no se cumpla y no se pueda	Satisfactorio

	eliminar el rol porque no existe.

Tabla 16: Resultados de las pruebas de Caja Blanca

Analizados estos resultados se llega a la conclusión de que el sistema no presenta ningún problema en las funciones implementadas.

- **Pruebas de Aceptación (validación)**
- ✓ **De caja negra**
  - **Partición equivalente**

Secciones	Resultado esperado	Resultado obtenido
Insertar rol	Que se marque la opción de insertar Rol, Que se pueda escribir en el campo rol, un rol determinado, que se marque los permisos para ese rol, y que se pueda insertar.	Se muestra el formulario correctamente. El campo rol no acepta entrada de números como dato válido. Las cacillas de verificación permiten seleccionar los permisos y estos son los módulos a los que tendrá acceso ese rol.
Modificar rol	Que se muestren los permisos, y que se pueda modificar mediante el campo aceptar o cancelar la operación, por el campo cancelar, en caso de que los datos sean correctos, modificar el rol, y mostrar un mensaje informando, del éxito de la operación.	Se muestran los permisos y permite modificarlos, al hacerlo se modifican los módulos a los cuales tiene acceso el rol.

Eliminar rol	Que se muestren los roles y permita seleccionar el que se desea eliminar.	Se elimina el rol de la base de datos.
Buscar rol	Que se muestre el campo de texto donde se va a especificar el criterio de búsqueda, y el icono, donde se va a dar clic para realizar la búsqueda, luego sale una ventana mostrando los criterios de búsqueda encontrados.	Se encuentra el rol buscado según los criterios de búsquedas introducidos.
Autenticar usuario	El sistema debe dar la opción de introducir usuario y contraseña para autenticarse.	Se muestran las opciones para introducir usuario y contraseña y acceder al sistema.
Insertar usuario	El sistema debe dar la opción de introducir los datos del usuario a insertar.	El sistema brinda la opción de introducir los datos y luego almacena el usuario a la base de datos.
Modificar usuario	El sistema debe dar la opción de introducir los datos del usuario a insertar.	El sistema brinda la opción de introducir los datos y luego modifica el usuario en la base de datos.
Eliminar usuario	El sistema debe dar la opción de introducir los datos del usuario a insertar.	El sistema brinda la opción de introducir los datos y luego elimina el usuario de la base de datos.

Tabla 17: Resultados de las pruebas de Caja Negra

- **Pruebas de integración**
- ✓ **Integración incremental ascendente**

Luego de realizar las pruebas de caja negra se integraron los módulos y se pudo comprobar que están debidamente integrados y que funcionan correctamente. Cuando se crea un usuario y se le otorgan los permisos según el rol que se le asigne, el usuario accede a los módulos a los cuales puede acceder según las restricciones de permisos asignadas. También el sistema permite adicionar o eliminar usuarios correctamente, así como modificarlo, y en caso de que este en el momento de adicionarlo ya existe en la base de datos aparece un mensaje informándolo, y en el caso de eliminarlo, si no existe también aparece un mensaje dándolo a conocer. Además, los usuarios tienen que ser del dominio UCI sino no son admitidos por el sistema, todas estas y otras funciones están validadas y funcionan correctamente.

- **Pruebas de sistema**
- ✓ **Pruebas de Stress**

Durante la realización de las pruebas de stress al sistema SGPDTU se pudo recopilar cierta información referente al funcionamiento del sistema bajo condiciones extremas algunos de estos datos se muestran a continuación:

Label	Muestras	Media	Mediana	Línea 90 %	Min	Max	% Error	Rendimiento	Kb/Seg
Pág. Inicio	50	40042	34500	63094	25188	72156	0.00	37.2/min	3.3
UserGestioner	200	20434	23766	27125	688	28296	0.00	1.1/sec	7.0
Headerlog.png	50	313	157	1063	16	1484	0.00	2.0/sec	82.8
Total	14050	2035	31	2016	0	195391	0.00	18.7/sec	117.6

Tabla 18: Muestra de datos obtenidos en la realización de las pruebas de stress.

Luego de observar los resultados obtenidos se llega a la conclusión de que el sistema se demora más de lo establecido para cargar con esta cantidad de usuarios conectados simultáneamente, por lo que se debe tener en cuenta que el servidor donde se va a montar la aplicación debe tener buenas condiciones

técnicas, por ejemplo 2 GB de memoria RAM como mínimo, entre otras para evitar estos conflictos. Para más información diríjase a la tabla que a continuación se muestra:

Cantidad de conexiones	Resultado esperado	Resultado obtenido
50	Que el sistema responda satisfactoriamente para todos los usuarios y a la velocidad requerida.	El sistema se demora más del tiempo establecido para cargar por lo que se debe seguir trabajando en esto para mejorar las funcionalidades y todos estos usuarios puedan acceder al mismo sin ningún problema.

Tabla 19: Resultados de las pruebas de stress.

Conclusiones parciales

En este capítulo se han podido observar todos los casos de pruebas que fueron aplicados así como la fecha de la aplicación. También se exponen en el mismo la descripción de todas las variables que se utilizan en el módulo de seguridad de la aplicación que se está probando, se pudo analizar los resultados de las pruebas realizadas. Además, durante la realización de este capítulo surgen algunos consejos importantes a tener en cuenta a la hora de diseñar buenos casos de prueba y ejecutarlos:

- El implementador debe evitar probar sus propios programas ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas, ya que es normal que los errores que cometió programando no los identifique al ejecutar los casos de prueba.

CONCLUSIONES

El aseguramiento de la calidad del software permite reducir notablemente los costos de producción e implantación, y proporciona mayor confianza en el cumplimiento de los requisitos del cliente, siendo las pruebas de software un elemento imprescindible para asegurar la calidad y para verificar el cumplimiento de los requisitos funcionales y los impuestos por el cliente.

Durante la realización de este trabajo de diploma se ejecutaron y evaluaron 18 casos de prueba diseñados con diferentes tipos de pruebas para obtener la mayor cantidad de errores posibles existentes, donde se pudo observar que el módulo de seguridad del Sistema de Gestión de Procesos de la Dirección de Televisión Universitaria está listo.

Se estableció una metodología a aplicar en las pruebas de Caja Negra y en las de Caja Blanca para la ejecución de los casos de prueba, así como en las de integración y sistema. La aplicación de las pruebas a los casos de uso del módulo de Seguridad permitió verificar que no existen errores en los casos de uso ni en la lógica del programa de la aplicación SGP-DTU.

REFERENCIAS BIBLIOGRÁFICAS

autores, Varios. Breves notas sobre la Medición de los Atributos Externos del Software. Breves notas sobre la Medición de los Atributos Externos del Software. [En línea] [Citado el: 20 de enero de 2010.] http://www.sc.ehu.es/jiwdocoj/mmis/externas.htm. 4.

Granada., Profesores del dpto de lenguajes y sistemas informaticos de la universidad de. Departamento de lenguajes y sistemas informaticos de la universidad de Granada. Departamento de lenguajes y sistemas informaticos de la universidad de Granada. [En línea] [Citado el: 15 de noviembre de 2009.] http://aporia.ugr.es/lsi/.

Informáticas, Profesores de Ingenieria de Software de la Universidad de las Ciencias. Conferencias de de Ingenieria de Software. [Digital]

Mañas, José A. 1994. Laboratorio de programación. Laboratorio de programación. [En línea] 16 de marzo de 1994. [Citado el: 20 de noviembre de 2009.] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm#s2. 9.

Mijares, Damarys Lugo. Pruebas en Programación Orientada a Objetos. Pruebas en Programación Orientada a Objetos. [En línea] [Citado el: 17 de noviembre de 2009.] http://www.ldc.usb.ve/~teruel/ci3711/test3/#integ. 11.

Ricardo, Jose. 2010. La importancia de asegurar la calidad del software. La importancia de asegurar la calidad del software. [En línea] 2010. [Citado el: 22 de enero de 2010.] http://rjmenco.spaces.live.com/blog/cns!A07325753C2053C!159.entry.

Vega, Herman. 2005. GNOME Bluetooth Control Remoto Proyect. GNOME Bluetooth Control Remoto Proyect. [En línea] 7 de julio de 2005. [Citado el: 25 de octubre de 2009.] http://gbtcr.chileforge.cl/.

aBooch, G., Rumbaugh, J., Jacobson,. 2004. *El Proceso Unificado de Desarrollo de Software.* La Habana Cuba : s.n., 2004.

G2009. Grupo soluciones GSInnova. *Grupo soluciones GSInnova.* [En línea] 2009. [Citado el: 10 de 04 de 2010.] http://www.rational.com.ar/herramientas/rup.html.

2010. HighBeam. *HighBeam.* [En línea] 2010. [Citado el: 06 de 03 de 2010.] http://business.highbeam.com/3058/article-1G1-143826412/dise-o-de-un-estandar-de-verificaci-n-y-validaci-n.

libre, Asociación Peruana de Software. 2003-2009. EQSOFT Consultoría y Soporte EIRL (Lima-Perú). *EQSOFT Consultoría y Soporte EIRL (Lima-Perú).* [En línea] Desarrollo de Software, Redes y Comunicaciones, Seguridad de la

Información, Auditoría Informática, Capacitación, Consultorías., 2003-2009. [Citado el: 20 de marzo de 2010.] http://www.eqsoft.net/presentas/modelos_de_calidad_y_software_libre.pdf.

Luis Fernández Sanz, Miren Idoia Alarcón Rodríguez. Necesidades de medición en la gestión y el aseguramiento de calidad del software. *Necesidades de medición en la gestión y el aseguramiento de calidad del software.* [En línea] [Citado el: 20 de marzo de 2010.] http://www.sc.ehu.es/jiwdocoj/remis/docs/aseguracal.htm.

McCall.J. 1997. *Factors in software Quality.* s.l. : General Electric, 1997.

Cómo realizar Pruebas de Carga y Estrés en JMeter [Libro] / aut. Almenares Liudmila Sánchez. - Ciudad de la Habana : [s.n.], 2008.

Scribd. (s.f.). Recuperado el 15 de junio de 2010, de Scribd: http://www.scribd.com/doc/12983228/Fases-en-RUP

BIBLIOGRAFÍA

Acuña, C. J. Pruebas de Software (Caja Negra). Universidad Rey Juan Carlos.

Aguirre, H. d. (27 de agosto de 2008). Mainsoft, tecnología para sus negocios. Recuperado el 10 de diciembre de 2009, de Mainsoft, tecnología para sus negocios: http://www.mainsoft.cl/mainsoft/productosyservicios/gestionaplicaciones/aseguramientocalidad/qa.html

B, I. A. (2009). Calidady software.com. Recuperado el 9 de febrero de 2010, de Calidady software.com: http://www.calidadysoftware.com/testing/pruebas_unitarias3.php

El prisma. (s.f.). Recuperado el 10 de noviembre de 2009, de El prisma: http://www.elprisma.com/apuntes/administracion_de_empresas/gestiondelacalidad/

Escobar, L. M. (2003). La calidad del software y su importancia en el mercado.

Ing. Violena Hernández Aguilar, I. M. Proceso de pruebas de caja negra basado en la descripción de casos de uso.

Krug, S. (2000). “Don’t Make Me Think. A Common Sense Approach to Web Usability.”. Circle.com Library.

libre, Asociación Peruana de Software. 2003-2009. EQSOFT Consultoría y Soporte EIRL (Lima-Perú). EQSOFT Consultoría y Soporte EIRL (Lima-Perú). [En línea] Desarrollo de Software, Redes y Comunicaciones, Seguridad de la Información, Auditoría Informática, Capacitación, Consultorías., 2003-2009. [Citado el: 20 de marzo de 2010.] http://www.eqsoft.net/presentas/modelos_de_calidad_y_software_libre.pdf.

Luis Fernández Sanz, Miren Idoia Alarcón Rodríguez. Necesidades de medición en la gestión y el aseguramiento de calidad del software. Necesidades de medición en la gestión y el aseguramiento de calidad del software. [En línea] [Citado el: 20 de marzo de 2010.] http://www.sc.ehu.es/jiwdocoj/remis/docs/aseguracal.htm.

Mañas, J. A. (16 de marzo de 1994). Laboratorio de programación. Recuperado el 20 de noviembre de 2009, de Laboratorio de programación: http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm#s2

Menco Orozco, R. (s.f.). Recuperado el 13 de octubre de 2009, de http://rjmenco.spaces.live.com/blog/cns!A07325753C2053C!159.entry

Mijares, D. L. (s.f.). Pruebas en Programación Orientada a Objetos. Recuperado el 17 de noviembre de 2009, de Pruebas en Programación Orientada a Objetos: http://www.ldc.usb.ve/~teruel/ci3711/test3/#integ

Morales, L. (s.f.). Baquia Knowledge center. Recuperado el 20 de octubre de 2009, de Baquia Knowledge center: http://www.baquia.com/noticias.php?id=9778

Natalia Juristo, A. M. (2007). TÉCNICAS DE EVALUACIÓN DE SOFTWARE.

Nielsen, J. (2000). "Usabilidad. Diseño de sitios Web". Prentice Hall.

Nielsen, J. (1995-2010). www.useit.com. Recuperado el 27 de noviembre de 2009, de www.useit.com: http://www.useit.com/

Prado, E. R. (2007). Actas de Talleres de Ingeniería del Software y Bases de Datos, Vol. 1.

Pressman, R. S. (2005). Ingieneria de Software, un enfoque práctico.

autores, Varios. Breves notas sobre la Medición de los Atributos Externos del Software. *Breves notas sobre la Medición de los Atributos Externos del Software.* [En línea] [Citado el: 20 de enero de 2010.] http://www.sc.ehu.es/jiwdocoj/mmis/externas.htm. 4.

Booch, G., Rumbaugh, J., Jacobson,. 2004. *El Proceso Unificado de Desarrollo de Software.* La Habana Cuba : s.n., 2004.

Granada., Profesores del dpto de lenguajes y sistemas informaticos de la universidad de. Departamento de lenguajes y sistemas informaticos de la universidad de Granada. *Departamento de lenguajes y sistemas informaticos de la universidad de Granada.* [En línea] [Citado el: 15 de noviembre de 2009.] http://aporia.ugr.es/lsi/.

2009. Grupo soluciones GSInnova. *Grupo soluciones GSInnova.* [En línea] 2009. [Citado el: 10 de 04 de 2010.] http://www.rational.com.ar/herramientas/rup.html.

2010. HighBeam. *HighBeam.* [En línea] 2010. [Citado el: 06 de 03 de 2010.] http://business.highbeam.com/3058/article-1G1-143826412/dise-o-de-un-estandar-de-verificaci-n-y-validaci-n.

Informáticas, Profesores de Ingenieria de Software de la Universidad de las Ciencias. *Conferencias de de Ingenieria de Software.* [Digital]

McCall.J. 1997. *Factors in software Quality.* s.l. : General Electric, 1997.

Ricardo, Jose. 2010. La importancia de asegurar la calidad del software. *La importancia de asegurar la calidad del software.* [En línea] 2010. [Citado el: 22 de enero de 2010.] http://rjmenco.spaces.live.com/blog/cns!A07325753C2053C!159.entry.

Cómo realizar Pruebas de Carga y Estrés en JMeter [Libro] / aut. Almenares Liudmila Sánchez. - Ciudad de la Habana : [s.n.], 2008.

Scribd. (s.f.). Recuperado el 15 de junio de 2010, de Scribd: http://www.scribd.com/doc/12983228/Fases-en-RUP

GLOSARIO

Calidad: Medida de la cierta característica deseable.

ACS: Aseguramiento de la Calidad del Software

V&V: Verificación y validación.

SGPDTU: Sistema de gestión de procesos de la dirección de televisión universitaria.

CEV: Clases de equivalencia válidas.

CEI: Clases de equivalencia inválidas.

RUP: Proceso unificado de desarrollo

Var: Variable.