Universidad de las Ciencias Informáticas

Facultad 6

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

Título: **"Diseño y aplicación de pruebas al Sistema de Gestión de Audio, Video y Streaming para Venezolana de Televisión (VTV)".**

Autor(es): **Darmalina Valenciano Cuza**

Tutor(es): **Ing. Yesleny Becerra Torreira**

Ciudad de La Habana, 2011

Año 53 del Triunfo de la Revolución

Dedicatoria

A mis padres por ser, sin duda alguna, los mejores del mundo.
A mi Ñoñi, por ser tan oportuno en mi vida.
A mi familia por confiar en mí, los que están y los que no.
A abuelo Beto que donde quiera que esté, está muy orgulloso de mí.

Agradecimientos

Esta es la parte más complicada de la toda la tesis, porque es donde único te dejan hablar en primera persona, y donde te permites ser agradecido con todos aquellos que han sido de gran influencia, tanto en la realización del trabajo como en la vida. No quiero que se me quede nadie, pero la verdad es que si mencionara a todos los que quiero tendría que contar con unas cuantas páginas, una por los tantos años que llevo haciendo esta carrera y dos porque soy muy vieja, y hay a quienes le debo la vida.

Mimi, no hay palabras, no hay números, no hay nada que pueda describir todo lo que te debo, gracias por permitirme seguir siendo la niña de tus ojos.

Pipa, sé que darías lo que fuera por estar aquí conmigo, quiero que sepas que te extraño muchísimo y te doy las gracias por tenerme en el lugar más alto de tu vida.

A mi novio Ransel le estoy eternamente agradecida por ser tan paciente y entregarse a todo lo que tiene que ver conmigo.

A mi familia por no perder la fe y ser todos tan lindos. Mis tías y tíos por tanto cariño, mis primos que son como mis hermanos, mis sobrinos que son lo más hermoso que ojos humanos hayan visto, a abuela Darmalina por estar ahí cada día y mis hermanos que están siempre tan lejos. A quienes son como mi familia Hilda, Tere y abuelita por toda su preocupación.

A mi mejor amiga, o mejor hermana Dachy, por todos los momentos mágicos que me has regalado, porque con amigas como tú la vida solo es en colores.

A mis otros mejores amigos de la vida Tay, Lisi, Raquelita, Clenda, Ledys, Yane, Yusniel, Libecita y Adelito, aunque estén de cierta forma distantes están en mi corazón.

Si de amigos se trata son muchos nombres, sin aires de dejar a personas atrás, hay algunos que no puedo dejar de mencionar:

Sucerdo, Yudith, Yaimit, Ody, Dilenis y Susi gracias por hacer que estos últimos años fueran más ameno. A mis amigos del pre y la universidad en especial a Julio, Niurvis, Yailyn, el Cori, Daylin, Reinaldo, Juan Alberto, Jossy, Raiza, Olguita, Lucila e Ismaray.

A los amigos de mi Ñoñi que se han convertido en mis amigos: Luisi, Daniel, Annia, Manresa y Orelmis.

Todos mis compañeros de primer año, en especial a Alegna, Yadira, Karen, el Chiti, Yoa.

A todos mis compañeros del proyecto en especial Yane, Daril, Yori, Leyanis y Yasmani.

A mis compañeros de aula en especial a la Flaca, Yalili, Liu, Lili, Yiyi, Adnan, Shogun, Isidro y David.

En cuestión de profesorado, quiero dar las gracias muy en especial a Febe Ángel Ciudad, por ser tan ético, tan amigo, un excelente ser humano, gracias por todo lo que me enseñaste. A José Carlos Santiesteban por toda su ayuda con la programación. Alain eres el mejor guía que he tenido en la historia, gracias por soportar todas mis inmadureces. Yusde y Zori son un amor, gracias por estar cada vez que los necesité. Yoenis y Yuri por tanto aliento. A mi oponente Liuba por ser incondicional.

Gracias a todo el que contribuyó con la realización de este sueño y por qué no al que no lo hizo enseñándome a crecerme.

Declaración de autoría

Declaro que soy el único autor 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 _____.

Darmalina Valenciano Cuza	**Yesleny Becerra Torreira**
_____	_____
Autor	**Tutor**

Datos de contacto

Darmalina Valenciano Cuza.

Correo: dvalenciano@estudiantes.uci.cu

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Ing. Yesleny Becerra Torreira.

Ingeniera en Ciencias Informáticas.

Correo: ybecerra@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Opiniones y avales

Resumen

El presente trabajo de diploma titulado “Diseño y aplicación de pruebas al Sistema de Gestión de Audio, Video y Streaming para Venezolana de Televisión (VTV)” se realizó con el objetivo fundamental de garantizar la calidad de este producto, corrigiendo la mayor cantidad de errores no encontrados por los desarrolladores y propiciando así una buena aceptación por parte del cliente.

Se llevó a cabo un proceso de pruebas que permitió detectar en qué estado se encuentra el producto y documentarlo. Para ello fue preciso realizar una fundamentación teórica donde se abordan diferentes conceptos emitidos por varios autores relacionados con la calidad del software, las pruebas de software y el aseguramiento de la calidad en el proceso de desarrollo. Se examinó lo referente al flujo de pruebas que define el Proceso Unificado de Desarrollo (RUP) que es la metodología seleccionada por el equipo de desarrollo, para guiar el proceso de construcción del mismo. Se investigaron los tipos de pruebas y los niveles que se tuvieron en cuenta en la selección de las pruebas que se aplicaron al producto.

Se confeccionó un plan de pruebas para facilitar el entendimiento de los pasos a seguir para el diseño y aplicación de las pruebas ya seleccionadas según sus características y las del producto. Una vez aplicadas las pruebas de software, se lleva a cabo una comparación entre los resultados esperados y los obtenidos, documentando recomendaciones para las siguientes versiones del producto.

Palabras claves: Calidad, pruebas de software.

Tablas y figuras

Índice

Introducción

La calidad en la industria del software es, hoy en día, un engranaje competitivo, por la necesidad de mejores productos, construidos en el menor tiempo posible y a menor costo. Para evitar la falta de calidad en los mismos acarreando consigo la insatisfacción de los clientes, las empresas que se dedican al desarrollo de software intentan aumentar la calidad del software producido. El mercado valora cada día más la calidad en los productos, procurando que lleguen a mano de los clientes finales con una probabilidad mínima de errores.

En ocasiones los profesionales no saben obtener de manera eficaz la calidad del software debido al no cumplimiento del calendario del proceso de desarrollo, cuando la respuesta puede ser sencilla. Una de las herramientas clave para obtener software de alta calidad y con un minúsculo margen de error es contar con un proceso de pruebas efectivo.

El proceso de pruebas ayuda a asegurar que todas las partes de un sistema funcionen como se espera. Si el mismo cuenta con una previa planeación y se aplica desde los inicios de desarrollo del software se tornará más práctico. Al ejecutar las pruebas se reportan los resultados, resolviendo finalmente los problemas más relevantes, trayendo consigo la ventaja de poder garantizar la calidad del software, satisfacción de los requisitos y ahorrar tiempo y recursos en el desarrollo.

Cuba no está exenta de lo anteriormente expuesto, por lo que surge la idea de que la Universidad de las Ciencias Informáticas (UCI) además de su labor docente, logre informatizar la sociedad cubana y lo haga con la mayor calidad posible. La universidad cuenta con 7 facultades, y la facultad 6, es la encargada de brindar soluciones informáticas en el campo de Señales Digitales y Geoinformática, en el Centro de Desarrollo GEySED. La UCI mantiene contratos con diferentes proyectos productivos en varios países, destacándose la República Bolivariana de Venezuela, permitiéndose de esta forma introducirse al mercado nacional y latinoamericano de software.

En la actualidad la República Bolivariana de Venezuela cuenta con la televisora Venezolana de Televisión (VTV), donde toda la información audiovisual que posee se encuentra almacenada en soporte digital y formatos analógicos, haciéndose difícil su control y acceso. Para facilitar el control y la recuperación de información, ubicados en los archivos de VTV, especialmente su organización y preservación se requiere de la implementación de un software. Una vez digitalizados los materiales audiovisuales presentes en el archivo de VTV, se permitirá el acceso, la catalogación y la recuperación de los mismos. Siendo una alternativa para evitar la declinación de los materiales y la pérdida del patrimonio político, cultural y social de la República.

Introducción

El Departamento de Señales Digitales de la facultad 6 integrado por varios proyectos, donde se encuentra Captura y Catalogación de Medias, se dedica al desarrollo del producto Sistema de Gestión de Audio, Video y Streaming para VTV. El mismo surge por la necesidad de automatizar los procesos de Catalogación y Medias, así como realizar préstamos y búsquedas de manera automática.

El proyecto no tiene definido los procesos para el aseguramiento de la calidad de sus productos, esto trae consigo una escaza documentación de la planificación del proceso de pruebas, su aplicación y registro de resultados, trayendo como consecuencia retrasos en el cronograma de ejecución, aumento del costo de producción y en el tiempo de entrega del producto a los usuarios finales.

Con la aplicación de un buen proceso de pruebas al producto Sistema de Gestión de Audio, Video y Streaming para VTV, se puede alcanzar la calidad demandada, al darle solución a los errores encontrados y documentados en el proceso, verificando que tenga una correcta implementación y un buen levantamiento de los requisitos funcionales que exige el cliente. Como objetivo principal de este argumento se quiere lograr que el producto llegue con un mínimo de defectos posibles a manos de los usuarios finales y de esta forma salga con la calidad requerida antes de ser liberado y entregado a los clientes.

Debido a lo antes planteado se tiene como **Problema a Resolver**: ¿Cómo detectar los posibles errores en la construcción del Sistema de Gestión de Audio, Video y Streaming para VTV que permita la verificación y validación de los requisitos del cliente?

Dando salida al siguiente **Objeto de Estudio**: Proceso de pruebas de software.

Delimitando como **Objetivo General**: Diseñar y aplicar pruebas de software al Sistema de Gestión de Audio, Video y Streaming para VTV que permita la verificación y validación del sistema.

Se enmarca el **Campo de Acción** en: El proceso de pruebas al Sistema de Gestión de Audio, Video y Streaming para VTV.

Se plantea como **Idea a Defender**: El diseño e implementación de las pruebas al Sistema de Gestión de Audio, Video y Streaming para VTV, garantizará detectar la mayor cantidad de defectos posibles.

Para el desarrollo de la investigación según lo antes planteado se definieron las siguientes **Tareas de Investigación**:

- Caracterizar los diferentes tipos de pruebas.
- Seleccionar las pruebas que serán aplicadas.

- Describir el proceso de pruebas para el Sistema de Gestión de Audio, Video y Streaming para VTV.
- Elaborar el plan de pruebas para el Sistema de Gestión de Audio, Video y Streaming para VTV.
- Diseñar los Casos de Prueba.
- Aplicar las pruebas al Sistema de Gestión de Audio, Video y Streaming para VTV.
- Caracterizar el estado del producto una vez finalizado el proceso de pruebas.

Con el propósito de resolver el problema y lograr los objetivos de la investigación se utilizaron métodos científicos como: los Métodos Teóricos los cuales permiten el estudio de las características del objeto de investigación que no son observables.

Analítico-Sintético: Método que tiene como objetivo analizar las teorías, documentos que permitan extraer los elementos más importantes del proceso de pruebas de Software.

Análisis Histórico-Lógico: Método que permite constatar teóricamente cómo ha evolucionado la calidad del software conjuntamente con el proceso de pruebas ya existente, adaptándolo al desarrollo de la investigación. Lo histórico se da cuando se observan los resultados en investigaciones realizadas sobre el tema y sus conclusiones. Mientras que lo lógico permite extraer la información más importante de lo que se estudia.

En la investigación se utilizarán dos métodos Empíricos, el método de *Entrevista* con el Analista y Jefe del proyecto de Captura y Catalogación de Medias, cumpliendo como objetivo fundamental el conocimiento de la misión y visión del mismo, sabiendo cuáles serían los casos de uso a los que se les aplicarán las pruebas seleccionadas en el presente trabajo de diploma. El método de *Observación* para evidenciar a través de la observación si los requisitos fueron completados satisfactoriamente.

El primer capítulo, **Fundamentación teórica** aborda conceptos y criterios de varios autores que se corresponden con la calidad del software, el aseguramiento de la misma y lo que va enlazado al proceso de pruebas, haciendo énfasis en la metodología de desarrollo RUP para utilizar el proceso de pruebas que define.

El segundo capítulo, **Planificación del Proceso de Prueba** se enmarcará en la creación de un plan de pruebas, se definen estrategias a seguir, certificando los procedimientos de pruebas que serán la base para diseñar y ejecutar los casos de pruebas.

El tercer capítulo, **Resultados Obtenidos** se hace un análisis de los resultados obtenidos en la aplicación de las pruebas al software haciéndose un resumen de los principales errores encontrados durante las mismas.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción

El capítulo que a continuación se expone aborda los temas fundamentales de la investigación. Haciendo mención de conceptos y definiciones de varios autores, relacionados con la calidad del software y el proceso de pruebas. Se identifican los tipos de pruebas existentes y sus técnicas para la verificación y validación del producto final, teniendo en cuenta la metodología usada.

1.2 Calidad del software. Conceptos y definiciones.

La calidad del software a pesar de no tener una definición estándar, cada autor da su punto de vista al respecto. Permitiendo así que cada quien adecue sus conceptos a su favor.

Crosby (1979) definió la calidad del software como la" conformidad con los requisitos" **(Lovelle, 1999)**.

"Concordancia del software producido con los requisitos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requisitos implícitos no establecidos formalmente, que desea el usuario"**.** Según libro de Ingeniería del software **Pressman (1998) (Hernandez, 2008).**

La definición más utilizada es la brindada por el IEEE: "La calidad del software es el grado con el que un sistema, componente o proceso cumplen los requisitos especificados y las necesidades o expectativas del cliente o usuario" **(Febles, 2006)**.

Los requisitos de un sistema a desarrollar son los que determinan de qué manera van a ser finalizados. Si se cumple al pie de la letra cada exigencia, las probabilidades son altas con respecto a la obtención de una mayor calidad. Muchas veces los clientes no saben expresar concretamente lo que desean en un producto determinado, por lo que la calidad del mismo va a depender tanto de lo que se demanda evidentemente, es decir, los requisitos explícitamente planteados, como los implícitos.

Hay autores que opinan que la calidad de un producto se evidencia cuando el mismo está funcionando, si es eficiente, flexible, correcto, confiable, comprensible, portable, utilizable y seguro, entonces cumple cualitativamente con lo que finalmente el cliente desea.

Si bien para algunos autores la condición necesaria para alcanzar la calidad del software es contar con un buen levantamiento de requisitos, para otros esta solo puede medirse si el software está terminado y sus características lo hacen un software

íntegro. Ambas ideas no son mutuamente excluyentes pues el primer planteamiento no deja de estar implícito en el segundo.

Un producto será confiable cuando, además de la calidad que requiere, se logre su mantenimiento a lo largo del ciclo de su desarrollo, siendo este el principal objetivo del aseguramiento de la calidad.

1.3 Aseguramiento de la calidad

Lo primero que se debe conocer es que el aseguramiento de la calidad de software (Software Quality Assurance (SQA)): "Conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) satisfará los requisitos dados de calidad" **(Granja, 2007)**.

El aseguramiento de calidad del software se diseña para cada aplicación antes de comenzar a desarrollarla. Pretende dar confianza en que el producto tiene calidad y está presente en:

- Métodos y herramientas de análisis, diseño, programación y prueba.
- Inspecciones técnicas formales en todos los pasos del proceso de desarrollo del software.
- Estrategias de prueba multiescala.
- Control de la documentación del software y de los cambios realizados.
- Procedimientos para ajustarse a los estándares (y dejar claro cuando se está fuera de ellos).
- Mecanismos de medida (métricas).
- Registro de auditorías y realización de informes.

Para el aseguramiento de la calidad del software las actividades que se definen son:

- Métricas de software para el control del proyecto.
- Verificación y validación del software a lo largo del ciclo de vida (Incluye las pruebas y los procesos de revisión e inspección).
- La gestión de la configuración del software.

Algunos métodos del aseguramiento:

- Revisiones técnicas y de gestión (su objetivo es la evaluación).
- Inspección (su objetivo es la verificación).
- Pruebas (su objetivo es la validación).

- Auditorias (su objetivo es la confirmación del cumplimiento) **(Lovelle, 1999)**.

El aseguramiento de la calidad del software se podría decir que es la herramienta que gestiona la calidad. Asegurando la calidad de los resultados de cada una de las fases del ciclo de vida del software y con esto, testificar la calidad del producto final.

Todo desarrollo de software es riesgoso y difícil de controlar. Para lograr una mayor calidad en el producto que se está desarrollando, este debe realizarse basándose en una metodología de desarrollo de software, debido a que las mismas imponen un proceso disciplinado sobre el desarrollo de software, con el fin de hacerlo más predecible y eficiente.

1.4 Metodología de desarrollo de software

El objetivo que se persigue con el uso de una metodología es producir software de alta calidad dentro de una planificación establecida, siendo esto fundamental, pues en ocasiones los proyectos se realizan sin metodología alguna, trayendo como consecuencia que se entreguen fuera de tiempo y/o la documentación del software no quede detallada.

¿Qué se entiende por metodología de desarrollo de software?

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.

Características deseables de una metodología

- Existencia de reglas predefinidas.
- Cobertura total del ciclo de desarrollo.
- Verificaciones intermedias.
- Planificación y control.
- Comunicación efectiva.
- Utilización sobre un abanico amplio de proyectos.
- Fácil formación.
- Herramientas CASE.
- Actividades que mejoren el proceso de desarrollo.
- Soporte al mantenimiento.
- Soporte de la reutilización de software.

Clasificación de las metodologías

- Estructuradas, no formal para un sistema de gestión.
  - Orientadas a Procesos.
  - Orientadas a Datos: Jerárquicos y No jerárquicos.
  - Mixtas.
- Orientadas a objetos, formal para un sistema de tiempo real **(Quesada, 2006-2007)**.

La metodología que se utilizó durante el desarrollo del producto Sistema de Gestión de Audio, Video y Streaming para VTV fue el Proceso Unificado de Desarrollo (RUP).

1.4.1 RUP. Proceso Unificado de Desarrollo

El proceso unificado de desarrollo (RUP) es una metodología para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan 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.

Figura 1: Proceso Unificado de Desarrollo (RUP)

Características principales de RUP

- "Guiado por los Casos de Uso: Los Casos de Uso son el instrumento para validar la arquitectura del software y extraer los casos de prueba."
- "Centrado en la arquitectura: Los modelos son proyecciones del análisis y el diseño constituye la arquitectura del producto a desarrollar."
- "Iterativo e incremental: Durante todo el proceso de desarrollo se producen versiones incrementales (que se acercan al producto terminado) del producto en desarrollo" **(Ferrer, 2002)**.

RUP se divide en cuatro fases:

- Inicio (define el alcance del proyecto).
- Elaboración (definición, análisis, diseño).
- Construcción (implementación).
- Transición (fin del proyecto y puesta en producción).

Disciplinas con que RUP cuenta:

- Modelamiento del negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requisitos: 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 (requisitos), 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 **(Silva, 2010)**.

Durante las cuatro fases de RUP se integran las pruebas de software, en la fase de Construcción se puede observar que luego de la implementación se realizan las pruebas, para ello RUP propone un proceso que incluye actividades como:

Planificar las pruebas: Esta actividad ejecuta las tareas que organizan y dirigen las pruebas de software. Es la encargada de planificar los recursos y esfuerzos de las pruebas en cada iteración. Describe la estrategia de prueba que identifica los tipos de pruebas que se van a realizar, cómo hacerlo y cuándo ejecutarlas.

Diseñar las pruebas: Esta actividad permite el diseño de los casos de pruebas que se van a realizar, además de identificar los procedimientos a seguir para elaborar dichos casos de pruebas.

Ejecutar las pruebas: Esta actividad permite establecer las pruebas, realizar pruebas de Integración y Sistema, probablemente en repetidas ocasiones (iterativamente), a fin de ver si el desconocimiento de cualquier factor podría afectar a los resultados.

Evaluar las pruebas: El propósito de esta actividad es evaluar sus esfuerzos. Establecen una comparación entre todos los resultados obtenidos y los especificados en el plan de pruebas. De esta forma se determina el número de pruebas que se deben realizar además del nivel de calidad de software **(Jacobson, 2000)**.

Esta serie de actividades son desarrolladas por cinco roles que intervienen en dicho proceso, como son: Administrador de la calidad, Diseñador de pruebas, Analista de pruebas, Probador, y Revisor técnico.

En el flujo de trabajo de pruebas se generan artefactos como:

- **Modelo de pruebas**

  Está compuesto por procedimientos, componentes, así como los casos de pruebas los cuales brindan una descripción de cómo van a ser probados los elementos de la implementación.

- **Caso de prueba**

  Especifica una vía o forma para realizarle pruebas al sistema. Incluye condiciones, aspectos de entrada y los resultados correspondientes con los que se proseguirá la prueba.

- **Procedimiento de prueba**

  Conforma las instrucciones capaces de especificar cómo se realizan los casos de prueba o cómo interactuar con una herramienta de pruebas. Describe los aspectos fundamentales para lograr un buen dominio y manejo de las pruebas.

- **Componente de prueba**

Automatiza los procedimientos de pruebas y verifican los componentes en el modelo de implementación.

- **Plan de prueba**

Es el documento que describe los tipos de pruebas que se llevarán a cabo, sus objetivos, los niveles, los recursos y la planificación en general del proceso de pruebas.

- **Defecto**

Es una falla o un problema detectado durante revisiones al software. Un síntoma de un funcionamiento negativo del sistema.

- **Evaluación de prueba**

Es el proceso que analiza el estado de los defectos encontrados, es decir, consiste en la evaluación de los resultados obtenidos en las pruebas **(Jacobson, 2000).**

Beneficios que aporta RUP

- Permite desarrollar aplicaciones sacando el máximo provecho de las nuevas tecnologías, mejorando la calidad, el rendimiento, la reutilización, la seguridad y el mantenimiento del software mediante una gestión sistemática de los riesgos.
- Permite la producción de software que cumpla con las necesidades de los usuarios, a través de la especificación de los requisitos, con una agenda y costo predecible.
- Enriquece la productividad en equipo y proporciona prácticas óptimas de software a todos sus miembros.
- Permite llevar a cabo el proceso de desarrollo práctico, brindando amplias guías, plantillas y ejemplos para todas las actividades críticas.
- Proporciona guías explicitas para áreas tales como modelado de negocios, arquitectura Web, pruebas y calidad. También proporciona guías para desarrollar en plataformas IBM WebSphere y Microsoft Web Solution para acelerar el desarrollo de los proyectos.
- Se integra estrechamente con herramientas Rational, permitiendo a los equipos de desarrollo aprovechar todas las ventajas de las características de los productos Rational, el Lenguaje de Modelado Unificado (UML) y otras prácticas óptimas de la industria.

- Unifica todo el equipo de desarrollo de software y mejora la comunicación al brindar a cada miembro del mismo una base de conocimientos, un lenguaje de modelado y un punto de vista de cómo desarrollar software.
- Optimiza la productividad de cada miembro del equipo al poner al alcance la experiencia derivada de miles de proyectos y muchos líderes de la industria.
- No solo garantiza que los proyectos abordados serán ejecutados íntegramente sino que además evita desviaciones importantes respecto a los plazos.
- Permite una definición acertada del sistema en un inicio para hacer innecesarias las reconstrucciones parciales posteriores **(Ferrer, 2002)**.

La metodología de desarrollo permite gran organización, ayudando a mejorar la calidad, el rendimiento y reutilización del software mediante un proceso práctico. Dentro de sus disciplinas se encuentra el proceso de pruebas de software, que contribuye a que se construya un software con alta calidad y logre alcanzar las necesidades del cliente.

1.5 Introducción a las pruebas de software. Definiciones.

En el ciclo de vida de un software se realizan pruebas de software, con el fin de corregir errores que podrían comenzar desde el primer momento del proceso en que los objetivos estén planteados incorrectamente, así en sus últimos pasos, como diseño y desarrollo. Para garantizar la calidad en el desarrollo de un software primeramente se debe tener conocimiento de ¿Qué se entiende por pruebas de software?

1.5.1 Pruebas de software y sus objetivos

Prueba, es la acción y efecto de probar (hacer un examen o experimento de las cualidades de alguien o algo). Las pruebas, por lo tanto, son los ensayos que se hacen para saber cómo resultará algo en su forma definitiva, o los argumentos y medios que pretenden demostrar la verdad o falsedad de algo **(Definición de prueba, 2008)**.

Cuando se habla de las pruebas de software se pueden definir de la siguiente manera: Proceso realizado concurrentemente a través de las diferentes etapas de desarrollo de software, cuyo objetivo es apoyar la disminución del riesgo de aparición de fallas y faltas en operación **(Cárdenas, 2009)**.

La prueba del software es un elemento crítico para la garantía de la calidad del Software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces **(Usaola, 2005)**.

Como parte que es de un proceso industrial, la fase de pruebas añade valor al producto que se maneja: todos los programas tienen errores y la fase de pruebas los descubre; ahí reside su valor. El objetivo específico de la fase de pruebas es encontrar la mayor cantidad posible de errores, con un mínimo de costo y tiempo **(Mañas, 1994)**.

Un objetivo de prueba es un conjunto de interacciones entre el sistema y un caso de prueba (que reemplaza a un actor) para verificar que el comportamiento del sistema es el comportamiento definido en sus casos de uso. Los objetivos de pruebas se definen como caminos a través del modelo de comportamiento o como diagramas de actividades dónde sólo existe un único camino **(Torres, 2006)**.

Las pruebas de software definitivamente son imprescindibles para una mejor calidad en la terminación de un producto, para acercarse a un software sin errores se debe ante todo tener un conocimiento de las mismas y lograr escoger cuales deben aplicarse según se corresponda. Deben sostenerse en las diferentes etapas de desarrollo del software y como objetivo fundamental tener un buen caso de prueba que ayude a detectar el error no visto antes, controlar las deficiencias y detectarlas en el tiempo requerido, así luego corregir, ahorrando tiempo y costo. Siempre que se cumplan los objetivos de las pruebas de software se puede ver la calidad del producto una vez terminado.

1.5.3 Principios de las pruebas

Principios de pruebas:

- La prueba puede ser usada para mostrar la presencia de errores, pero nunca de su ausencia.
- Evitar casos de pruebas no planificados, no reusables y triviales a menos que el programa sea verdaderamente sencillo.
- Los casos de pruebas tienen que ser escritos no solo para condiciones de entrada válidas y esperadas sino también para condiciones no válidas e inesperadas.
- Los casos de pruebas tienen que ser escritos para generar las condiciones de salida deseadas.
- Hacer un seguimiento de las pruebas hasta los requisitos del cliente (trazabilidad).
- Plantear y diseñar las pruebas antes de generar ningún código.
- El 80% de todos los errores se centran solo en el 20% de los módulos.
- Empezar las pruebas en módulos individuales y avanzar hasta probar el sistema entero.
- No son posibles las pruebas exhaustivas.
- Para ser más efectivas deben realizarse por un equipo independiente al equipo de desarrollo.
- Se deben evitar los casos de prueba no documentados ni diseñados con cuidado.
- No deben realizarse planes de prueba suponiendo que prácticamente no hay defectos en los programas y, por tanto, dedicando pocos recursos a las pruebas **(Pressman, 2002)**.

Es muy significativo conocer y entender los principios básicos que guían las pruebas de software, antes de realizar la aplicación de métodos para el diseño de casos de prueba efectivos. Esto es de gran importancia por plantear que se realicen pruebas basadas en los requisitos funcionales con el fin de comprobar que se satisfacen las especificaciones. Además, previo a su aplicación debe existir una planificación con el propósito de obtener casos de pruebas diseñados y documentados con cuidado, evitando que los mismos sean triviales y no reusables. Es importante también por hacer uso de las pruebas para mostrar la presencia de errores, empezando por módulos individuales hasta el sistema completo, y sea más efectiva si se realiza por un equipo independiente al equipo de desarrollo.

1.5.4 Estrategias de prueba de software

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. También describe un mapa con los pasos que hay que llevar a cabo como parte de la prueba, cuando se debe planificar y realizar esos pasos, cuanto esfuerzo, tiempo y recursos se van a requerir.

Una estrategia de prueba del software debe ser lo suficientemente flexible como para promover un enfoque personalizado. Al mismo tiempo, debe ser lo adecuadamente rígido como para promover una planeación razonable y un seguimiento administrativo del avance del proyecto. Si se considera el proceso desde un punto de vista procedimental, la prueba consiste en una serie de pasos que implementan de manera secuencial. Estos pasos se describen a continuación:

- Pruebas Unitarias: se realizan con el fin de descubrir errores dentro del ámbito de un módulo.
- Pruebas de Integración: se realizan para detectar errores asociados con la interacción.
- Pruebas del Sistema: su propósito primordial es ejercitar profundamente el sistema basado en computadora.
- Pruebas de Aceptación: a partir de ellas el cliente verifica que el software funciona según sus expectativas **(Pressman, 2005)**.

Pruebas Unitarias

La prueba de unidad se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de Caja Blanca **(Vallecillo, 2004)**.

¿Por qué realizar pruebas unitarias?

• *Asegura calidad del código entregado*. Es la mejor forma de detectar errores tempranamente en el desarrollo. No obstante, esto no asegura detectar todos los errores, por tanto, las pruebas de Integración y Aceptación siguen siendo necesarias.

• *Ayuda a definir los requisitos y responsabilidades de cada método en cada clase probada*.

• *Constituye una buena forma de ejecutar pruebas de concepto*. Cuando es necesario hacer pruebas de conceptos sin integrar usar pruebas unitarias se convierte en un método efectivo.

• *Permite hacer refactoring*[1] *tempranamente en el código*. No es necesario todo un ciclo de integración para hacer refactoring en la aplicación, basta con ver cómo se comporta un caso de prueba para hacer refactoring unitario sobre la clase que se está probando en cuestión.

• *Permite incluso hacer pruebas de Stress tempranamente en el código*. Por ejemplo un método que realice una consulta SQL que exceda los tiempos de aceptación es posible optimizarla antes de integrar con la aplicación.

• *Permite encontrar errores o bugs tempranamente en el desarrollo*. Y está demostrado que mientras más temprano se corrijan los errores, menos costará corregirlos **(Rodríguez, 2006)**.

Pruebas de Integración

El objetivo de las pruebas de Integración es verificar si los componentes o subsistemas interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida, y se ajustan a los requisitos especificados en las verificaciones correspondientes. La *prueba de Integración* es una técnica sistemática para construir la estructura del programa mientras que al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción.

Existen dos tipos de integración la no incremental y la incremental. A menudo, hay una tendencia a intentar una integración *no incremental*; es decir, a combinar todos los módulos y probar todo el programa en su conjunto. El resultado puede ser un poco caótico con un gran conjunto de fallos y la consiguiente dificultad para identificar el módulo (o módulos) que los provocó.

En contra, se puede aplicar la integración *incremental* en la que el programa se prueba en pequeñas porciones en las que los fallos son más fáciles de detectar. Existen dos tipos de integración incremental, la denominada *ascendente* y *descendente*. Los pasos a seguir para cada caso son:

Integración incremental ascendente:

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

[1] Técnica de la ingeniería de software para estructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

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.

Integración incremental descendente:
1. Se usa el módulo de control principal como controlador de la prueba, creando *resguardos* (módulos que simulan el funcionamiento de los módulos que utiliza el que está probando) para todos los módulos directamente subordinados al módulo de control principal.
2. Dependiendo del enfoque de integración elegido (es decir, primero-en-profundidad, o primero-en-anchura) se van sustituyendo uno a uno los resguardos subordinados 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 **(Juristo, 2005)**.

Pruebas de 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:** Verificar los mecanismos de protección.
- **Prueba de Resistencia:** Enfrenta a los programas a situaciones anormales.
- **Prueba de Rendimiento:** Prueba el rendimiento del software en tiempo de ejecución **(Tello, 2011)**.

Pruebas de Aceptación

Estos tipos de pruebas se suelen hacer inicialmente en el entorno del desarrollador, denominadas *Pruebas Alfa*, y seguidamente en el entorno del cliente denominadas *Pruebas Beta*.

Pruebas Alfa: se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso. Las pruebas Alfa se llevan a cabo en un entorno controlado.

Pruebas Beta: se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba Beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

A la hora de realizar estas pruebas, el producto está listo para implantarse en el entorno del cliente. El usuario debe ser el que realice las pruebas, ayudado por personas del equipo de pruebas, siendo deseable, que sea el mismo usuario quien aporte los casos de prueba.
Estas pruebas se caracterizan por:

- Participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas.
- Están enfocadas a probar los requisitos de usuario, o mejor dicho a demostrar que no se cumplen los requisitos, los criterios de aceptación o el contrato. Si no se consigue demostrar esto el cliente deberá aceptar el producto.
- Corresponden a la fase final del proceso de desarrollo de software.

Es muy recomendable que las pruebas de Aceptación se realicen en el entorno en que se va a explotar el sistema (incluido el personal que lo maneje). En caso de un producto de interés general, se realizan pruebas con varios usuarios que reportarán sus valoraciones sobre el producto. Para la generación de casos de prueba de aceptación se utilizan técnicas de Caja Negra **(Juristo, 2005)**.

Pruebas de Regresión

La regresión consiste en la repetición selectiva de pruebas para detectar fallos introducidos durante la modificación de un sistema o componente de un sistema. Se efectuarán para comprobar que los cambios no han originado efectos adversos no intencionados o que se siguen cumpliendo los requisitos especificados.

En las pruebas de Regresión hay que:

- Probar íntegramente los módulos que se han cambiado.
- Decidir las pruebas a efectuar para los módulos que no han cambiado y que han sido afectados por los cambios producidos.

Este tipo de pruebas ha de realizarse, tanto durante el desarrollo cuando se produzcan cambios en el software, como durante el mantenimiento **(Juristo, 2005)**.

Las pruebas de Regresión en varias bibliografías consultadas aparecen como una prueba más dentro de las pruebas de software, encontrada fuera de los niveles, y pueden ser aplicadas en cualquiera de los mismos, dígase bajo o alto. Las bibliografías consultadas donde dicha prueba se encuentra dentro del nivel de Integración es referenciando al libro de Ingeniería de Software, un enfoque práctico, del autor Roger Pressman. En este trabajo queda plasmado según el primer criterio, ya que pueden ser realizadas en cualquier nivel incluyendo el de Integración.

1.5.5 Técnicas de pruebas

Las técnicas de pruebas permitirán examinar los detalles procedimentales del código y la lógica del programa así como la interfaz y su funcionalidad. Por medio de las diferentes técnicas se brindan criterios variados para generar casos de pruebas que provoquen fallos en los programas. Es por ello que es de vital importancia conocer dichas técnicas.

Cualquier producto de ingeniería se puede probar de dos formas:

- Pruebas de Caja Negra: Realizar pruebas de forma que se compruebe que cada función es operativa.
- Pruebas de Caja Blanca: Desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada.

Cuando se llega a la codificación muchas veces se pregunta ¿qué pruebas se debe hacer? Obteniendo respuestas como revisión de código y test unitario (pruebas de Caja Blanca).

Pruebas de Caja Blanca

Las pruebas de Caja Blanca se basan en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran

cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar.

Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas del diseño o el código. Mediante los métodos de prueba de la Caja Blanca, el ingeniero de software puede obtener casos de prueba que garanticen que:

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

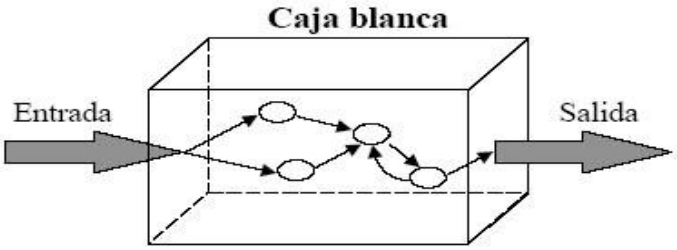

Figura 2: Enfoque de diseño de pruebas de Caja Blanca
(Mario G. Piattini, 2004)

Algunas de las técnicas de pruebas existentes enmarcadas en las pruebas de Caja Blanca son las que a continuación se explican brevemente:

Prueba del Camino Básico

Es una técnica propuesta por McCabe la cual le permite al diseñador de casos de prueba obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de un conjunto básico de caminos de ejecución. Diseñando casos de prueba que garanticen que se ejecute al menos una vez cada sentencia del programa.
Los aspectos claves de esta técnica son:

- *Grafo de flujo o grafo del programa:* representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de éste. (Cada

nodo representa una o más sentencias procedimentales y cada arista representa el flujo de control)

- *Complejidad ciclomática:* es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto de las pruebas, el cálculo de la complejidad ciclomática representa el número de caminos independientes del conjunto básico de un programa. Esta medida ofrece al probador de software un límite superior para el número de pruebas que debe realizar para garantizar que se ejecutan por lo menos una vez cada sentencia.

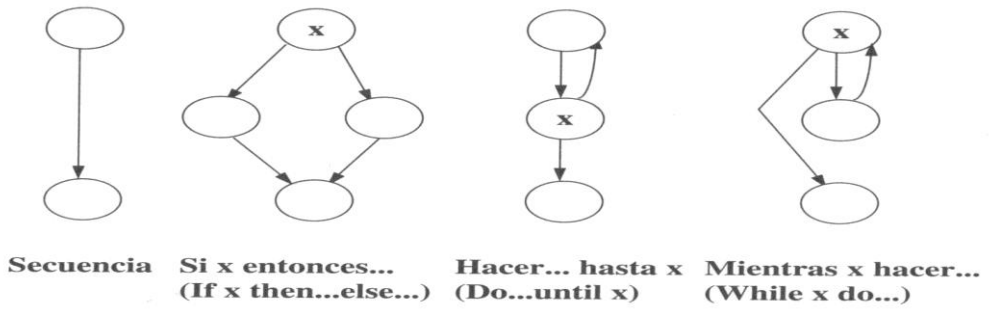

Figura 3: Grafo de flujo de las estructuras básicas de programa
(InfoMedia, 2009)

Prueba de Condición

Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. Algunos conceptos empleados alrededor de esta prueba son los siguientes:

Condición simple: es una variable lógica o una expresión relacional (E1 < *operador - relacional* > E_2).

Condición compuesta: está formada por dos o más condiciones simples, operadores lógicos y paréntesis.

En general los tipos de errores que se buscan en una prueba de condición, son los siguientes:

- Error en operador lógico (existencia de operadores lógicos incorrectos, desaparecidos, sobrantes).
- Error en variable lógica.
- Error en paréntesis lógico.

- Error en operador relacional.
- Error en expresión aritmética.

Prueba de la Estructura de control:

Dentro de este tipo de prueba se contempla el método del Camino Básico mencionado anteriormente pero además existen otras pruebas asociadas que permiten ampliar la cobertura de la prueba y mejorar su calidad. Estas son:

Prueba del Flujo de datos: selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

Prueba de Bucles: es una técnica que se centra exclusivamente en la validez de las construcciones de bucles (bucles simples, anidados, concatenados y no estructurados).

Bucles simples. Se les aplica el siguiente conjunto de pruebas:

- Pasar por alto totalmente el bucle.
- Pasar una sola vez por el bucle.
- Pasar dos veces por el bucle.
- Hacer m pasos por el bucle con $m < n$ (donde n es el número máximo de pasos permitidos por el bucle).
- Hacer n - 1, n y n + 1 pasos por el bucle.

Bucles anidados. Si se empleara el mismo enfoque de prueba de bucles simples a los bucles anidados, el número de pruebas aumentaría considerablemente por lo cual Beizer sugiere emplear el siguiente enfoque:

- Comenzar por el bucle más interior.
- Establecer o configurar los demás bucles con sus valores mínimos.
- Llevar a cabo las pruebas de bucles simples para el bucle más interior, mientras se mantienen los parámetros de iteración de los bucles externos en sus valores mínimos. Añadir otras pruebas para valores fuera de rango o excluidos.
- Progresar hacia fuera, llevando a cabo pruebas para el siguiente bucle, pero manteniendo todos los bucles externos en sus valores mínimos y los demás bucles anidados en sus valores típicos.
- Continuar hasta que se hayan probado todos los bucles.

Bucles concatenados. Estos bucles se pueden probar utilizando el enfoque de bucles simples, siempre y cuando cada uno de los bucles sea independiente del resto de lo contrario se debe emplear el enfoque de bucles anidados.

Bucles no estructurados. Siempre que sea posible estos bucles deben rediseñarse.

Prueba de Caja Negra

El método de Caja Negra se enfoca en los requisitos fundamentales del software y permite obtener entradas que prueben todos los requisitos funcionales del programa. No considera la codificación dentro de sus parámetros a evaluar, es decir, que no están basadas en el conocimiento del diseño interno del programa. Con este tipo de pruebas se intenta encontrar:

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

Los casos de prueba de la Caja Negra pretenden demostrar que:

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

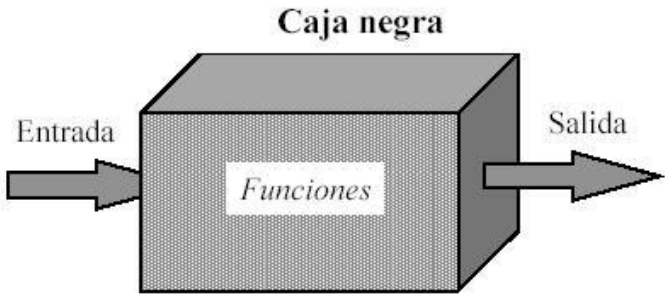

Figura 4: Enfoque de diseño de pruebas de Caja Negra
(Mario G. Piattini, 2004).

Algunas de las técnicas empleadas para desarrollar las pruebas de Caja Negra son las que a continuación se exponen:

Partición Equivalente

La Partición Equivalente es un método de prueba de Caja Negra que 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 **(Juristo, 2006)**.

El diseño de casos de prueba para Partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Con el uso de los conceptos introducidos en la sección anterior, si es posible enlazar un conjunto de objetos mediante relaciones simétricas, transitivas y reflexivas, entonces existe una clase de equivalencia. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada. Por lo general, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición booleana. Las clases de equivalencia se definen de acuerdo con las siguientes directrices:

- Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada requiere un valor específico, se definen una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y otra no válida.
- Si una condición de entrada es booleana, se definen una clase de equivalencia válida y otra no válida **(Pressman, 2005)**.

Al aplicar estas directrices para la derivación de clases de equivalencia, se desarrollarán y ejecutarán los casos de prueba para cada objeto de los datos del dominio de entrada. Los casos de prueba se seleccionan de modo que el mayor número de atributos de clase de equivalencia se ejercita una vez.

El objetivo de Partición Equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software. Se toma un riesgo porque se escoge no probar todo. Así que se necesita tener mucho cuidado al escoger las clases **(Juristo, 2006)**.

Análisis de valores límite

Los errores tienden a darse más en los límites del campo de entrada que en el centro. Por ello, se ha desarrollado el análisis de valores límite (AVL) como técnica de prueba. El análisis de valores límite lleva a una elección de casos de prueba que ejerciten los valores límite.

El análisis de valores límite es una técnica de diseño de casos de prueba que completa a la Partición Equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.

Condiciones sublímite. Las condiciones límite normales son las más obvias de descubrir. Estas son definidas en la especificación o son evidentes al momento de utilizar el software. Algunos límites, sin embargo, son internos al software, no son necesariamente aparentes al usuario final pero aún así deben ser probadas por el probador. Estas son conocidas como condiciones sublímites o condiciones límite internas. Una condición sublímite común es la tabla de caracteres ASCII, por ejemplo, si se está evaluando una caja de texto que acepta solamente los caracteres AZ y az, se debe incluir los valores en la partición inválida justo «debajo de» y «encima de» esos caracteres de la tabla ASCII, [, y {.

- *Gráfica Causa-efecto.* 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.

Un gráfico de causa-efecto es un lenguaje formal al cual se traduce una especificación. El gráfico es realmente un circuito de lógica digital (una red combinatoria de lógica), pero en vez de la notación estándar de la electrónica, se utiliza una notación algo más simple. No hay necesitad de tener conocimiento de electrónica con excepción de una comprensión de la lógica booleana (entendiendo los operadores de la lógica AND, OR, y NOT) **(Juristo, 2006)**.

Luego de ver algunos conceptos que hacen referencia al logro de la calidad de un software y las pruebas existentes para lograr dicha calidad, se puede decir que la UCI además de ser creada con el objetivo de desarrollar software también se destaca realizando diferentes pruebas a sus productos.

Pruebas de software en la UCI

En la UCI se realizan pruebas dirigidas por Calisoft (Centro para la Excelencia en el Desarrollo de Proyectos Tecnológicos), con el objetivo de evaluar la calidad de sus productos. El centro cuenta para dicha evaluación con el LIPS (Laboratorio Industrial de Pruebas de Software) donde se realizan pruebas Funcionales, de Carga y Stress, de Usabilidad, de Aceptación y Liberación. Define una estrategia de evaluación que establece como llevar a cabo las pruebas y las revisiones asociadas a los artefactos que generan los proyectos de desarrollo de software.

Si no cumpliera dicho centro su objetivo de elevar la calidad de los software desarrollados en la universidad, conjuntamente con los equipos de calidad en cada facultad, traería como consecuencia atraso en el cronograma del producto y elevaría el costo del mismo al corregirlo una vez más. En dependencia del tipo de aplicación que es el producto se le realizan las pruebas, la estrategia es lo suficientemente flexible a todo tipo de proyecto que se desarrolle en la universidad, dígase multimedias, proyectos de gestión, portales, entre otros.

1.6 Conclusiones Parciales

A partir de la revisión teórica efectuada de los conceptos y definiciones relacionados con la calidad de software, su aseguramiento y las pruebas de software, se observa una estrecha relación entre estos conceptos, siendo las pruebas de software una disciplina enfocada principalmente en la evaluación y aseguramiento de la calidad del producto, reafirmándose que el objetivo de las pruebas es mejorar la calidad del software.

Con el fin de obtener calidad en el producto, verificándolo y validándolo, se logró caracterizar las pruebas de software y sus técnicas, favoreciéndose una efectiva selección de las pruebas a aplicar.

Capítulo 2: Planificación del Proceso de Prueba.

2.1 Introducción

En el capítulo anterior luego de haberse realizado una caracterización de los diferentes tipos de pruebas, da paso a definir en este capítulo una estrategia de pruebas para el producto Sistema de Gestión de Audio, Video y Streaming para VTV.

El proceso de prueba comienza haciendo una selección de técnicas y métodos que serán aplicados, se configura el entorno de prueba y se especifica cada uno de los elementos de las pruebas definidas.

2.2 Sistema de Gestión de Audio, Video y Streaming para VTV

A partir de la entrevista realizada al Analista y Jefe del proyecto Captura y Catalogación de Medias se obtuvo información referente al trabajo que desempeñan y su enfoque, teniendo en cuenta cómo y para que se desarrolla el producto Sistema de Gestión de Audio, Video y Streaming. Es una solución encaminada a la organización, catalogación, gestión y recuperación de los archivos de video digitalizados en la televisora Venezolana de Televisión, como parte de la tarea de digitalización del archivo audiovisual que hoy asume la entidad. El proyecto cuenta con 24 casos de uso, se le realizarán pruebas a 14 de ellos.

Las funciones generales del sistema permitirán lo siguiente:

- Carga inicial de la información audiovisual digitalizada al servidor central del Centro de Datos.
- Catalogación y descripción, según los datos necesarios en VTV, de los materiales digitalizados.
- Recuperación, visualización y descarga de los archivos existentes en el centro de datos según los permisos correspondientes.
- Gestión de los datos de usuarios y roles del sistema.
- Emisión de reportes de actividad de los usuarios en el sistema.

Estas funcionalidades se encuentran dentro de cinco Módulos que son los siguientes:

- Control de Ingesta.
- Catalogación.
- Consulta y Recuperación.
- Procesamiento.
- Administración.

2.2.1 Características a probar

Las características a probar en un software son muy variables ya que dependen de las funcionalidades de cada proyecto y de los objetivos que se quiera lograr al aplicar las pruebas seleccionadas, aun así, existen atributos que no varían, por ejemplo:

- Si la aplicación cumple con las funcionalidades requeridas desde el comienzo de su desarrollo.
- Si al ejecutar la aplicación no existen errores.
- Si es comprensible para el usuario.
- Si es la interfaz de la aplicación amigable a la vista del cliente y fácil de usar.
- Si se cuenta con una documentación detallada que corresponda con la aplicación.
- Si se cuenta con un código fuente entendible para otros programadores permitiendo la reutilización de código.

Es medible la aplicación si está bien documentada, además de ver si su código fuente es entendible para otros programadores permitiendo la reutilización de código, además de su mantenimiento para que la aplicación perdure.
Aun así se medirá, para el producto Sistema de Gestión de Audio, Video y Streaming para VTV, que sus funcionalidades generales se ejecuten tal y como se espera.

2.3 Planeación de las pruebas

El Plan de pruebas es una guía donde se establece paso a paso como se realizarán las pruebas. De esta manera, lograr que sean satisfactorias y tengan resultados con la mayor calidad posible.
En el Plan de pruebas definido para el proyecto Captura y Catalogación de Medias se identifican los elementos que serán probados, y una estrategia de pruebas que será aplicada con el objetivo de lograr un mejor diseño de los Casos de Prueba. Se definen recursos requeridos, evolución y alcance de las pruebas. De esta manera, se incrementa la probabilidad de encontrar errores en el sistema.

2.3.1 Objetivos

Ejecutar las pruebas en el proyecto Captura y Catalogación de Medias tiene como objetivo principal detectar errores en el sistema mediante los Casos de Uso ya definidos. Del modo en que se puedan corregir dichos errores contribuyendo a que el sistema tenga mayor calidad funcionalmente.

Se tiene en cuenta que un buen Caso de Prueba tiene una alta probabilidad de encontrar un error que el equipo de desarrollo no haya tenido en cuenta.

2.3.2 Escenario de las pruebas

Las pruebas a realizar se definieron en los escenarios siguientes:

Inicialmente, la prueba de Unidad se centra en cada módulo individualmente, comprobando que funcionan apropiadamente como una unidad, haciendo un uso intensivo de las técnicas de pruebas de Caja Blanca, viéndose la técnica del Camino Básico en los casos de prueba diseñados. Para los casos de uso críticos de los módulos del proyecto estos son fundamentales por un correcto funcionamiento del sistema.

A continuación, se deben ensamblar o integrar los módulos para formar el paquete de software completo, en este caso específicamente la integración ascendente, proporcionando que los módulos se integren de abajo hacia arriba. Durante la integración, las técnicas que más prevalecen son las de diseño de caso de prueba de Caja Negra (Partición Equivalente), probando cada una de las entradas con las salidas esperadas.

Luego que el software se ha integrado se realizan pruebas de Sistema, comprobando los criterios de validación, requisitos funcionales, de comportamiento y de rendimiento, para verificar que cada elemento acopla de manera adecuada, haciendo uso de la técnica de Caja Negra (Partición Equivalente), realizando prueba de Stress para evidenciar el rendimiento del software en tiempo de ejecución, prueba de Usabilidad para comprobar que los usuarios interactúan fácilmente con la interfaz del software y por último de Seguridad de la manera que se pueda lograr que solo hagan uso de la aplicación aquellos que tienen los permisos.

Por último se dará lugar a las pruebas de Aceptación realizadas por usuarios finales, conjuntamente o no, con los desarrolladores como fase final del proceso de desarrollo de software.

2.3.3 Script de pruebas

Para realizar las pruebas se hace necesario hacer usos de recursos tangibles dentro de los que se encuentran los de software y hardware presentados a continuación:

Especificaciones de Software:

- **En las estaciones cliente:**
    - Sistema Operativo: Cualquier SO.

- Navegador: Cualquier navegador excepto Safari (óptimo Mozilla Firefox 3.6 o superior).

- **En los servidores:**
  - Sistema Operativo: OpenSuse 11.3.
  - Sistema Gestor de Base de Datos:

    PostgreSQL 8.4 (server BD) Licencia Código Abierto. TPL.

    pgAdmin3, Licencia Código Abierto. TPL.

- **Aplicación Servidor de Streaming:**
  - Flumotion v0.6.1, Licencia GPL.
  - Apache Server, Licencia ASF, compatible con GPL.

- **Aplicación Servidor de Codificación de Datos:**
  - Biblioteca Ice 3.4.0, Licencia GNU v2.
  - Framework Qt, Licencia Qt GNU LGPL v. 2.1.
  - MEncoder, Licencia GPL.
  - ffmpeg2theora v2.7, Licencia GNU LGPL.
  - Ffmpeg, Licencia GPL/LGPL.

- **Aplicación Servidor de Indexación de Datos:**
  - Framework Qt, Licencia Qt GNU LGPL v. 2.1.
  - OpenCV v2.1, Licencia BSD.

- **Aplicación Servidor de Aplicaciones.**
  - Framework Qt, Licencia Qt GNU LGPL v. 2.1.
  - Biblioteca Ice 3.4.0, Licencia GNU v2.
  - SQLite, sin licencia.

- **Aplicación Servidor Web**
  - Apache Server, Licencia ASF, compatible con GPL.
  - Framework Symphony.
  - Framework Dojo.

Especificaciones de Hardware:

- **En las estaciones cliente:**
  - Procesador: Core2Duo.
  - Memoria: 2 GB.
  - Almacenamiento: 160 GB.
- **En los servidores:**

  Servidor de Base de Datos:
  - Arquitectura 64 bits (x64).
  - Procesador: 2 QuadCore.
  - Memoria: 16 GB.
- **Servidor de Streaming:**
  - Arquitectura 64 bits (x64).
  - Procesador: 2 QuadCore.
  - Memoria: 16 GB.
- **Servidor de Codificación:**
  - Arquitectura 64 bits (x64).
  - Procesador: 2 QuadCore.
  - Memoria: 8 GB.
- **Servidor de Indexación:**
  - Arquitectura 64 bits (x64).
  - Procesador: 2 QuadCore.
  - Memoria: 8 GB.
- **Servidor de Aplicaciones:**
  - Arquitectura 64 bits (x64).
  - Procesador: 1 QuadCore.
  - Memoria: 8 GB.

2.3.4 Casos de Uso seleccionados para el proceso de pruebas

Requisitos generales: Los Casos de Prueba a realizar dependen de los requisitos planteados inicialmente por el cliente, ellos determinan la funcionalidad del sistema y componen condiciones que deben cumplir, formando parte de la documentación del proyecto desde sus inicios. Una de las funciones principales en el proceso de pruebas, que se le aplica al proyecto, es verificar su correcta implementación, para cuando el cliente reciba su producto pueda ver que sus funcionalidades se realizan de manera correcta. Los casos de uso que se han seleccionado para el proceso de pruebas son los críticos para el producto Sistema de Gestión de Audio, Video y Streaming para VTV:

- CU Autenticar usuario.
- CU Configurar acceso LDAP.
- CU Buscar usuario LDAP.
- CU Gestionar usuario.
- CU Buscar material.
- CU Buscar material a catalogar.
- CU Catalogar material.
- CU Administrar subclip.
- CU Reproducir material.
- CU Descargar material.
- CU Atender Solicitud.
- CU Insertar Solicitud.
- CU Modificar Solicitud.
- CU Eliminar Solicitud.

2.3.5 Estrategia de pruebas

Por los recursos con que cuenta el proyecto Captura y Catalogación de Medias las pruebas seleccionadas serán aplicadas por un Diseñador de Casos de Prueba, el mismo asumirá las funciones de otros roles encargándose de planificar y diseñar las pruebas. También realiza las pruebas al sistema y evalúa los resultados. Se desarrollan actividades fundamentales en el flujo de trabajo de pruebas como son: planificar, diseñar, aplicar y evaluar las pruebas. Con dichas actividades se generan, para el proyecto, artefactos como: Plan de pruebas, Casos de pruebas y Evaluación de las pruebas recogidas en el Documento de No Conformidades. Las pruebas seleccionadas para aplicar y sus procedimientos son:

Selección de las pruebas a aplicar

Pruebas de Caja Blanca. Técnica de Camino Básico

¿En qué consiste la técnica de Camino Básico?

Está basada en una medida cuantitativa del software denominada complejidad ciclomática (McCabe, 1976), que define el número de caminos independientes del programa, siendo esta una cota superior para el número de casos de prueba que se deben realizar para asegurar coberturas de sentencias. La técnica propone generar casos de pruebas que ejecuten todos los caminos básicos independientes, es decir, obtener un caso para cada camino **(Tuya, 2007)**.

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 **(Pressman, 2000)**.

A cada nodo que se genera de esta forma se le denomina nodos predicados. A continuación se muestra un ejemplo donde se evidencian los mismos antes mencionados.

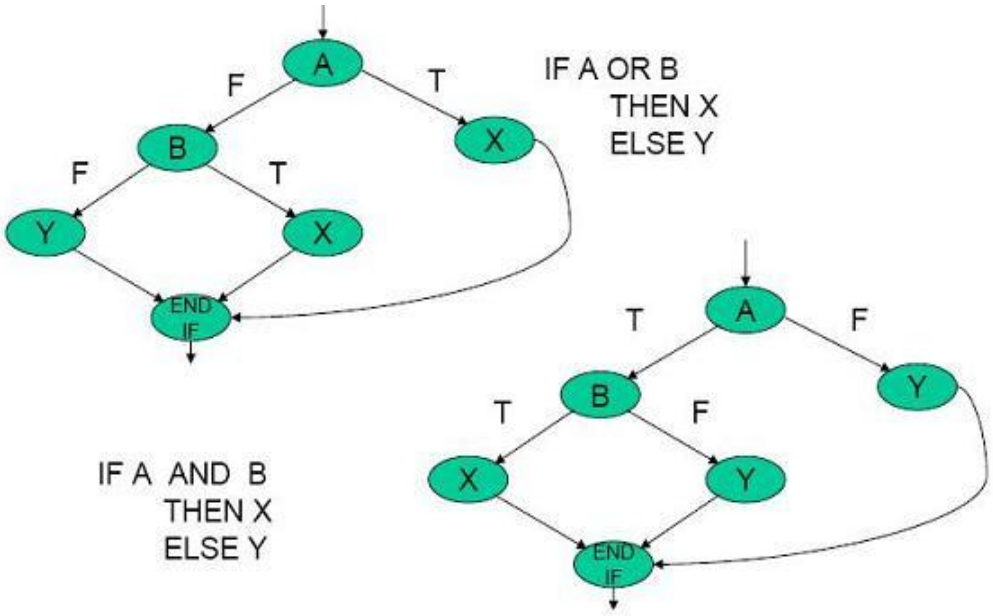

Figura 5: Grafos asociados a condiciones lógicas compuestas (Bravo, 2007).

Cada construcción lógica de un programa tiene una representación.

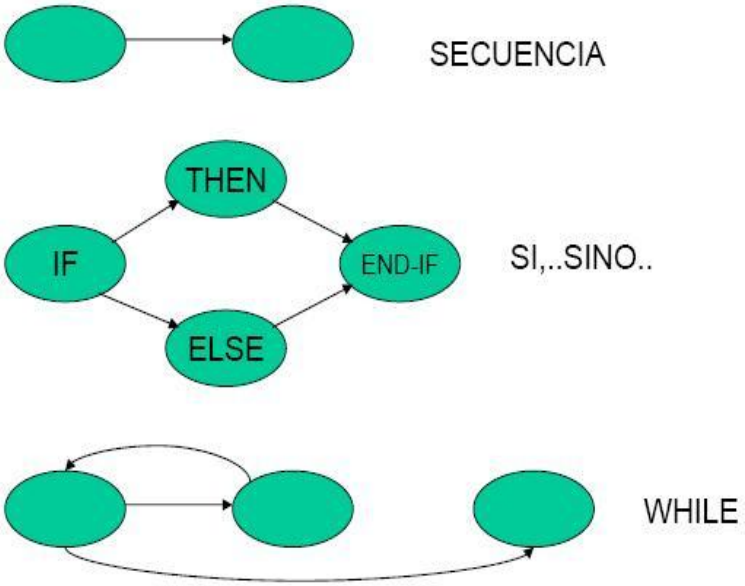

Figura 6: Representación del grafo de flujo de las estructuras de control
(**Bravo, 2007)**

¿Cómo calcular la complejidad ciclomática?

La complejidad ciclomática:

- Es una **medida del software** que aporta una valoración cuantitativa de la complejidad lógica de un programa.
- Dentro del contexto del método de prueba del Camino Básico define el **número de caminos independientes** de un programa.
- Por **camino independiente** se entiende aquel que introduce un nuevo conjunto de sentencias o una nueva condición. En términos del grafo, por una arista que no haya sido recorrida antes.
- Brinda una **cota o límite superior** para el número de casos de prueba.

Formas de cálculo:

- El número de regiones del grafo es igual a la complejidad ciclomática.
- V(G)=A-N+2
  - Donde A es el número de aristas y N es el número de nodos contenidos en el grafo.
- V(G)=P+1
  - Donde P es el número de nodos predicados contenidos en el grafo.

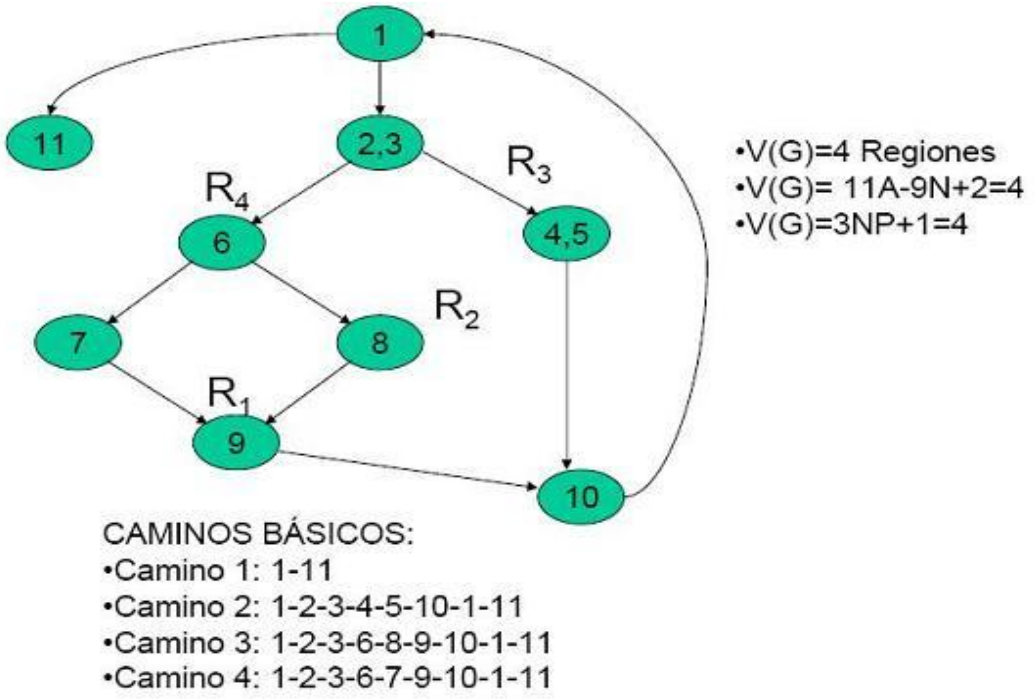

Figura 7: Complejidad ciclomática. Caminos básicos
(Bravo, 2007).

¿Cómo determinar el conjunto básico de caminos independientes?

Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una condición, respecto a los caminos existentes. En términos del diagrama 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:

- 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.

- 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.
- 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.
- 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.

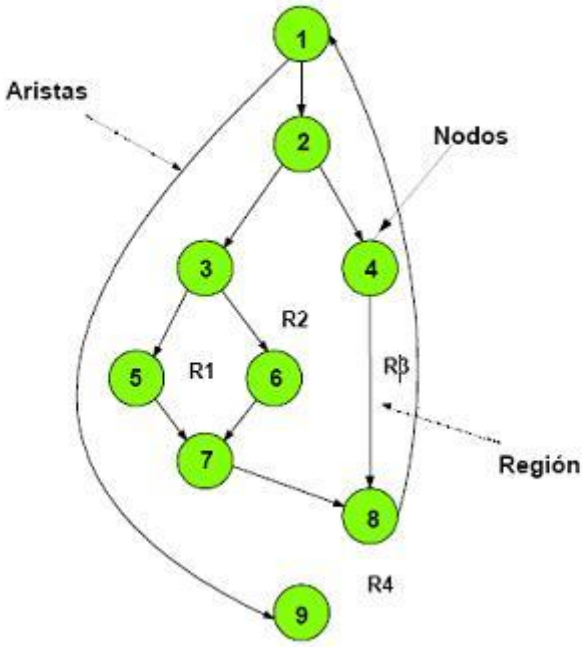

Figura 8: Número de regiones del grafo de flujo
(Juristo, 2006).

Así por ejemplo, para la del grafo de la Figura 8 los cuatro posibles caminos independientes generados serían:

Camino 1: 1-9
Camino 2: 1-2-4-8-1-9
Camino 3: 1-2-3-5-7-8-1-9
Camino 4: 1-2-5-6-7-8-1-9

Estos cuatro caminos constituyen el *Camino Básico* para el grafo de flujo correspondiente.

Derivar los casos de prueba que fuerzan la ejecución de cada camino.

El último paso es construir los casos de prueba que fuerzan la ejecución de cada camino. Una forma de representar el conjunto de casos de prueba es como se muestra en la Tabla 1.

Número del Camino	Caso de Prueba	Resultado Esperado

Tabla 1: Posible representación de casos de prueba para pruebas estructurales (Risoto, 2007)

Es seleccionada esta técnica por las características que posee la misma de poder detectar un error no encontrado hasta el momento, en el código, por los desarrolladores. Cuando se aplica este método se asegura que los casos de prueba diseñados ejecuten todas las sentencias del programa al menos una vez y que las condiciones sean probadas independientemente de su valor (verdadero o falso), además le aprueba al diseñador de casos de prueba una medida de la complejidad lógica de un diseño procedimental teniendo como ayuda esa medida para la definición de los caminos básicos posibles de ejecución. Es la técnica más completa y la que mayor seguridad le va a ofrecer al probador por los resultados que brinda en correspondencia con las otras técnicas de pruebas de Caja Blanca.

Pruebas de Caja Negra. Técnica de Partición Equivalente

¿En qué consiste la técnica de Partición Equivalente?

La partición de equivalencia es un método de prueba de Caja Negra que divide el campo de entrada de un programa en *clases de datos* de los que se pueden derivar casos de prueba. La Partición Equivalente se dirige a una definición de casos de prueba que descubran *clases de errores*, reduciendo así el número total de casos de prueba que hay que desarrollar.

En otras palabras, este método intenta dividir el dominio de entrada de un programa en un número finito de *clases de equivalencia.* De tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada clase es equivalente a una prueba realzada con cualquier otro valor de dicha clase. Esto quiere decir que si el caso de prueba correspondiente a una clase de equivalencia detecta un error, el resto de los casos de prueba de dicha clase de equivalencia deben detectar el mismo error. Y viceversa, si un caso de prueba no ha detectado ningún error, es de esperar que ninguno de los casos de prueba correspondientes a la misma clase de equivalencia encuentre ningún error.

El diseño de casos de prueba según esta técnica consta de dos pasos:

1. Identificar las clases de equivalencia.

2. Identificar los casos de prueba.

Identificar las clases de equivalencia

Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Las clases de equivalencia se identifican examinando cada condición de entrada (normalmente una frase en la especificación) y dividiéndola en dos o más grupos. Se definen dos tipos de clases de equivalencia, las *clases de equivalencia válidas*, que representan entradas válidas al programa, y las *clases de equivalencia no válidas*, que representan valores de entrada erróneos. Estas clases se pueden representar en una tabla como la Tabla 2.

Condición Externa	Clases de Equivalencia Válidas	Clases de Equivalencia No Válidas

Tabla 2: Tabla para la identificación de clases de equivalencia
(Risoto, 2007)

Identificar los casos de prueba

El objetivo es minimizar el número de casos de prueba, así cada caso de prueba debe considerar tantas condiciones de entrada como sea posible. No obstante, es necesario realizar con cierto cuidado los casos de prueba de manera que no se enmascaren

faltas. Así, para crear los casos de prueba a partir de las clases de equivalencia se han de seguir los siguientes pasos:

- Asignar a cada clase de equivalencia un número único.
- Hasta que todas las clases de equivalencia hayan sido cubiertas por los casos de prueba, se tratará de escribir un caso que cubra tantas clases válidas no incorporadas como sea posible.
- Hasta que todas las clases de equivalencia no válidas hayan sido cubiertas por casos de prueba, escribir un caso para cubrir una única clase no válida no cubierta.

La razón de cubrir con casos individuales las clases no válidas es que ciertos controles de entrada pueden enmascarar o invalidar otros controles similares. Por ejemplo, si se tienen dos clases válidas: "introducir cantidad entre 1 y 99" y "seguir con letra entre A y *Z*", el caso *105 1* (dos errores) puede dar como resultado *105 fuera de rango de cantidad*, y no examinar el resto de la entrada no comprobando así la respuesta del sistema ante una posible entrada no válida **(Juristo, 2006)**.

Se elige la técnica Partición Equivalente a aplicar debido a que se reduce el número de casos de pruebas al considerar tantas condiciones de entradas como sea posible, teniendo en cuenta las entradas válidas e inválidas y la realización correcta de casos de pruebas se pueden disminuir la presencia de errores. Por tal razón aplicando el diseño de casos de pruebas con la técnica de Partición Equivalente se puede conseguir una prueba más completa y descubrir y corregir el mayor número de errores antes de que se comiencen las pruebas del cliente.

Pruebas de Integración

En el capítulo anterior se habían visto las pruebas de Integración, de las cuales se existen la Incremental y la no Incremental.

Incremental:

- Se sigue el diseño modular
- Se solapa con la prueba de unidad

Lo que significa que se prueban por parte los módulos, es decir, en pequeñas porciones para que de esta manera sea mucho más fácil la detección de fallas.

De la Incremental existen dos tipos:

- Ascendente
- Descendente

No incremental (Big-bang). Significa que si los módulos son combinados y se quiere probar el programa en su conjunto suele ser un desastre a la hora de querer obtener un buen resultado, ya que se obtienen distintos fallos y no se sabe específicamente de que módulo salió.

Integración Ascendente: las pruebas de integración ascendente, como su nombre lo indica, consiste básicamente en tratar el módulo atómico (módulo de nivel jerárquico más bajo) hacia arriba. Dado que los módulos son integrados de abajo hacia arriba, el procesamiento requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardo.

Ventaja:

- No se necesitan resguardos

Se selecciona la prueba de Integración Ascendente porque permite ir integrando cada uno de los módulos en la medida en que se terminan y probar cada una de las funcionalidades de los mismos para eliminar cualquier inconveniente en la integración con los demás módulos. Además de los módulos ser integrados de arriba hacia debajo de manera jerárquica tal y como lo hace dicha prueba.

Pruebas de Sistema:

Pruebas de Stress

¿Qué se entiende por prueba de Stress?

Es el proceso de poner demanda en un sistema o dispositivo y medir su respuesta. Se consideran pruebas no funcionales.

¿Por qué hacer pruebas de Stress?

Reducir el riesgo de “caídas del sistema”

- Aprovechar los recursos, de la tecnología de la información, más eficientemente.
- Conocer los límites que soporta el sistema.
- Permite tomar decisiones sobre configuraciones de hardware, ajustes de software y selección de arquitecturas.
- Los fallos por estos motivos suelen ser muy costosos.

En general los objetivos suelen ser

Mejorar:

- Rendimiento.
- Escalabilidad.
- Estabilidad **(Natalia Juristo, 2005)**.

Se selecciona la prueba de Stress dentro de las pruebas de Sistema porque al trabajar el proyecto una aplicación web se deben validar estabilidad y rendimiento. Este tipo de prueba permite demostrar la rapidez y calidad con que el sistema muestra su información. La herramienta automatizada que se utiliza en la UCI, por el Centro de Calidad para Soluciones Tecnológicas en el departamento de Pruebas de Software, para aplicar dicha prueba es JMeter.

Pruebas de Usabilidad

Delimitación del Concepto de Usabilidad

Usabilidad se define en el estándar ISO 9241 como “el grado en el que un producto puede ser utilizado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un determinado contexto de uso”, y en el estándar ISO 14598-1 se define calidad de uso de forma análoga. Como se puede comprobar, en esta definición se liga la usabilidad de un sistema a usuarios, necesidades y condiciones específicas. Por tanto, la usabilidad del sistema no es un atributo inherente al software, no puede especificarse independientemente del entorno de uso y de los usuarios concretos que vayan a utilizar el sistema.

Atributos de Usabilidad

La usabilidad es una cualidad demasiado abstracta como para ser medida directamente. Para poder estudiarla se descompone habitualmente en los siguientes cinco atributos básicos:

- **Facilidad de aprendizaje**: Cuán fácil es aprender la funcionalidad básica del sistema, como para ser capaz de realizar correctamente la tarea que desea realizar el usuario. Se mide normalmente por el tiempo empleado con el sistema hasta ser capaz de realizar ciertas tareas en menos de un tiempo dado (el tiempo empleado habitualmente por los usuarios expertos). Este atributo es muy importante para usuarios noveles.

- **Eficiencia**: El número de transacciones por unidad de tiempo que el usuario puede realizar usando el sistema. Lo que se busca es la máxima velocidad de realización de tareas del usuario. Cuanto mayor es la usabilidad de un sistema, más rápido es el usuario al utilizarlo, y el trabajo se realiza con mayor rapidez. Nótese que eficiencia del software en cuanto su velocidad de proceso no implica necesariamente eficiencia del usuario en el sentido en el que aquí se ha descrito.
- **Recuerdo en el tiempo**: Para usuarios intermitentes (que no utilizan el sistema regularmente) es vital ser capaces de usar el sistema sin tener que aprender cómo funciona partiendo de cero cada vez. Este atributo refleja el recuerdo acerca de cómo funciona el sistema que mantiene el usuario, cuando vuelve a utilizarlo tras un periodo de no utilización.
- **Tasa de errores**: Este atributo contribuye de forma negativa a la usabilidad de un sistema. Se refiere al número de errores cometidos por el usuario mientras realiza una determinada tarea. Un buen nivel de usabilidad implica una tasa de errores baja. Los errores reducen la eficiencia y satisfacción del usuario, y pueden verse como un fracaso en la transmisión al usuario del modo de hacer las cosas con el sistema.
- **Satisfacción**: Éste es el atributo más subjetivo. Muestra la impresión subjetiva que el usuario obtiene del sistema.

¿Por qué aplicar prueba de Usabilidad?

La principal razón por la cual aplicar prueba de Usabilidad es la obtención de un sistema que hace al usuario más productivo, aumentando su eficiencia y satisfacción al utilizarlo.

La usabilidad es un tema crítico para la aceptación de un sistema, si el sistema no es percibido como una herramienta que ayuda al usuario a realizar sus tareas, se dificulta la aceptación del mismo. Si las tareas del usuario no son respaldadas convenientemente por el sistema, entonces no se está respondiendo adecuadamente a las necesidades del usuario, y el equipo de desarrollo se está alejando del objetivo principal de la construcción de un sistema software.

Técnicas de prueba de Usabilidad

Las principales técnicas que ayudan a conseguir un mejor nivel de usabilidad en el software desarrollado son:

- Especificaciones: Análisis de usuarios, análisis de tareas y especificaciones de usabilidad.
- Diseño: Diseño de la interacción, prototipado y participación de usuarios.
- Evaluación: Test de usabilidad y evaluación heurística.

Evaluación

La evaluación de usabilidad permite conocer el nivel de usabilidad que alcanza el prototipo actual del sistema, e identificar los fallos de usabilidad existentes.

La evaluación se realiza usualmente mediante test de usabilidad, complementados con evaluaciones heurísticas.

Test de Usabilidad

Los test de usabilidad son la práctica de usabilidad más extendida. Consisten en presentar al usuario una serie de tareas a realizar, y pedirle que las realice con el prototipo del sistema. Las acciones y comentarios de usuario se recopilan para un análisis posterior. Para conseguir unos test de usabilidad con resultados fiables, las condiciones del test y del lugar donde éste se realiza deben ser lo más parecidas posibles al entorno de uso previsto para el sistema.

Motivación: Se basa en que no es posible conocer el nivel de usabilidad de un sistema si no se prueba con usuarios reales.

Procedimiento: En primer lugar, es preciso decidir con qué distintos grupos de usuarios se va a probar el sistema, y cuántos participantes de cada grupo se van a obtener para realizar los test. A continuación, se deben diseñar las tareas de test cuya realización se va a pedir a los participantes; normalmente se sacan del resultado del análisis de tareas, intentado enmarcarlas en un contexto de uso real.

Hay que decidir también otros detalles, como la posibilidad de pedir ayuda al evaluador, qué tipo de información se dará a los participantes acerca del sistema antes de comenzar con el test en sí, o si se dará la posibilidad al participante de acceder libremente al sistema para obtener opiniones acerca de su impresión global.

Otra variante consiste en realizar cada test con dos usuarios para observar los comentarios que intercambian. Es habitual pedir al participante que “piense en voz alta” mientras intenta llevar a cabo las tareas; este procedimiento permite identificar partes del sistema con un nivel pobre de usabilidad.

Cuando el test está preparado y los participantes han sido reunidos, los test se llevan a cabo, opcionalmente grabados en audio o vídeo. Otra posibilidad es registrar las acciones de los usuarios en un fichero del sistema para un análisis posterior. Una vez

se han realizado todos los test, los datos recogidos son analizados y los resultados se aplican en el siguiente ciclo de diseño. **(Grau, 2000)**

Se selecciona dentro de las pruebas a aplicar pruebas de Usabilidad por ser uno de los temas que cobra más importancia dentro del desarrollo del software. Si el usuario no mantiene una estrecha relación con la interfaz del software de la manera más amigable posible no se obtendrá un buen rendimiento por su parte.

En estos tiempos en los que el software va dirigido a un público cada vez más amplio y menos expertos, es factible aplicar esta prueba, aunque el Sistema de Gestión de Audio, Video y Streaming para VTV desarrolle su aplicación para usuarios muy específicos y que tienen conocimientos básicos para trabajar en la aplicación, la misma debe tener un entorno agradable a la vista del usuario, para que se sienta a gusto en su ambiente de trabajo y de esta manera produzca a mayor escala y con más eficiencia.

De las técnicas que se encuentran dentro de la prueba de Usabilidad se escoge Evaluación y dentro de la misma Test de Usabilidad por ser la más lograda en comparación con las demás. El resto de las técnicas, Especificaciones y Diseños no se deben tener en cuenta por ser parte del levantamiento de requisitos, es decir, son técnicas que se ven en el proceso de desarrollo del software sin embargo en la que se selecciona se trabaja directamente con los usuarios y se pueden ver los objetivos que quieras lograr de la manera que los quieras lograr con solo realizar un buen test.

No se tiene en cuenta la Evaluación heurística dentro de esta técnica porque para ello se necesita de un experto en el tema para que realice las pruebas y se hace más difícil la aplicación ya que requiere de tiempo

Luego de precisar que pruebas serán aplicadas se debe conocer el procedimiento para cada una de ellas.

Procedimientos para las pruebas seleccionadas:

Pruebas de Caja Blanca. Técnica de Camino Básico

- Identificar dentro del código a revisar los nodos.
- Dibujar el grafo de flujo.
- Calcular la complejidad ciclomática.
- Determinar el conjunto de caminos independientes.
- Preparar el caso de prueba para cada camino.

Pruebas de Caja Negra. Técnica de Partición Equivalente

En el diseño de casos de prueba para Partición Equivalente se procede en dos pasos:

- *Se identifican las clases de equivalencia.* Las clases de equivalencia son identificadas tomando cada condición de entrada (generalmente una oración o una frase en la especificación) y repartiéndola en dos o más grupos.

Es de notar que dos tipos de clases de equivalencia están identificados: las clases de equivalencia válidas representan entradas válidas al programa, y las clases de equivalencia inválidas que representan el resto de los estados posibles de la condición (es decir, valores erróneos de la entrada).

- *Se define los casos de prueba.* El segundo paso es el uso de las clases de equivalencia para identificar los casos de prueba.

El proceso es como sigue:

  - Se asigna un número único a cada clase de equivalencia. Hasta que todas las clases de equivalencia válidas hayan sido cubiertas por los casos de prueba.
  - Se escribe un nuevo caso de prueba que cubra la clase de equivalencia válida.
  - Por último, hasta que los casos de prueba hayan cubierto todas las clases de equivalencia inválidas, se escribe un caso de la prueba que cubra una, y solamente una, de las clases de equivalencia inválidas descubiertas **(Myers, 2004)**.

Pruebas de Integración. Incremental – Ascendente

- Se combinan los módulos de bajo nivel en grupos que realicen una subfunción específica.
- Se escribe un controlador (un programa de control de la prueba) para coordinar la entrada y salida de los casos de prueba.
- Se prueba el grupo.

Pruebas de Sistema

Prueba de Stress

- Determinar el número de usuarios que se desea intervengan a la vez en un/os camino/s crítico/s.
- Simular con la herramienta conveniente las incorporaciones de usuarios que ejecutan el mismo camino, todos a la vez.

- Incorporar a la herramienta utilizada más usuarios que los necesarios hasta conseguir que el sistema colapse.

Pruebas de Usabilidad

- Decidir con qué distintos grupos de usuarios se va a probar el sistema.
- Se deben diseñar las tareas de test cuya realización se va a pedir a los participantes.
- Pedir al participante que “piense en voz alta” mientras intenta llevar a cabo las tareas.
- Registrar las acciones de los usuarios en un fichero del sistema para un análisis posterior.
- Una vez realizados todos los test, los datos recogidos son analizados y los resultados se aplican en el siguiente ciclo de diseño.

2.3.6 Cronograma de pruebas

No	Tarea	Fecha	Responsable	Participantes
1	Pruebas de Caja Blanca	2-9 de marzo	Darmalina Valenciano Cuza	Darmalina Valenciano Cuza
2	Pruebas de Caja Negra	30 de marzo-6 de abril	Darmalina Valenciano Cuza	Darmalina Valenciano Cuza
3	Pruebas de Caja Blanca	30 de mayo-6 de junio	Darmalina Valenciano Cuza	Darmalina Valenciano Cuza
4	Pruebas de Caja Negra	30 de mayo-6 de junio	Darmalina Valenciano Cuza	Darmalina Valenciano Cuza
5	Pruebas de Integración	pendiente		

6	Pruebas de Stress	pendiente		
7	Pruebas de Usabilidad	pendiente		
8	Pruebas de Seguridad	propuesta		

2.4 Conclusiones Parciales.

La elaboración de la planificación de las pruebas contando con una estrategia como guía del proceso, permitió la selección de las pruebas a aplicar: Unidad, Integración y Sistema, haciendo uso intensivo de las técnicas de Caja Blanca y Caja Negra. Para ello se tuvo en cuenta las características del producto Sistema de Gestión de Audio, Video y Streaming, tras la entrevista realizada al Analista y Jefe del proyecto Captura y Catalogación de Medias, en este caso, una aplicación web que se encuentra en la fase de Construcción.

Capítulo 3: Resultados obtenidos.

3.1 Introducción

En el siguiente capítulo, se expondrán los Diseños de Casos de Pruebas realizados a los distintos módulos del proyecto Captura y Catalogación de Medias, para el producto Sistema de Gestión de Audio, Video y Streaming para VTV. Luego se aplica el proceso de pruebas y se registran los resultados obtenidos en el mismo.

3.2 Diseño de los casos de pruebas (DCP)

3.2.1 DCP Caja Blanca. Camino Básico.

Nombre del Caso de Uso: Buscar Material a Catalogar

```
public function buscarMaterialesACatalogar($procedencia, $fechaInicio, $fechaFin)
{
    $medias = null; 1
    if ($procedencia != "") { 2
        if ($fechaInicio == null && $fechaFin == null) 3
            $medias = Doctrine::getTable("Media")->findBy("procedencia", $procedencia)->toArray();4
        else if ($fechaInicio != null && $fechaFin == null)5
            $medias = $this->getMediasMayoresFechaDadaConProcedencia($fechaInicio, $procedencia);6
        else if ($fechaInicio == null && $fechaFin != null)7
            $medias = $this->getMediasMenoresFechaDadaConProcedencia($fechaFin, $procedencia);8
        else
            $medias = $this->getMediasEntreIntervaloFechasConProcedencia($fechaInicio, $fechaFin, $procedencia);9
    }
    else {
        if ($fechaInicio == null && $fechaFin == null) 10
            $medias = Doctrine::getTable("Media")->findAll()->toArray(); 11
        else if ($fechaInicio != null && $fechaFin == null) 12
            $medias = $this->getMediasMayoresFechaDada($fechaInicio); 13
```

```
        else if ($fechaInicio == null && $fechaFin != null) 14
            $medias = $this->getMediasMenoresFechaDada($fechaFin); 15
        else
            $medias = $this->getMediasEntreIntervaloFechas($fechaInicio,
$fechaFin); 16
    }
    return $medias; 17
} 17
```

Grafo de flujo de datos que se genera:

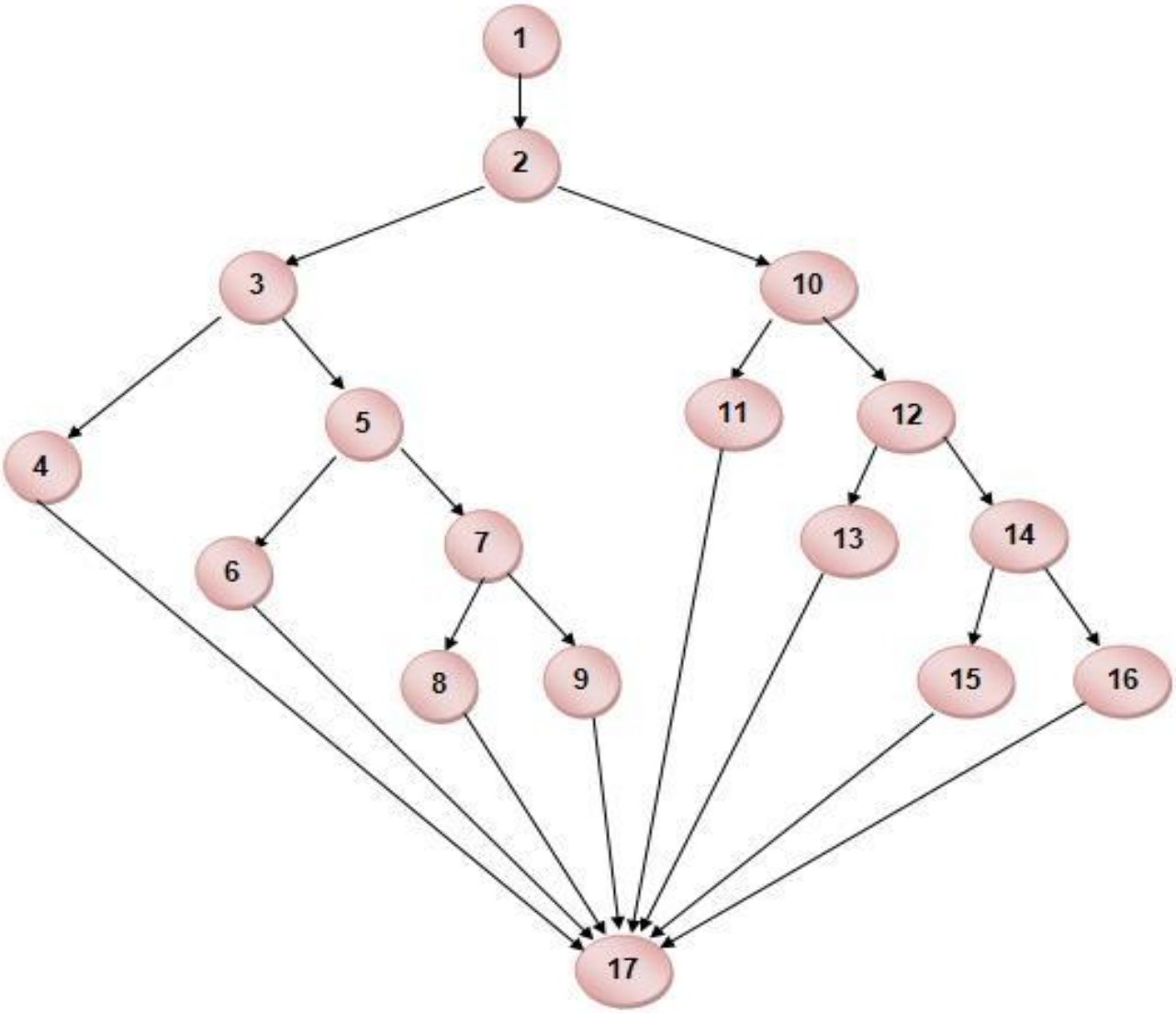

Figura 9: Grafo de Flujo

Cálculo de complejidad ciclomática del grafo:

V (G) = Número de Aristas - Número de Nodos + 2
V (G) = 23-17+2
V (G) = 8

Caminos independientes:

Camino 1: 1-2-3-4-17
Camino 2: 1-2-3-5-6-17
Camino 3: 1-2-3-5-7-8-17
Camino 4: 1-2-3-5-7-9-17
Camino 5: 1-2-10-11-17
Camino 6: 1-2-10-12-13-17
Camino 7: 1-2-10-12-14-15-17
Camino 8: 1-2-10-12-14-16-17

CP para Camino 1: 1-2-3-4-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:
$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)
$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.
$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:
Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en este primer camino, si el valor de $procedencia es distinto de vacío y los valores de $fechaInicio y $fechaFin son vacíos.

CP para Camino 2: 1-2-3-5-6-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:

$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)

$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.

$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el segundo camino, si los valores de $procedencia y de $fechaInicio son distintos de vacío y el valor de $fechaFin es vacío. Devuelve un valor mayor que la fecha y la procedencia dadas.

CP para Camino 3: 1-2-3-5-7-8-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:

$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)

$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.

$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el tercer camino, si el valor de $procedencia es distinto de vacío y los valores de $fechaInicio y $fechaFin distintos de vacío, distintivamente. Devuelve un valor menor que la fecha o las fechas, y procedencia dadas.

CP para Camino 4: 1-2-3-5-7-9-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:

$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)

$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.

$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el cuarto camino, si el valor de $procedencia es distinto de vacío y los valores de $fechaInicio y $fechaFin son distintos de vacío, distintivamente. Devuelve un valor entre las fechas dadas, y la procedencia.

CP para Camino 5: 1-2-10-11-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:

$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)
$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.
$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el quinto camino, si los valores de $fechaInicio y $fechaFin son vacíos.

CP para Camino 6: 1-2-10-12-13-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:

$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)
$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.
$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el sexto camino, si los valores de $fechaInicio y $fechaFin son vacíos o $fechaInicio es distinto de vacío. Devuelve un valor mayor que la fecha de inicio.

CP para Camino 7: 1-2-10-12-14-15-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:
$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)
$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.
$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el séptimo camino, si los valores de $fechaInicio y $fechaFin son vacíos o $fechaFin es distinto de vacío. Devuelve un valor menor que la fecha de fin.

CP para Camino 8: 1-2-10-12-14-16-17

Caso de prueba: Probando la función buscarMaterialesACatalogar

Entradas:
$procedencia= Filtro de búsqueda. Muestra texto en un Combobox. (Opcional)
$fechaInicio= Campo para seleccionar una fecha con el formato dd-mm-yy.
$fechaFin= Campo para seleccionar una fecha con el formato dd-mm-yy.

Resultado esperado:

Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el octavo camino, si los valores de $fechaInicio y $fechaFin son distintos de vacío. Devuelve un valor entre las fechas dadas.

3.2.2 DCP Caja Negra. Partición Equivalente.

Nombre del Caso de Uso: Autenticar Usuario

Descripción General: El caso de uso se inicia cuando se introduce el usuario y la contraseña para acceder en la aplicación, estos son verificados; el caso de uso finaliza

mostrando la interfaz correspondiente según los permisos del usuario que se autentica.

Condiciones de Ejecución: El usuario debe existir en la Base de Datos y en el servidor LDAP y estar habilitado en el sistema.

Secciones a Probar en el Caso de Uso:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Autenticar usuario	**EC 1.1:** Autenticar usuario satisfactoriamente	El actor escribe su nombre de usuario y contraseña y presiona el botón “Aceptar” para entrar en el sistema. El sistema verifica que exista conexión con el servidor LDAP, luego verifica que el usuario exista en el servidor LDAP y la validez de la contraseña introducida, posterior a esto sistema verifica que exista conexión con la base de datos y verifica que el usuario exista en la Base de Datos. Verifica los permisos que tiene asignados y en dependencia de los permisos le permite acceder a la aplicación mostrando la interfaz correspondiente.

	EC 1.2: Autenticar usuario fallido.	Si no se puede establecer conexión con el servidor LDAP, el sistema muestra el siguiente mensaje: “No se puede establecer conexión con el servidor LDAP”. Si no se puede establecer conexión con la Base de Datos, el sistema muestra el siguiente mensaje: “No se puede establecer conexión con los datos”. Si los datos de autenticación introducidos por el actor no son válidos, el sistema muestra el siguiente mensaje: “Usuario o contraseña incorrecta”. El sistema da la posibilidad de volverse a introducir los datos.

Tabla 3: Diseño de Caso de prueba del CU: Autenticar usuario

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	usuario	Campo de texto	No	El usuario introducirá en este campo su usuario para autenticarse.
2	contraseña	Campo de texto	No	El usuario introducirá en este campo su contraseña para autenticarse.

Tabla 4: Descripción de las variables del CU: Autenticar usuario

ID del escenario	Escenario	usuario	contraseña	Respuesta del Sistema	Resultado de la Prueba
EC 1.1	Autenticar usuario satisfactoriamente.	V	V	El sistema verifica que exista conexión con el servidor LDAP, luego verifica que el usuario exista en el servidor LDAP y la validez de la contraseña introducida, posterior a esto sistema verifica que exista conexión con la base de datos y verifica que el usuario exista en la Base de Datos. Verifica los permisos que tiene asignados y en dependencia de los permisos le permite acceder a la aplicación mostrando la interfaz correspondiente.	

ID del escenario	Escenario	usuario	contraseña	Respuesta del Sistema	Resultado de la Prueba
EC 1.2	Autenticar usuario fallido.	I	V	Si no se puede establecer conexión con el servidor LDAP, el sistema muestra el siguiente mensaje: “No se puede establecer conexión con el servidor LDAP”. Si no se puede establecer conexión con la Base de Datos, el sistema muestra el siguiente mensaje: “No se puede establecer conexión con los datos”. Si los datos de autenticación introducidos por el actor no son válidos, el sistema muestra el siguiente mensaje: “Usuario o contraseña incorrecta”. El sistema da la posibilidad de volverse a introducir los datos.	

ID del escenario	Escenario	usuario	contraseña	Respuesta del Sistema	Resultado de la Prueba
EC 1.2	Autenticar usuario fallido.	V	I	Si no se puede establecer conexión con el servidor LDAP, el sistema muestra el siguiente mensaje: “No se puede establecer conexión con el servidor LDAP”. Si no se puede establecer conexión con la Base de Datos, el sistema muestra el siguiente mensaje: “No se puede establecer conexión con los datos”. Si los datos de autenticación introducidos por el actor no son válidos, el sistema muestra el siguiente mensaje: “Usuario o contraseña incorrecta”. El sistema da la posibilidad de volverse a introducir los datos.	

Tabla 5: Matriz de datos del CU: Autenticar usuario

3.3 Resultados de las pruebas aplicadas

3.3.1 Caja Blanca. Camino Básico

Caminos	Caso de Prueba	Resultado esperado	Resultado de la prueba
1	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en este primer camino, si el valor de $procedencia es distinto de vacío y los valores de $fechaInicio y $fechaFin son vacíos.	Satisfactorio
2	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el segundo camino, si los valores de $procedencia y de $fechaInicio son distintos de vacío y el valor de $fechaFin es vacío. Devuelve un valor mayor que la fecha y la procedencia dadas.	Satisfactorio
3	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el tercer camino, si el valor de $procedencia es distinto de vacío y los valores de $fechaInicio y $fechaFin distintos de vacío, distintivamente. Devuelve un valor menor que la fecha o las fechas, y procedencia dadas.	Satisfactorio

4	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el cuarto camino, si el valor de $procedencia es distinto de vacío y los valores de $fechaInicio y $fechaFin son distintos de vacío, distintivamente. Devuelve un valor entre las fechas dadas, y la procedencia.	Satisfactorio
5	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el quinto camino, si los valores de $fechaInicio y $fechaFin son vacíos.	Satisfactorio
6	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el sexto camino, si los valores de $fechaInicio y $fechaFin son vacíos o $fechaInicio es distinto de vacío. Devuelve un valor mayor que la fecha de inicio.	Satisfactorio
7	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el séptimo camino, si los valores de $fechaInicio y $fechaFin son vacíos o $fechaFin es distinto de vacío. Devuelve un valor menor que la fecha de fin.	Satisfactorio

8	Probando la función buscarMaterialesACatalogar	Se obtienen los materiales no catalogados en dependencia de los filtros que se especifiquen y del valor de ellos, en el octavo camino, si los valores de $fechaInicio y $fechaFin son distintos de vacío. Devuelve un valor entre las fechas dadas.	Satisfactorio

Tabla 6: Resultado de Prueba de Caja Blanca

Al realizarse las pruebas de Caja Blanca al producto Sistema de Gestión de Audio, Video y Streaming para VTV, se perciben códigos satisfactorios en los casos de usos críticos ya implementados, como resultado de dicha prueba.

3.3.2 Caja Negra. Partición Equivalente.

Sección	Resultado esperado	Resultado de la prueba
Autenticar usuario	Que se muestre la interfaz correspondiente según los permisos que posee el usuario.	Autentica al usuario correctamente y muestra la interfaz correspondiente sus permisos.

Tabla 7: Resultado de Prueba de Caja Negra

Por el proyecto Captura y Catalogación de Medias encontrarse en plena fase de Construcción, solo se le pudo aplicar prueba de Caja Negra con la técnica de Partición Equivalente a cuatro casos de uso críticos, donde se encontraron dificultades en sus dos iteraciones. En la primera iteración se encontraron ocho no conformidades significativas, de las cuales tres pasaron de estado pendiente a resuelta. En la segunda iteración se encontró otra no conformidad para un total de nueve no conformidades significativas de las cuales aún quedan pendientes seis. Los principales errores encontrados son funcionales, de validación y documentación.

Luego de realizarse dicha prueba es que se puede llevar a cabo la prueba de Integración que momentáneamente no se puede aplicar por no tener ningún Módulo completamente implementado, aun así lo que se ha podido probar ha tenido

resultados tanto satisfactorios como no satisfactorios, registrados en un documento de No Conformidades (ver Anexos). Cuando se integre se podrán aplicar las pruebas de Usabilidad y de Stress. Valorando la posibilidad de que se apliquen dichas pruebas se podría tener en cuenta la prueba de Seguridad.

3.4 Conclusiones parciales

Se logró establecer un enlace entre los diseños de casos de prueba y la aplicación de las pruebas propuestas. De los 14 diseños de pruebas de Caja Negra (ver Anexos), se emplearon solo cuatro para aplicar, por encontrarse el proyecto en plena fase de Construcción y no tener las interfaces que se corresponden con los diseños para esos casos de uso. Los resultados obtenidos en tres de esos diseños fueron insatisfactorios, siendo solo para uno de ellos satisfactorio y de los nueve diseños de Caja Blanca (ver Anexos) realizados, los resultados fueron satisfactorios en su totalidad. Todos los resultados fueron registrados en un Documento de No Conformidades.

Conclusiones generales

Con el fin de permitir la verificación y validación del producto Sistema de Gestión de Audio, Video y Streaming para VTV en el presente trabajo de diploma se concluyó:

- Las pruebas seleccionadas para detectar errores en el software evaluado fueron las pruebas de Unidad, Integración y Sistema por las características vigentes del proyecto.
- Se diseñaron casos de pruebas que permitieron que se aplicaran pruebas mediante las técnicas de Caja Blanca y Caja Negra, verificando su correcta implementación, sus funcionalidades y los requisitos exigidos por el cliente.
- La comparación entre los resultados esperados y los obtenidos, arrojó resultados enteramente satisfactorios para la técnica de Caja Blanca e insatisfactorios en su mayoría para la técnica de Caja Negra. Tres de las nueve no conformidades encontradas en esta técnica fueron resueltas y las seis restantes se encuentran en estado pendiente.
- Tras la aplicación de las técnicas de Caja Blanca y Caja Negra fueron documentados los errores encontrados, para darle seguimiento al perfeccionamiento del producto por parte del equipo de pruebas y desarrollo.

Recomendaciones

Por el proyecto Captura y Catalogación de Medias encontrarse en la fase de Construcción, no se logró aplicar todas las pruebas necesarias para un correcto funcionamiento del producto Sistema de Audio, Video y Streaming para VTV. Una vez aplicadas las pruebas de Caja Negra mediante la técnica de Partición Equivalente y las pruebas de Caja Blanca mediante la técnica de Camino Básico y además encontrarse implementados los módulos del proyecto, se recomienda:

- Realizar las pruebas ya aplicadas a otros casos de usos del proyecto.

- Realizar pruebas de Integración, de Usabilidad, Stress y Seguridad, rigiéndose por la estrategia de pruebas definida en este trabajo de diploma.

Referencias bibliográficas

Bravo, Dr. Jose Vicente Alvarez. 2007. ***Tema 7: Validación. Valladolid : s.n., 2007.***

Cárdenas, Maria Clara Choucair. 2009. ***Pruebas de software. Documentos de investigación. [Online] 2009. http://www.acis.org.co/index.php?id=972.***

2008. ***Definición de prueba. [Online] 2008. http://definicion.de/prueba/.***

Febles, Ailin. 2006. ***Calidad del Software. Ciudad de la Habana : s.n., 2006.***

Ferrer, Gregorio Robles y Jorge. 2002. ***Programación Extrema y Software Libre. [Online] 2002. http://es.tldp.org/Presentaciones/200211hispalinux/gregorio2/progm-ext-soft-libre-html/..***

Granja, Juan Carlos. 2007. ***Métricas, técnicas del software. Ciudad de la Habana : s.n., 2007.***

Grau, Xavier Ferré. 2000. ***Principios Básicos de Usabilidad para Ingenieros Software. Valladolid, España. : s.n., 2000.***

Hernandez, Gérald Lomprey y Saulo. 2008. ***LA IMPORTANCIA DE LA CALIDAD EN EL DESARROLLO DE SOFTWARE. México : s.n., 2008.***

Infomedia. 2009. ***Proyecto de ISIII. Pruebas Extra. Pruebas de Caja Blanca. 2009.***

Jacobson, James Rumbaugh Grady Booch Ivar. 2000. ***El Proceso Unificado de Desarrollo de Software. Madrid : s.n., 2000.***

Juristo, Ana M. Moreno Sira Vegas Natalia. 2005. ***TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 2005.***

—. 2006. ***Técnicas de evaluación del software. 2006.***

Lovelle, Juan Manuel Cueva. 1999. ***Calidad del Software. España : Pampa, 1999.***

Mañas, J.A. 1994. ***Pruebas de Programas. 1994.***

Myers, Glenford J. 2004. ***The art of sotware Testing. Wiley : Segunda Edición, 2004.***

Piattini, Jose A. Calvo-Manzano Joaquin Cervera Bravo Luis Fernandez Sanz Mario G. 2004. ***Análisis y diseño de aplicaciones informáticas de gestión, una perspectiva de ingeniería del software. 2004.***

Pressman, Roger. 2002. ***Ingeniería de Software. Un enfoque práctico. España : Quinta Edición, 2002.***

—. 2005. ***Ingeniería de Software. Un enfoque práctico. s.l. : Sexta Edición, 2005.***

—. 2002. ***Ingeniería del Software, un enfoque práctico. 2002.***

Quesada, Juan Antonio López. 2006-2007. ***Fundamentos de Ingeniería de Software. Murcia : s.n., 2006-2007.***

Risoto, Manuel Mejías. 2007. ***Departamento de Lenguajes y Sistemas Informáticos. [Online] 2007. www.lsi.us.es/docencia/get.php?id=361.***

Rodríguez, Jorge. 2006. ***Pruebas Unitarias. 2006.***

Silva, Andy Pérez. 2010. ***Metodologías de desarrollo de software. Su aplicación y uso. 2010.***

Tello, Dr. Eduardo A. Rodríguez. 2011. ***Importancia de las pruebas de software. Tamaulipas : s.n., 2011.***

Torres, Javier J. Gutiérrez María J. Escalona Manuel Mejías y Jesús. 2006. ***MODELOS Y ALGORITMOS PARA LA GENERACIÓN DE OBEJTIVOS DE PRUEBAS. Barcelona : s.n., 2006.***

Tuya, Javier. 2007. ***Pruebas de software. 2007.***

Usaola, Dr. Macario Polo. 2005. ***Mantenimiento Avanzado de Sistemas de Información. Pruebas del Software. Ciudad Real : s.n., 2005.***

Vallecillo, Manuel F. Bertoa Jose M. Troya Antonio. 2004. ***Aspectos de Calidad en el Desarrollo de Software basado en Componentes. 2004.***

Bibliografía

Bravo, Dr. Jose Vicente Alvarez. 2007. ***Tema 7: Validación. Valladolid : s.n., 2007.***

Cárdenas, Maria Clara Choucair. 2009. ***Pruebas de software. Documentos de investigación. [Online] 2009. http://www.acis.org.co/index.php?id=972.***

2008. ***Definición de prueba. [Online] 2008. http://definicion.de/prueba/.***

Domínguez, Juana Bustamante. 2011. ***Tipos de pruebas. 2011.***

Febles, Ailin. 2006. ***Calidad del Software. Ciudad de la Habana : s.n., 2006.***

Ferrer, Gregorio Robles y Jorge. 2002. ***Programación Extrema y Software Libre. [Online] 2002. http://es.tldp.org/Presentaciones/200211hispalinux/gregorio2/progm-ext-soft-libre-html/..***

Goñi, J.R. Zubizarreta J.Iturrioz A. 2006. ***Ingeniería de Software. País Vasco : s.n., 2006.***

Granja, Juan Carlos. 2007. ***Métricas, técnicas del software. Ciudad de la Habana : s.n., 2007.***

Grau, Xavier Ferré. 2000. ***Principios Básicos de Usabilidad para Ingenieros Software. Valladolid, España. : s.n., 2000.***

Hernandez, Gérald Lomprey y Saulo. 2008. ***LA IMPORTANCIA DE LA CALIDAD EN EL DESARROLLO DE SOFTWARE. México : s.n., 2008.***

Infomedia. 2009. ***Proyecto de ISIII. Pruebas Extra. Pruebas de Caja Blanca. 2009.***

Jacobson, James Rumbaugh Grady Booch Ivar. 2000. ***El Proceso Unificado de Desarrollo de Software. Madrid : s.n., 2000.***

Juristo, Ana M. Moreno Sira Vegas Natalia. 2005. ***TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 2005.***

—. 2006. ***Técnicas de evaluación del software. 2006.***

Lovelle, Juan Manuel Cueva. 1999. ***Calidad del Software. España : Pampa, 1999.***

Mañas, J.A. 1994. ***Pruebas de Programas. 1994.***

Myers, Glenford J. 2004. ***The art of sotware Testing. Wiley : Segunda Edición, 2004.***

Piattini, Jose A. Calvo-Manzano Joaquin Cervera Bravo Luis Fernandez Sanz Mario G. 2004. ***Análisis y diseño de aplicaciones informáticas de gestión, una perspectiva de ingeniería del software. 2004.***

Pressman, Roger. 2002. ***Ingeniería de Software. Un enfoque práctico. España : Quinta Edición, 2002.***

—. 2005. ***Ingeniería de Software. Un enfoque práctico. s.l. : Sexta Edición, 2005.***

—. 2002. ***Ingeniería del Software, un enfoque práctico. 2002.***

Quesada, Juan Antonio López. 2006-2007. ***Fundamentos de Ingeniería de Software. Murcia : s.n., 2006-2007.***

Risoto, Manuel Mejías. 2007. ***Departamento de Lenguajes y Sistemas Informáticos. [Online] 2007. www.lsi.us.es/docencia/get.php?id=361.***

Rodríguez, Jorge. 2006. ***Pruebas Unitarias. 2006.***

Silva, Andy Pérez. 2010. ***Metodologías de desarrollo de software. Su aplicación y uso. 2010.***

Tello, Dr. Eduardo A. Rodríguez. 2011. ***Importancia de las pruebas de software. Tamaulipas : s.n., 2011.***

Torres, Javier J. Gutiérrez María J. Escalona Manuel Mejías y Jesús. 2006. ***MODELOS Y ALGORITMOS PARA LA GENERACIÓN DE OBEJTIVOS DE PRUEBAS. Barcelona : s.n., 2006.***

Tuya, Javier. 2007. ***Pruebas de software. 2007.***

Usaola, Dr. Macario Polo. 2005. ***Mantenimiento Avanzado de Sistemas de Información. Pruebas del Software. Ciudad Real : s.n., 2005.***

Vallecillo, Manuel F. Bertoa Jose M. Troya Antonio. 2004. ***Aspectos de Calidad en el Desarrollo de Software basado en Componentes. 2004.***