Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: “Estrategia de pruebas para el producto Visor de imágenes 2D del proyecto Vismedic de la facultad 5”

Autora: Yaimelys Baeza Orta.

Tutora: Ing. Adriana Santos Lebeque.

Asesor: Msc. Manuel Villanueva Betancourt.

La Habana, Julio 2012

“Año 53 de la Revolución”

Declaración de autoría

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

Para que así conste firmamos la presente el día 03 de Julio del año 2012.

Firma de la Autora
Yaimelys Baeza Orta

Firma de la Tutora
Ing. Adriana Santos Lebeque

Dedicatoria

A mi mamá por ser la mejor madre del mundo, por apoyarme siempre en tratar de lograr todas mis metas y estar siempre a mi lado en los momentos difíciles.

A Guille: mi novio, mi compañero, mi amigo, mi confidente, por su amor infinito, su perseverancia y por confiar siempre en mí.

A mi papá por estar siempre a mi lado y apoyarme en todo.

A mis hermanos por tener confianza en mí y ser tan especiales en mi vida.

A mis sobrinos, mis niños, que fueron la luz de mi inspiración y ojalá pueda ser ejemplo para ellos.

Yaimi

Agradecimientos

Quiero agradecer con todo mi corazón:

A la Revolución, a Fidel, por brindarme la posibilidad de estudiar en la Universidad de las Ciencias Informáticas y permitir realizar mis sueños.

A mis padres por su apoyo incondicional, en especial a mi mama por estar siempre a mi lado en los momentos alegres y en los tristes, por su amor infinito y por enseñarme que en el camino puedes estar al pie de la montaña, solo que debes tener confianza en ti mismo y así lograrás llegar a la cima.

A Guille por tener tanta paciencia conmigo y aguantar mis malcriadeces, por siempre darme mucho de su amor incondicional, por entenderme en los momentos difíciles y creer siempre en mí, por enseñarme que no existe nada imposible.

A mi hermana Yanelys y mi hermano William por confiar siempre en mí y ayudarme a vencer mis metas.

A mis sobrinos Miguelito, Danita, Willito y Alecito, por siempre demostrarme que a pesar de la distancia nuestro amor y cariño crece con el paso del tiempo y espero que un día puedan verme como ejemplo.

A mi tío Kike y mi tía Xiomara les agradezco enormemente por estar siempre ahí cuando los necesitamos, por ser nuestra mano derecha y demostrarnos su amor incondicional.

A mis suegros Aidita y Guillermo por siempre demostrarme su amor y tratarme como una de sus niños también, por malcriarme en mis gustos y por siempre apoyarme en todo.

A mi cuñada Suly por ser como la hermana pequeña, por demostrarme su confianza y por demostrarme su cariño y apoyo.

A mis abuelos mima y papi por siempre demostrarme su amor y cariño.

A Migue y a Mallelyn por apoyarme siempre.

A Adriana por todos sus consejos.

A Maribel por ayudarme enormemente y brindarme su amistad.

A mis amigas Yanet, Liset, Yainurys y amigo Batista, que siempre estuvieron cerca brindándome su apoyo incondicional y confianza, por los momentos buenos que pasamos que nunca voy a olvidar.

A mis amigos Yusi, Yainier, Carlos Arce, Erieilys, por siempre darme su mano en todo momento.

A todas esas personas que en su momento marcaron espacios en mi vida y que siempre le agradeceré por su total apoyo y confianza.

Yaimi

Resumen

En la actualidad, el campo de la informática se ha impuesto en el desarrollo de la sociedad, buscando cada día la mejora y calidad con que se produce el software. Uno de los temas más relevantes actualmente es la Realidad Virtual (RV); esta ciencia ha encontrado desarrollos ya consolidados en la medicina y la defensa, además de relevantes oportunidades en el entretenimiento y la promoción empresarial. Cuba se ha sumado al desarrollo de software utilizando la RV, ejemplo de ello es la creación de un Visor de Imágenes Médicas implementado por la Universidad de las Ciencias Informáticas (UCI) en colaboración con el Instituto Nacional de Oncología y Radiología de La Habana.

En el presente trabajo se define una estrategia de prueba para garantizar la calidad del producto para cumplir con este objetivo se realizó un estudio del estado del arte de las pruebas de software aplicadas en el proceso de desarrollo de software en el mundo y en la UCI, además se investigó sobre las estrategias de pruebas, los visores de imágenes médicas y se caracterizó el producto Visor de Imágenes 2D. Se establecieron los tipos de pruebas que se van a aplicar, así como los roles, actividades y artefactos a evaluar, se valida la estrategia propuesta y se analizan los resultados obtenidos en la ejecución de las pruebas realizadas.

PALABRAS CLAVES: Estrategia de Pruebas, Prueba, Visor de Imágenes.

Índice

Índice de Tablas

Índice de Figuras

Introducción

El avance de la informática ha propiciado en los últimos años el surgimiento de un nuevo término: “Realidad Virtual (RV)”. Este concepto encierra amplias aplicaciones que van encaminadas a simular objetos y procesos del mundo real. Los Sistemas de Realidad Virtual (SRV), se han expandido considerablemente a diferentes sectores de la sociedad. Esta tecnología puede encontrarse en aplicaciones en la defensa, la medicina, la educación y el entretenimiento.

En el caso de la medicina, desde el surgimiento en la década de los 80 de los Rayos X, las imágenes médicas han adquirido vital importancia en el campo de la medicina pues brindan la posibilidad a los médicos de realizar un diagnóstico no invasivo de patologías presentes en el organismo humano. Estos especialistas disponen de imágenes médicas de diversas modalidades, tales como: Tomografías Computarizadas (CT) y Resonancias Magnéticas (MRI). Como complemento, las herramientas informáticas y los métodos de tratamiento, análisis y visualización de imágenes digitales, han resultado de gran utilidad para el desarrollo de este tipo de aplicaciones médicas. Por otro lado a los usuarios de estas aplicaciones informáticas como radiólogos y cirujanos les resulta muy conveniente que el software para el diagnóstico médico brinde la posibilidad de realizarlo desde una vista bidimensional y tridimensional.

Nuestro país, pese a los problemas económicos que enfrenta debido a las limitaciones que nos han sido impuestas, ha decidido insertarse en la industria del software, uno de sus principales exponentes es la Universidad de las Ciencias Informáticas (UCI), la cual surge al calor de la batalla de ideas y cuya misión es, producir software y servicios informáticos a partir de la vinculación estudio – trabajo como modelo de formación.

En Cuba se ha logrado un avance en la realidad virtual. En la UCI se han desarrollado aplicaciones utilizando sistemas virtuales, ejemplo de ello: Simulador de carro, Simulador de tiro, Simulador quirúrgico, elaboración de juegos virtuales educativos, etc.

La necesidad de la calidad en los productos se ve presente también en los sistemas virtuales. El desarrollo de la industria de software implica que los productos realizados deben ser confiables, precisos, rápidos de utilizar, flexibles y además deben de estar bien documentados.

Introducción

El Centro de Desarrollo de Informática Industrial (CEDIN), es uno de los centros de producción de la UCI, este tiene como misión fundamental el desarrollo de software con una mayor calidad. El CEDIN en coordinación con el "Instituto Nacional de Oncología y Radiología de la Habana" han desarrollado el proyecto Vismedic (Sistema Multiplataforma de Visualización Médica Bidimensional y Tridimensional), donde el producto que ofrece es un Visor de Imágenes 2D para el tratamiento de imágenes médicas el cual es de gran ayuda para los especialistas.

El proyecto Vismedic tiene como su principal objetivo entregar en tiempo el producto, con todas las funcionalidades pactadas con el cliente y con la calidad requerida. El CEDIN cuenta con un proceso de prueba definido guiado por RUP, pero este no se ajusta a las características del producto Visor de imágenes 2D, para ello es necesario definir una estrategia de pruebas para este tipo de producto que garantice la calidad del mismo, esta situación podría ocasionar falta de seguridad y eficiencia, además de generar insatisfacción en el cliente y la pérdida de prestigio y preferencia dentro del mercado nacional.

Por todo lo antes expuesto se considera plantear el siguiente **Problema científico de la investigación:** *¿Cómo contribuir a mejorar la calidad del producto Visor de Imágenes 2D del proyecto Vismedic?*

Para darle solución al problema anterior se propone como **objeto de estudio:** *El proceso de desarrollo de software* del *producto Visor de Imágenes 2D.*

Para darle solución al problema planteado se establece como **objetivo general**: *Desarrollar y aplicar una estrategia de pruebas para el producto Visor de Imágenes 2D, contribuyendo a mejorar la calidad requerida del producto.*

Por todo lo anterior se define como **campo de acción:** *El proceso de pruebas de software del producto Visor de Imágenes 2D.*

Para dar cumplimiento al objetivo general se plantean las **tareas de investigación** siguientes:

- ✓ Elaboración del marco teórico de la investigación a partir del estado del arte existente en Cuba y en el mundo.

- ✓ Estudio sobre las metodologías de desarrollo de software y las pruebas aplicadas en estas, facilitando así la selección de pruebas a aplicar al producto Visor de Imágenes 2D.
- ✓ Caracterización del producto Visor de Imágenes 2D, permitiendo obtener una mejor comprensión de este.
- ✓ Definición de la estrategia de prueba de software para seleccionar las pruebas que se van a aplicar al producto Visor de Imágenes 2D.
- ✓ Definición de las listas de chequeo para las pruebas de interfaz de usuario y usabilidad permitiendo la ejecución de estas.
- ✓ Realización del diseño de los Casos de pruebas para la ejecución de estas.
- ✓ Validación de los resultados de las pruebas.

Métodos de investigación

- ✓ Métodos teóricos:

  Histórico – lógico: Permite realizar una investigación sobre cómo ha evolucionado la Realidad Virtual en Cuba y el mundo.

  Analítico – sintético: Permite realizar un estudio profundo sobre estrategia de prueba de software, los tipos de pruebas referentes a los visores de imágenes.
- ✓ Métodos empíricos:

  Entrevista: El método de la entrevista se utilizó para obtener los problemas y necesidades del producto.

  Realización de pruebas: lo cual permitirá el proceso de validación de la estrategia.

La **idea a defender** es: *La aplicación de una estrategia de pruebas de software elaborada para el producto Visor de Imágenes 2D, permitirá que el software entregado a los clientes cumpla con los requisitos de software establecidos en el tiempo y con la calidad requerida.*

El trabajo estará estructurado en Introducción y 2 capítulos:

En el **Capítulo 1** se realiza un estudio del estado del arte de las pruebas de software aplicadas en el proceso de desarrollo de software en el mundo y en la UCI, además se realiza una investigación sobre las estrategias de prueba y los visores de imágenes médicas. Se caracteriza la situación actual de proyecto Vismedic y el producto Visor de Imágenes 2D.

En el **Capítulo 2** se describe detalladamente la estrategia de pruebas de software propuesta, se establecen los tipos de pruebas que se van a aplicar, así como los roles, actividades y artefactos a evaluar, se valida la estrategia propuesta y se analizan los resultados obtenidos en la ejecución de las pruebas realizadas.

Capítulo 1: Fundamentación Teórica

1.1 Introducción del capítulo

En este capítulo se abordarán temas como la calidad de un producto de software, sus principales conceptos y diferentes puntos de vista acerca del mismo. Otro aspecto a tratar lo constituyen las pruebas que se le aplican a un producto de software para determinar su calidad, teniendo en cuenta los niveles de pruebas por los que debe pasar un producto. Se llevará a cabo una caracterización general del proyecto Vismedic en la cual se expondrán temas como el surgimiento de este y las funcionalidades que brinda.

1.2 Calidad del software

El término calidad tiene muchas acepciones, pero la básica es aquella que dice: aquel producto o servicio que se adquiere satisfaga las expectativas sobradamente, es decir, que el producto o servicio funcione tal y como se quiere. La palabra "*Calidad*" siempre será entendida de manera diferente por cada persona, ya que para unos la calidad residirá en un producto y en otros en su servicio posventa.

Hoy en día se le han dotado de diferentes definiciones entre las que se pueden mencionar:
"*La calidad del software es el grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario*" (1).

"*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*" (2).

El modelo de calidad establecido en la primera parte del estándar ISO 9126-1 (Organización Internacional de Estandarización), clasifica la calidad del software en un conjunto estructurado de características y subcaracterísticas de la siguiente manera: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenimiento y movilidad (12).

La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de

software para ser explotado durante un largo período, necesita ser confiable, mantenible y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación (20).

1.3 Aseguramiento de la Calidad del Software

El Aseguramiento de la Calidad se puede definir como el esfuerzo empleado en plantear, organizar, dirigir y controlar la calidad en un sistema de producción, con el objetivo inmediato de dar al cliente productos con la calidad adecuada. El término “aseguramiento” es aplicable a cualquier técnica o método que se ponga en práctica para garantizar el cumplimiento de determinado objetivo. Se asume que es más rentable prevenir los fallos de calidad que corregirlos o lamentarlos y se incorpora el concepto de la "prevención" en este aspecto, bajo la denominación de Aseguramiento de la Calidad. ***CMMI*** *(Capability Maturity Model Integration)*, modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software desarrollado por el Instituto de Ingeniería del Software (SEI-*Software Engineering Institute*) de la Universidad Carnegie Mellon y publicado en su primera versión en enero de 2002, plantea que el Aseguramiento de la Calidad, como conjunto de procesos, debe encargarse de:

- Evaluar la ejecución de los procesos, los elementos de trabajo y servicios según las descripciones, estándares y procedimientos establecidos.
- Identificar, documentar y corregir los elementos no conformes.
- Informar a las personas involucradas en el proceso sobre los resultados de las actividades del aseguramiento de la calidad.

1.4 Cómo controlar la calidad del software

El control de la calidad es la parte de la gestión de la calidad orientada al cumplimiento de los requisitos de la calidad (3).

Al Control de la calidad del Software lo acompañan las técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centrados en 2 objetivos fundamentales:

- Mantener bajo control un proceso.
- Eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

Para controlar la calidad del software es necesario ante todo, definir los parámetros, indicadores o criterios de medición.

El término Control de la Calidad, se define como: *"Conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requerimientos relativos a la calidad del producto o servicio"* (4).

El control de la calidad desde la perspectiva de un producto de software tiene sus particularidades. Las actividades que se establecen deben centrarse en mantener bajo control el proceso de desarrollo del software, y eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

1.5 Normas y estándares de calidad

La industria del software ha llamado la atención del mercado, gracias a dos factores esenciales como son: la velocidad con que ha crecido y su alcance. El software se convirtió con el tiempo en un negocio rentable al haber tanta demanda en ese campo. Muchas personas empezaron a desarrollar software en diferentes países naciendo las primeras grandes empresas de software, trayendo consigo un problema natural.

Había tantos desarrolladores en distintos países y trabajando en distintas aplicaciones que comenzó a haber diversidad de estilos y la calidad del producto final variaba mucho. Es por ello la necesidad de crear un estándar que permitiera a los consumidores de software decidir si el producto que estaban recibiendo era de calidad y si cumplía ciertos requisitos de funcionalidad.

Según ISO, un estándar es: "*Un conjunto de acuerdos documentados que contienen especificaciones técnicas u otros criterios precisos para ser usados constantemente, como reglas, lineamientos, o definiciones de características. Todo esto con la finalidad de asegurar que los materiales, productos, procesos y servicios son óptimos para su propósito*".

Los estándares de calidad determinan el nivel mínimo y máximo aceptable para un indicador. Si el valor del indicador se encuentra dentro del rango significa que se cumple con el criterio de calidad que se había definido y que las cosas transcurren conforme a lo previsto.

1.5.1 Norma ISO 9001:2000

Uno de los propósitos principales de un estándar es promover un intercambio de productos en base a ciertos lineamientos comunes.

La '**Norma ISO 9001'** ha sido elaborada por el Comité Técnico ISO/TC176 de ISO y especifica los requisitos para un sistema de gestión de la calidad que pueden utilizarse para su aplicación interna por las organizaciones, para certificación o con fines contractuales.

La norma ISO 9001:2000 contiene la especificación del modelo de gestión y los requisitos del Modelo, los requisitos que han de cumplir los sistemas de la calidad a efectos de confianza interna, contractuales o de certificación.

Dicha norma está estructurada en ocho apartados, refiriéndose los cuatro primeros a declaraciones de principios, estructura y descripción de la empresa, requisitos generales y requisitos de documentación, es decir, son de carácter introductorio. Los apartados cinco a ocho están orientados a procesos y en ellos se agrupan los requisitos para la implantación del sistema de calidad.

El propósito de ISO 9001 es asegurar a los clientes que los proveedores pueden brindar productos y servicios de calidad. Está pensado para llenar las necesidades del cliente, las del proveedor son secundarias. Una organización debe de alcanzar y sostener la calidad de un producto o servicio producido para seguir en la búsqueda continua de las necesidades explícitas o implícitas del cliente.

1.5.2 Modelo de Capacidad y Madurez (CMMI)

CMMI es un modelo para la mejora o evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software. Fue desarrollado por el Instituto de Ingeniería del Software de la Universidad Carnegie Mellon (SEI) y publicado en su primera versión en enero del 2002 (13).

Un nivel de madurez es un sistema evolutivo y bien definido para alcanzar el proceso de madurez del software. Cada nivel de madurez tiene dentro de sí mismo parámetros que permitan la mejora continua. Alcanzar un nivel dentro de la escala de CMMI significa seguir en busca de mejores prácticas y a la vez mantener los logros alcanzados.

Los niveles de madurez tienen varios objetivos:

- Definir un orden para medir la calidad de software gracias a la madurez de la compañía.
- Ayudar a la organización a ver qué procesos debe de mejorar en forma gradual para alcanzar el nivel óptimo.
- Mantener un proceso bien documentado.
- Lograr un producto controlado, verificable, validado y medido.

Los niveles son: inicial, repetible, definido, administrado y óptimo. Estos se describirán a continuación.

Nivel 1: Inicial

Este nivel es el primer estado en la evolución de las organizaciones que desarrollan software. En éste nivel se encuentran todas las empresas que no han logrado implementar las prácticas básicas de administración de proyectos e ingeniería de software definidas a partir del nivel 2 o superiores.

Nivel 2: Repetible

En el nivel repetible se establecen prácticas básicas de administración de proyecto que permiten establecer control de requerimientos, calendarizaciones y costos. El equipo que desarrolló el proyecto puede aprovechar su experiencia e inversión en procesos para aplicarla en un nuevo proyecto.

Nivel 3: Definido

En este nivel la empresa ha definido un conjunto de procesos, metodologías y herramientas usados por todos los niveles involucrados en el proceso, tanto administrativos como desarrolladores. Existen pautas y criterios definidos para adaptar un proceso estándar a las necesidades y características propias de cada proyecto. El nivel de definición es detallado y completo. En el proceso ya no existe dependencia de esfuerzos individuales, pues todos conocen el proceso.

Dentro de este nivel las áreas de procesos que más aportan al proceso de pruebas son las de Validación y Verificación.

Área de Proceso Validación

El objetivo que se persigue con esta área de proceso es demostrar que todo producto puede satisfacer las necesidades por las que fue creado. Como metas a implementar y prácticas a desarrollar se encuentran las siguientes:

Objetivo Específico 1: Preparar la validación**.**

- Práctica Específica 1.1: Seleccionar los productos a validar.
- Práctica Específica 1.2: Establecer el entorno de validación.
- Práctica Específica 1.3: Establecer los procedimientos y criterios de validación.

Objetivo Específico 2: Validar los productos o componentes de los productos**.**

- Práctica Específica 2.1. Realizar la validación.
- Práctica Específica 2.2. Analizar los resultados de la validación.

Área de Proceso Verificación

Como objetivo de esta área se encuentra el asegurar que los productos de trabajo responden a los requerimientos planteados. Como metas a implementar y prácticas a desarrollar se encuentran las siguientes:

Objetivo Específico 1: Preparar la verificación

- Práctica Específica 1.1: Seleccionar los productos de trabajo para la verificación.
- Práctica Específica 1.2: Establecer el entorno de verificación.
- Práctica Específica 1.3: Establecer los procedimientos y criterios de verificación.

Objetivo Específico 2: Realizar revisiones por terceros

- Práctica Específica 2.1: Preparar revisiones por terceros.
- Práctica Específica 2.2: Realizar revisiones por terceros.
- Práctica Específica 2.3: Analizar resultados de revisiones por terceros.

Objetivo Específico 3: Verificar los productos de trabajo seleccionados

- Práctica Específica 3.1: Realizar la verificación.
- Práctica Específica 3.2: Analizar los resultados de la verificación.

Nivel 4: Administrado

En este nivel la organización mide la calidad del producto y del proceso de software. Ambas partes son seguidas en forma cuantitativa y se controlan mediante métricas detalladas. La capacidad de rendimiento del proceso es previsible.

Nivel 5: Óptimo

La característica principal aquí es que la organización entera lleva a cabo mejoras continuas en el proceso, en base a la retroalimentación cuantitativa y al ensayo de ideas y tecnologías innovadoras. Todos los cambios realizados en cualquier parte del proceso son vigilados y controlados con el sistema de métricas. Durante todo el desarrollo no suelen suceder errores.

1.6 Metodologías 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.

En estas se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

Actualmente es imprescindible considerar los riesgos, aunque habitualmente las empresas no han sido concienciadas de los riesgos inherentes al procesamiento de la información mediante ordenadores, a lo que han contribuido, a veces, los propios responsables de informática, que no han sabido explicar con la suficiente claridad las consecuencias de una política de seguridad insuficiente o incluso inexistente. Por otro lado, debido a una cierta deformación profesional en la aplicación de los criterios de coste/beneficio, el directivo desconocedor de la informática no acostumbra a autorizar inversiones que no lleven implícito un beneficio demostrable, tangible y mensurable.

Las técnicas indican cómo debe ser realizada una actividad técnica determinada, identificada en la metodología, combinando el empleo de unos modelos o representaciones gráficas junto con el empleo de unos procedimientos detallados. Se debe tener en consideración que una técnica determinada puede ser

utilizada en una o más actividades de la metodología de desarrollo de software, teniendo en cuenta cuando cambiar una técnica por otra.

La evolución de la disciplina de ingeniería de software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. Las metodologías tradicionales haciendo énfasis en la planeación y las metodologías ágiles haciendo énfasis en la adaptabilidad del proceso, delinean las principales propuestas presentes en la literatura. De manera paralela, el tema de modelos para el mejoramiento de los procesos de desarrollo ocupa un lugar importante en la búsqueda de la metodología adecuada para producir software de calidad en cualquier contexto de desarrollo (31).

1.6.1 Metodologías tradicionales en el desarrollo

Las metodologías tradicionales se caracterizan por exponer procesos basados en planeación exhaustiva. Esta planeación se realiza esperando que el resultado de cada proceso sea determinante y predecible. La experiencia ha mostrado que como consecuencia de las características del software, los resultados de los procesos no son siempre predecibles y sobre todo, es difícil predecir desde el comienzo del proyecto cada resultado. Sin embargo, es posible por medio de la recolección y estudio de métricas de desarrollo lograr realizar estimaciones acertadas en contextos de desarrollo repetibles (32).

Remontándose a la historia, el modelo de cascada fue uno de los primeros modelos de ciclo de vida que formalizó un conjunto de procesos de desarrollo de software. Este modelo describe un orden secuencial en la ejecución de los procesos asociados. El modelo espiral se postuló como una alternativa al modelo de cascada. La ventaja de este modelo radica en el perfeccionamiento de las soluciones encontradas con cada ciclo de desarrollo, en términos de dar respuesta a los requerimientos inicialmente analizados. El modelo de cascada y el modelo espiral suponen, de manera general, que los requerimientos del cliente no cambian radicalmente en el transcurso del desarrollo del sistema.

El proceso unificado propone la elaboración de varios ciclos de desarrollo, donde cada uno finaliza con la entrega al cliente de un producto terminado. Este se enmarca entre los conocidos modelos iterativo-incremental, el cual es nombrado como Proceso Unificado de Desarrollo de Software (RUP).

1.6.2 Metodologías ágiles

Algunos grupos de desarrollo han experimentado soluciones que basan su fundamento en la adaptabilidad de los procesos de desarrollo, en lugar de seguir esperando lograr resultados predecibles de un proceso que no evoluciona. Esta comunidad de desarrolladores e investigadores han nombrado su trabajo bajo lo que conocemos como metodologías ágiles. Las metodologías ágiles no están en contra de administrar procesos de desarrollo, por el contrario, promueven la formalización de procesos adaptables.

La compilación de los principios y valores que resaltan las metodologías ágiles fue formalizada en el manifiesto para el desarrollo de software ágil. Este documento desarrollado por los representantes de cada una de las metodologías que en el momento se presentaban como ágiles, logra resumir en un conjunto de ideas las prácticas que una metodología de este estilo debe llevar a cabo. Como característica fundamental, la habilidad de responder al cambio es la principal característica de las metodologías ágiles (33).

Programación extrema (XP), una de las más difundidas, es una metodología de desarrollo de software ágil que define pocas reglas y pocas prácticas. XP promueve la adaptabilidad de los procesos de desarrollo basándose en los principios y prácticas que presenta.

La programación extrema es una de las metodologías desarrollo de software ágil más usadas para proyectos de corto plazo y equipos pequeños. Se le llama extrema, porque define pocas reglas y pocas prácticas a través de la eliminación de procesos y técnicas que carecen de valor. En el caso de los proyectos que trabajan con XP, deben seguir procesos disciplinados, pero más que eso, deben combinar la disciplina con la adaptabilidad necesaria del proceso. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

XP, dadas sus características, es la metodología más apropiada para un entorno caracterizado por requerimientos cambiantes originados mayormente por un mercado inestable y el inminente avance de la tecnología y los negocios, por tanto es recomendable aplicarla en caso de que los requisitos estén vagamente definidos y los clientes se pueden involucrar la mayor parte del tiempo en el desarrollo del proyecto.

XP insiste en la importancia de una aplicación adecuada y eficiente de las pruebas. Se harán pruebas todo el tiempo, no sólo de cada nueva clase (**pruebas unitarias**) sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos (**pruebas funcionales**) Las pruebas de integración se efectuarán siempre, antes de añadir cualquier nueva clase al proyecto, o después de modificar cualquiera existente (**integración continua**).

Las metodologías de Cristal se basan en el principio de que tipos diferentes de proyectos requieren tipos diferentes de metodologías. La metodología escogida debe depender de dos factores: el número de personas en el proyecto y las consecuencias de los errores. Conforme al principio de las metodologías ágiles, Scrum recalca la imposibilidad de encontrar procesos definidos y repetibles cuando no existen problemas, personas ni ambientes definidos y repetibles.

Gracias a lo planteado anteriormente, se llega a la conclusión de que entre los tipos de metodologías existentes, las más completas son las metodologías tradicionales ya que a través de las mismas es posible lograr realizar estimaciones acertadas en contextos de desarrollo repetibles por medio de la recolección y estudio de métricas de desarrollo.

Dentro de este tipo de metodología resalta RUP, la cual tiene gran importancia por su forma disciplinada de asignar tareas y responsabilidades, llevar a cabo un desarrollo iterativo, administrar los requisitos y en especial verificar la calidad de software.

1.6.3 Metodología RUP

Una de las metodologías pesadas más conocidas y utilizadas es la Metodología RUP (Rational Unified Process) o Proceso Unificado de Desarrollo de Software, 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.

RUP es un proceso de desarrollo de software, define quién hace qué, cómo y cuándo. RUP define cuatro elementos trabajadores (roles), que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los artefactos (productos), que responden a la pregunta ¿Qué? y los flujos de trabajo

de las disciplinas que responde a la pregunta ¿Cuándo? Divide el proceso de desarrollo en ciclos, teniendo un producto final al terminar cada ciclo. Cada ciclo se divide en fases que finalizan con un hito donde se debe tomar una decisión importante.
Una iteración puede realizarse por medio de una cascada como se muestra en la figura 1.

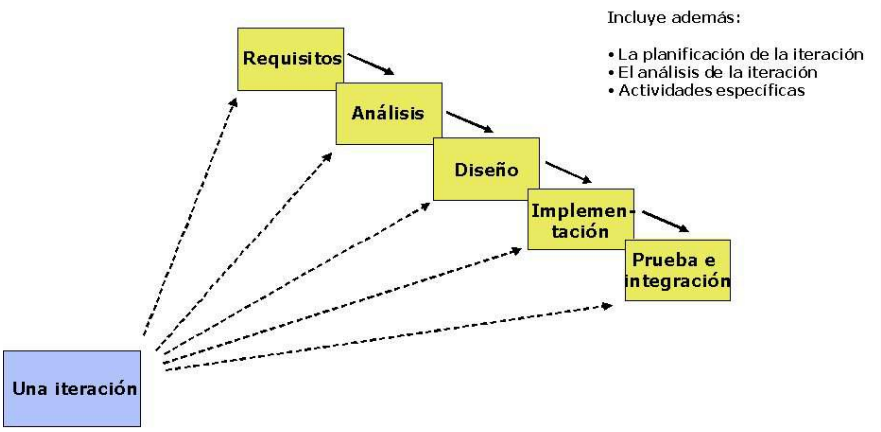

Figura 1 "Cascada".

Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar.

Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema. (5)

En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo como se muestra en la figura 2.

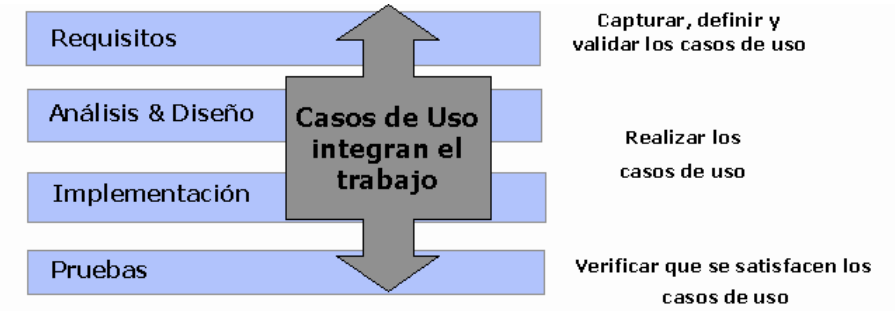

Figura 2 "Casos de usos".

1.6.3.1 Características principales de RUP

Las principales características de RUP son:

- Centrado en los modelos: Los diagramas son un vehículo de comunicación más expresivo que las descripciones en lenguaje natural. Se trata de minimizar el uso de descripciones y especificaciones textuales del sistema.
- 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.

1.6.3.2 Principios claves de RUP

La metodología RUP tiene 6 principios clave:

- **Adaptación del proceso:** El proceso debe adaptarse a las características de la organización para la que se está desarrollando el software.
- **Balancear prioridades:** Debe encontrarse un balance que satisfaga a todos los inversores del proyecto.
- **Colaboración entre equipos:** Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes y resultados.

- **Demostrar valor iterativamente:** Los proyectos se entregan, aunque sea de una forma interna, en etapas iteradas. En cada iteración se evaluará la calidad y estabilidad del producto y analizará la opinión y sugerencias de los inversores.
- **Elevar el nivel de abstracción:** Motivar el uso de conceptos reutilizables.
- **Enfocarse en la calidad:** La calidad del producto debe verificarse en cada aspecto de la producción.

1.6.3.3 Fases de RUP

RUP divide el proceso en cuatro fases (Inicio, Elaboración, Construcción y Transición), dentro de las cuales se realizan varias iteraciones en número variable según el proyecto. Ver Fig.3.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos y al establecimiento de una línea base de la arquitectura.

Durante la fase de inicio las iteraciones ponen mayor énfasis en actividades de captura de requisitos y modelado del negocio.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se seleccionan los Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realizan tantas iteraciones como sean necesarias hasta que se termine con la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

En cada fase participan todas las disciplinas, pero dependiendo de la fase, el esfuerzo dedicado a una disciplina varía.

1. **Inicio:** El objetivo es determinar la visión del proyecto y definir lo que se desea realizar.
2. **Elaboración:** Etapa en la que se determina la arquitectura óptima del proyecto.
3. **Construcción:** Se obtiene la capacidad operacional inicial.
4. **Transición:** Obtener el producto acabado y definido. La fase de Transición tiene como finalidad asegurar que el software esté disponible para sus usuarios finales. Los objetivos de esta fase es probar que el sistema cumpla con los requisitos funcionales y técnicas requeridas, y cumplir con la lista de verificación que da por culminado el producto.

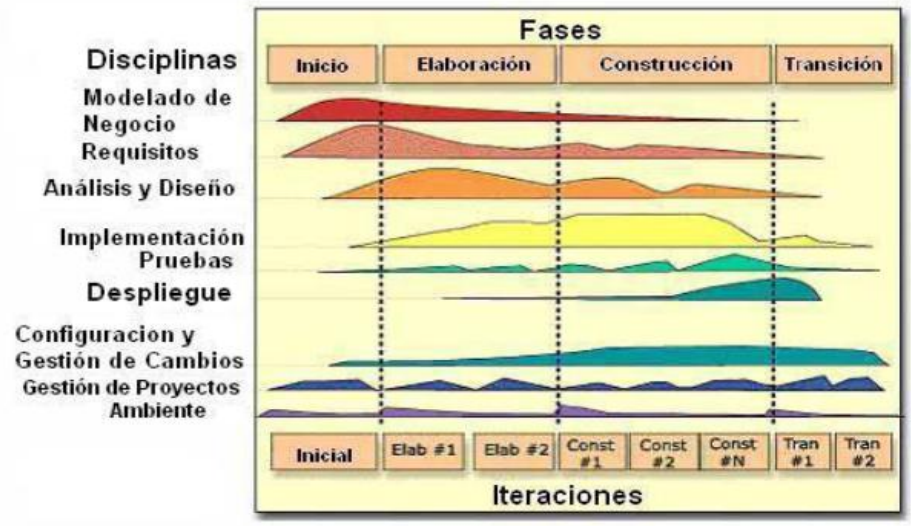

Figura 3 "Fases de RUP".

1.6.3.4 Flujos de trabajo de RUP

Los flujos de trabajos se definen a continuación:

1. **Ingeniería o modelado del negocio:** Analizar y entender las necesidades del negocio para el cual se está desarrollando el software.
2. **Requisitos:** Proveer una base para estimar los costos y tiempo de desarrollo del sistema.
3. **Análisis y diseño:** Trasladar los requisitos analizados anteriormente a un sistema automatizado y desarrollar una arquitectura para el sistema.

4. **Implementación:** Crear software que se ajuste a la arquitectura diseñada y que tenga el comportamiento deseado.
5. **Pruebas:** Asegurarse de que el comportamiento requerido es correcto y que todo lo solicitado está presente.
6. **Instalación o despliegue:** Produce release del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
7. **Configuración y Administración del Cambio:** Producir distribuciones del producto y distribuirlo a los usuarios.
8. **Administración de Proyectos:** Administrando horarios y recursos.
9. **Ambiente:** Administrando el ambiente de desarrollo.

1.6.3.5 Flujo de trabajo de prueba

El flujo de trabajo durante la prueba, incluyendo los trabajadores participantes y sus actividades:

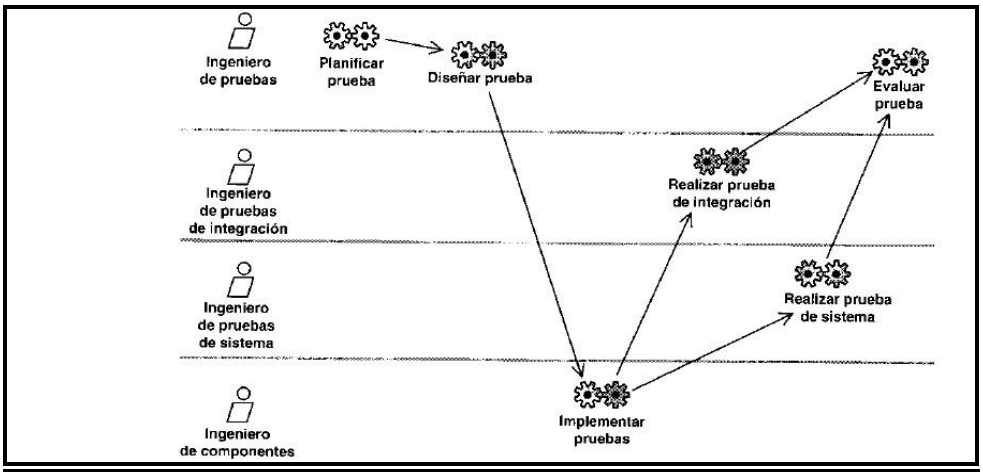

Figura 4 "Flujo de trabajo de prueba".

Objetivos de las pruebas

RUP plantea que los principales objetivos de las pruebas son:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas del sistema. Las pruebas de integración son necesarias para cada construcción dentro de la iteración, mientras que las pruebas de sistema son necesarias solo al final de la iteración.

- Diseñar e implementar las pruebas creando los casos de prueba que especifican que probar, creando los procedimientos de prueba que especifican como realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo como diseño o implementación, de forma que los defectos importantes puedan ser arreglados.

Artefactos de prueba de RUP

RUP define una serie de artefactos de prueba:

- **Modelo de Pruebas:** Describe principalmente cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema.
- **Caso de prueba:** Es uno de los pasos más importantes al realizar una estrategia de pruebas, ya que estos constituyen la fuente para evaluar los resultados del software, deben ofrecernos la posibilidad de encontrar la mayor cantidad de errores posibles en un tiempo y costo no muy elevados.
- **Procedimiento de Prueba:** Específica cómo realizar uno o varios casos de pruebas o partes de éstos. Puede ser una instrucción para un individuo sobre cómo realizar un caso de prueba manualmente, o una especificación de cómo interactuar manualmente con una herramienta de automatización de pruebas, para crear componentes ejecutables de prueba.
- **Componente de Prueba:** Automatiza uno o varios procedimientos de prueba o partes de ellos, se utilizan también para probar los componentes en el modelo de implementación, proporcionando entradas de pruebas, controlando y motorizando la ejecución de los componentes a probar e informando de los resultados de las pruebas, pueden ser implementados usando tecnología de objetos.
- **Plan de prueba:** Es el principal factor de éxito para la puesta en práctica de una estrategia de pruebas que permita entregar un software de mejor nivel, permite trazar el tipo de prueba que se le va a aplicar al producto, cuyo propósito es dejar de forma explícita el alcance, el enfoque, los recursos requeridos, el calendario y los responsables del proceso de pruebas
- **Defecto:** Es un síntoma de un fallo en el software o de un problema descubierto en una revisión.

- **Evaluación de Prueba:** Evaluación de los resultados de los esfuerzos de prueba. La evaluación es realizada por los diseñadores quienes comparan los resultados obtenidos con los objetivos trazados en el plan de prueba.

Roles de prueba según RUP

Un rol define el comportamiento y responsabilidades de un individuo, o de un grupo de individuos trabajando juntos como un equipo. (6)

Una misma persona puede desempeñar varios roles y un mismo rol puede ser representado por varias personas, en dependencia de la cantidad de personas que hayan disponibles en el proyecto para desempeñarlo.

El papel del probador es organizar, conducir y realizar una serie de actividades que permitan determinar la calidad del producto. Las pruebas pueden ser realizadas en conjunto con los responsables de los distintos roles de RUP.

La Metodología RUP define distintos tipos de roles de pruebas, los cuales están presentes durante el Flujo de Trabajo de Pruebas. (6)

- **Ingeniero de componentes:** Es el responsable de los componentes de prueba que automatizan algunos de los procedimientos de prueba.
- **Ingeniero de pruebas de integración:** Es el encargado de realizar las pruebas de integración que se necesitan para cada construcción producida en el flujo de trabajo de implementación. Se encarga también de documentar los defectos en los resultados de las pruebas de integración.
- **Ingeniero de pruebas de sistema:** Es el encargado de realizar pruebas al software, necesarias sobre una construcción que muestra y evalúa el resultado de una iteración completa.
- **Diseñador de Pruebas:** Planea las pruebas, es el responsable de la integridad del modelo de pruebas, asegurando que el modelo cumpla con su propósito. Selecciona y describe los casos de prueba y procedimientos de prueba. Además define el enfoque de prueba, las configuraciones del ambiente de prueba y los elementos de las pruebas. También identifica los mecanismos de pruebas y estructura la implementación de cada una de ellas.

1.7 Pruebas de Software

Las pruebas de software constituyen una fase del proceso de desarrollo de un software centrada en el favorecimiento a la calidad, fiabilidad y robustez del mismo, dentro del contexto o escenario previsto para ser utilizado. Es por ello que han recibido la importancia que merecen, como resultado de los trabajos dedicados por diversos autores a las mismas, en los que se han ofrecido un gran número de definiciones entre las que se encuentran las relacionadas a continuación:

- **Pressman:** Las Pruebas de software pueden definirse como "el *proceso de evaluación de un producto desde un punto de vista crítico, donde el "probador" (persona que realiza las pruebas) somete al producto a una serie de acciones indagadoras, y el producto responde con su comportamiento como reacción".* Es necesario probar los nuevos programas en un entorno de pruebas separado físicamente del de producción (2).

- **IEEE, 1990:** Una prueba puede ser: *"Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente" (1).*

- **RUP:** Una prueba *es una disciplina en el proceso de ingeniería de software cuyo objetivo es integrar y poner a prueba el sistema (21).* No es posible garantizar que un sistema esté correcto solo usando pruebas. Así que aunque se hagan esfuerzos extraordinarios, no podemos decir que el sistema está libre de defectos (22).

La importancia de las pruebas dentro del desarrollo del software se puede visualizar teniendo como referencia a los autores aludidos a continuación:

- **Dustin:** Las pruebas de software permiten pasar de forma confiable del cómodo ambiente planteado por la ingeniería de software, es decir del controlado ambiente de análisis, diseño y construcción, al exigente mundo real en el cual los entornos de producción someten los productos a todo tipo de fatiga (23).

- **Zuyu, Tsao, Wu:** Las pruebas de software basadas en componentes permiten la reutilización y por

ende la reducción de los ciclos de pruebas, lo cual se ve reflejado en la disminución de costos y tiempos (24).

- **Ilene Burnstein:** La necesidad de productos de software de alta calidad ha obligado a identificar y cuantificar factores de calidad como: capacidad de uso, capacidad de prueba, capacidad de mantenimiento, capacidad de ser medible, capacidad de ser confiable y a desarrollar practicas de ingeniería que contribuyen a la obtención de productos de alta calidad (25).

Bajo el nombre de pruebas de software se agrupan un conjunto de prácticas correctivas (frente a las prácticas preventivas que se aplican durante el proceso de construcción de software) cuyo objetivo es determinar la calidad de los sistemas software (7).

Un concepto más específico dado por algunos desarrolladores de software es que las pruebas son: "Cualquier intento de demostrar que el software tiene propiedades por debajo de la calidad requerida" (8).

La prueba es el proceso de ejecución de un programa con la intención de descubrir un error (2).

Las pruebas de software son la actividad más común de control de calidad realizada en los proyectos de desarrollo o mantenimiento de aplicaciones y sistemas. Aunque un aseguramiento de calidad de software más eficaz debería incluir otras técnicas como por ejemplo, inspecciones y revisiones (automatizadas o no) de modelos y documentos no ejecutables de las primeras fases de desarrollo.

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

Las pruebas de software no garantizan que un software esté libre de errores, sino que se detecten la mayor cantidad de defectos posibles en el mismo para su debida corrección. Si al ejecutar las pruebas no se encuentran errores esto indica que las pruebas realizadas no fueron lo suficientemente eficientes.

Según Kendall "las pruebas se realizan a lo largo del desarrollo del sistema y no simplemente al final. Esto significa sacar a la luz problemas no conocidos y no demostrar la perfección de programas" (9).

Las pruebas se realizan por subsistemas o módulos de programa según avanza el trabajo, se hacen en diferentes niveles y a diversos intervalos. Antes de que el sistema sea puesto en producción, todos los programas deben ser probados, revisados con datos de prueba y revisados para ver si los módulos trabajan juntos entre ellos. También debe ser probado el sistema trabajando como un todo. Esto incluye probar las interfaces entre subsistemas, la corrección de la salida y la utilidad y comprensibilidad de la documentación de la salida del sistema. Los programadores, analistas, operadores y usuarios juegan papeles diferentes en los diversos aspectos de la prueba (9).

La realización de pruebas es una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requisitos especificados, los resultados son observados y registrados, y se realiza una evaluación del sistema o componente (10).

Las siguientes definiciones son algunas de las recogidas en el diccionario de la IEEE en relación a las pruebas:

Pruebas: "es una actividad en la cual un sistema o uno de sus componentes se ejecuta dos veces en circunstancias previamente especificadas, los resultados se observan, se registran y se realiza una evaluación de algún aspecto".

Para Myers (11), probar (o la prueba) es el "proceso de ejecutar un programa con el fin de encontrar errores".

El nombre prueba, además de la actividad de probar, se puede utilizar para designar un conjunto de casos y procedimientos de prueba.

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

Defecto: "Un defecto en el software como, por ejemplo, un proceso, una definición de datos o un paso de procesamiento incorrecto en un programa".

Fallo: "La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados".

Error: Tiene varias acepciones:

- La diferencia entre un valor calculado, observado o medido y el valor verdadero, especificado o teóricamente correcto. Por ejemplo, una diferencia de dos centímetros entre el valor calculado y el real.
- Un defecto. Por ejemplo, una instrucción incorrecta en un programa.
- Un resultado incorrecto. Por ejemplo, un programa ofrece como resultado de la raíz cuadrada de 36 el valor 7 en vez de 6.
- Una acción humana que conduce a un resultado incorrecto (una metedura de pata). Por ejemplo, que el operador o el programador pulse una tecla equivocada.

Las pruebas se enfocan principalmente en la evaluación y aseguramiento de la calidad del producto, desarrollado a través de las siguientes prácticas:

- Encontrar fallas de calidad en el software y documentarlas.
- Recomendar sobre la calidad percibida en el software.
- Validar y probar las suposiciones hechas durante el diseño y la especificación de requerimientos de forma concreta.
- Validar que el software trabaja como fue diseñado.
- Validar que los requisitos son implementados apropiadamente.

1.7.1 Principios de las Pruebas

Para lograr una aplicación adecuada de los diferentes métodos de pruebas que existen se necesita que los ingenieros que trabajan en esta línea conozcan cuales son los principios que los ayudarán y guiarán en la realización del proceso de prueba que se va a efectuar.

Para Roger Pressman los principios básicos que guían el desarrollo de las pruebas de software son (2):

- La prueba puede ser usada para mostrar la presencia de errores, pero nunca de su ausencia.

- La principal dificultad del proceso de prueba es decidir cuándo parar.
- Evitar casos de pruebas no planificados, no reusables y triviales a menos que el programa sea verdaderamente sencillo.
- Una parte necesaria de un caso de prueba es la definición del resultado esperado.
- 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.
- El número de errores sin descubrir es directamente proporcional al número de errores descubiertos.
- Las pruebas deberían empezar por ¨ lo pequeño ¨y progresar hacia ¨ lo grande“.
- Con la excepción de las pruebas de unidad e integración, un programa deberá ser probado por la persona u organización que lo desarrolló.
- Asigna el programador más creativo a la prueba.

Otros de los principios de las pruebas fueron definidos por Davis (14), algunos de ellos se enumeran en los párrafos sucesivos a este comentario:

- ***A todas las pruebas se le debería poder hacer un seguimiento hasta los requisitos del cliente.*** Como se ha visto el principal objetivo es encontrar errores. Para el cliente los errores más graves son los que le impiden al sistema cumplir sus requisitos.

- ***Las pruebas deberían planificarse mucho antes de que empiecen.*** La planificación de las pruebas puede comenzar tan pronto como esté completo el modelo de requisitos. La definición detallada de los casos de prueba puede empezar una vez que se haya aprobado el modelo de diseño. Por tanto, se pueden planificar y diseñar todas las pruebas antes de generar ningún código.

- ***El principio de Pareto es aplicable a la prueba del software.*** El principio de Pareto implica que el 80 por ciento de todos los errores descubiertos durante las pruebas surgen al hacer un seguimiento de sólo el 20 por ciento de todos los módulos del programa. El problema está en aislar estos módulos sospechosos y probarlos.

- ***Las pruebas deberían empezar por lo pequeño y progresar hacia lo más grande***. Las primeras pruebas planeadas y ejecutadas en general se centran en módulos individuales del programa y a medida que avanzan las pruebas, se concentran en encontrar errores en grupos integrados de módulos y finalmente al sistema entero.

- ***No son posibles las pruebas exhaustivas.*** Esto se plantea porque incluso en un programa pequeño la cantidad de permutaciones de caminos es muy grande, por lo que es imposible cubrir todas las combinaciones de caminos. Sin embargo es posible cubrir adecuadamente la lógica del programa y asegurarse de que se han aplicado todas las condiciones del diseño procedimental.

- ***Para hacer más eficaces, las pruebas deberían ser realizadas por un equipo independiente.*** *Se* ha mostrado que el ingeniero de software que creó el sistema no es el más indicado para realizar las pruebas al sistema.

Luego de haber analizado los principios de las pruebas en el presente epígrafe, se llega a la conclusión de que el principio fundamental a tener en cuenta a la hora de realizar una prueba es que ésta no puede asegurar la ausencia de errores; solo puede demostrar que existen defectos en el software. Para llevar a cabo las pruebas existen diversos enfoques que determinan qué características del software se probarán, denominados tipos de pruebas.

1.7.2 Ejecución de pruebas

La ejecución de las pruebas se realizan de abajo hacia arriba, comenzando por las pruebas unitarias y terminando por las pruebas de aceptación, tienen una correspondencia directa entre las fases de desarrollo, los niveles de pruebas y los tipos de pruebas que se ejecutan.
En la figura 5 se muestra el modelo estándar de ejecución de pruebas:

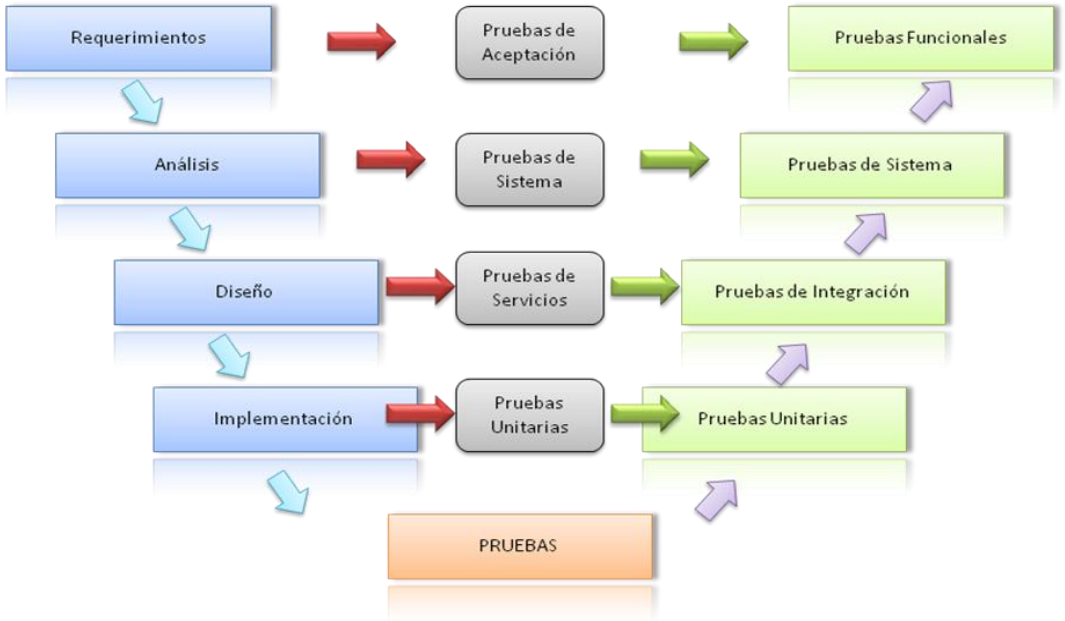

Figura 5 "Modelo estándar de ejecución de pruebas".

1.7.3 Tipos de pruebas de software

Según RUP, existen cuatro niveles de prueba: unidad, integración, sistema y aceptación. Un software se va perfeccionando mediante las iteraciones que se van realizando durante su ciclo de vida. Durante este proceso, las pruebas de software son fundamentales, permitiendo un fuerte acercamiento a los requerimientos establecidos con el cliente. En cada iteración, el equipo de desarrollo obtiene un resultado, siendo éste el principal candidato para las pruebas.

1.7.3.1 Pruebas de Unidad

Se basan en probar los componentes implementados como unidades individuales, estas pruebas, aíslan cada parte del programa y muestran que las partes individuales son correctas. A pesar de esto no descubrirán todos los errores del código, no descubren errores de integración, de rendimiento, ni otros problemas que afectan a todo el sistema en su conjunto.

Es la escala más pequeña de la prueba, está basada en la funcionalidad de los módulos del programa, como funciones, procedimientos, módulos de clase, etc. En ciertos sistemas también se verifican o se prueban los drivers y el diseño de la arquitectura (2).

Los casos de pruebas que se diseñen para este nivel de pruebas, deben descubrir errores como:

- Comparaciones entre tipos de datos distintos.
- Operadores lógicos o procedencia incorrecta.
- Igualdad esperada cuando los errores de precisión la hacen poco probable.
- Las variables o comparaciones incorrectas.
- Terminaciones de bucles inapropiadas o inexistentes.
- Fallo de salida cuando se encuentra una iteración divergente.
- Bucles que manejan variables modificadas de forma inapropiada.

Para probar los componentes implementados como unidades individuales, se realizan las pruebas de especificación o de caja negra, que verifican el comportamiento de la unidad observable externamente y pruebas de estructura, o de caja blanca, que verifican la implementación interna de la unidad.

1.7.3.2 Pruebas de Integración

Una vez que se les ha hecho la prueba de unidad a todos los módulos, nos preguntamos lo siguiente: “Si todos funcionan bien por separado, ¿por qué dudar de que funcionen todos juntos?”. Por supuesto, el problema es “ponerlos juntos” (interacción). Los datos se pueden perder en una interfaz; un módulo puede tener un efecto adverso e inadvertido sobre otro; las subfunciones, cuando se combinan, pueden no producir la función principal deseada; la imprecisión aceptada individualmente puede crecer hasta niveles inaceptables; y las estructuras de datos globales pueden presentar problemas; desgraciadamente, la lista sigue y sigue. 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.

La mayoría de los casos de prueba de integración pueden ser derivados de las realizaciones de caso de uso-diseño, ya que las realizaciones de casos de uso describen como interaccionan las clases y los objetos, y por tanto como interaccionan los componentes (6).

Se conocen dos tipos de integración, incremental y no incremental y aunque la selección de una estrategia de integración depende de las características del software y de la planificación del proyecto, en la mayoría de los casos existe la tendencia a aplicar la integración no incremental, es decir, a combinar todos los módulos por anticipado. Esto consiste en probar todo el programa en conjunto, lo que conduce normalmente al caos, debido a que se encuentra una gran cantidad de errores y la corrección de los mismos se hace difícil, puesto que es complicado aislar los errores al tener delante al programa completo. Una vez que se detectan esos errores aparecen otros nuevos y el proceso se repite en lo que parece ser un ciclo infinito.

Se aplica integración incremental cuando el programa se construye y se prueba en pequeños segmentos. Este tipo de integración tiene como ventajas que los errores son más fáciles de aislar y corregir, es más probable que se pueda probar completamente las interfaces y se puede aplicar un enfoque de prueba sistemática.

Existen dos estrategias de integración incremental:

- Integración Descendente
- Integración Ascendente

Integración Descendente

La prueba de integración descendente es un planteamiento incremental a la construcción de la estructura de programas. Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando por el módulo de control principal (programa principal).

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

El programa continúa desde el paso 2 hasta que se haya construido la estructura del programa entero.

Para llevar a cabo las pruebas de integración descendentes es necesario crear resguardos, los cuales son una serie de programas que reemplazan los módulos de bajo nivel. Esto se hace necesario cuando se requiere un proceso de los niveles más bajos de la jerarquía para poder probar adecuadamente los niveles superiores.

Para darle solución a este problema se tienen tres opciones (2):

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

Integración Ascendente

La prueba de la integración ascendente, como su nombre indica, empieza la construcción y la prueba con los módulos atómicos (es decir, módulos de los niveles más bajos de la estructura del programa).

Se puede implementar una estrategia de integración ascendente mediante los siguientes pasos (2):

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

A medida que la integración progresa hacia arriba se hace menos necesario el uso de los controladores de prueba.

Prueba de Regresión

La prueba de regresión consiste en volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios no han propagado efectos colaterales no deseados.

Prueba de Humo

La prueba de humo es un método de prueba de integración comúnmente utilizada cuando se ha desarrollado un producto software empaquetado. Es diseñado como un mecanismo para proyectos críticos por tiempo, permitiendo que el equipo de software valore su proyecto sobre una base sólida.

1.7.3.3 Pruebas de Sistema

Las Pruebas de Sistema, se realizan después de la construcción del sistema. Las mismas prueban a fondo el sistema, comprobando su funcionalidad e integridad globalmente, en un entorno lo más parecido posible al entorno final de producción.

Prueba de Recuperación

La prueba de recuperación es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente. Consiste en crear un evento de fallas o pérdida de datos, para que los usuarios vuelvan a cargar y recuperar a partir de una copia de respaldo. Con ello, se determina si los procedimientos de recuperación, son los más adecuados para cuando el sistema falle y no se pierdan los datos.

Prueba de Seguridad

Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios. Es el tipo de prueba que se encarga de certificar que los datos y las funciones del sistema solo son accesibles por los actores debidamente autorizados.

Pruebas de Stress o Resistencia

Es el tipo de prueba que se enfoca en comprobar cuál es el comportamiento del sistema bajo condiciones anormales, por ejemplo de carencia de recursos de memoria, procesador, sistemas externos con los que interactúa y carga excesiva de trabajo. El objetivo de estas pruebas es identificar las partes débiles del sistema (15).

Pruebas de Rendimiento o Carga

Está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. Es el tipo de prueba que se enfoca en comprobar los tiempos de respuesta del sistema

en una cantidad limitada de escenarios de trabajo (a nivel de números de usuarios y números de transacciones), bajo una configuración de hardware y software constante (15).

Prueba de Volumen Funcional

Este tipo de prueba se dedica a certificar la capacidad del sistema de manejar volúmenes de datos extremos, acorde a lo descrito en el documento de especificaciones funcionales (15).

Prueba de Compatibilidad

Consiste en asegurar que el software sea compatible con versiones determinadas de sistemas operativos, Web Server, bases de datos y demás componentes constituyentes de la arquitectura de la aplicación. Esta prueba también es conocida por otros autores como prueba de operabilidad y prueba de configuración (15).

Prueba de Usabilidad

Se trata de pruebas efectuadas con usuarios, con el objetivo de determinar si la organización de los contenidos y las funcionalidades que se ofrecen desde el Sitio Web son entendidas y utilizadas por los usuarios de manera simple y directa. Revisan una serie de factores con el fin de establecer si cumplen con las necesidades de los usuarios del sitio. Una prueba de usabilidad es contar con unos 5 usuarios en un laboratorio, que realizarán una navegación "asistida" por la aplicación a probar. El encargado de la prueba tomará nota de qué problemas encuentran los usuarios para realizar las tareas que se les hayan indicado, y así conocer qué errores de diseño tiene la aplicación.

Pruebas de Concurrencia o Contención

Es el tipo de prueba que se enfoca en certificar la capacidad del sistema de atender múltiples solicitudes de parte de los actores que acceden a un mismo recurso (un dato que esté almacenado en memoria, un conjunto de registros de bases de datos o una interfaz con un dispositivo de hardware o un sistema externo) (15).

Prueba de Funcionalidad

El objetivo de esta prueba es verificar la función del sistema al fijar la tensión en la validación de las funciones, métodos, servicios y casos de usos. Para validar la aplicación debe cumplir con los siguientes parámetros (15):

- Cumpla con los requisitos funcionales especificados en el diseño de la solución.
- Cumpla con los requisitos NO funcionales especificados en el diseño de la solución.
- Cumpla con las restricciones de entrada y salida de la información especificada en el diccionario de Datos.
- Cumpla íntegramente con la estructura referencial especificada en el Mapa de Navegación.

Prueba de Disponibilidad y Red

El objetivo de esta prueba es verificar el comportamiento de la aplicación cambiando la infraestructura de red al aplicar diferentes configuraciones, o retardos. Para validar la aplicación debe cumplir con los siguientes parámetros:

- Que no se reduzca la disponibilidad de los sistemas dentro de la solución, debido a la actividad de alguna persona o sistema. Ya sea, accidental o malintencionado.

Prueba de Instalación

Esta prueba se encarga de verificar que el sistema puede ser instalado en la plataforma del cliente y funciona correctamente.

Prueba de Configuración

Verifica que el sistema funciona en diferentes entornos de configuración. Ejemplo: Diferentes configuraciones de red (16).

Prueba de Negativas

Se encargan de intentar provocar que el sistema falle para así detectar sus debilidades. Se especifican Casos de Pruebas que intentan utilizar el sistema en formas para las que no ha sido diseñado, Ejemplo: Configuraciones de red incorrectas, capacidad de hardware insuficiente, carga de trabajo “imposible”, etc. (16).

Prueba de Vulnerabilidad

Es el tipo de prueba que se encarga de comprobar las vulnerabilidades del entorno de comunicaciones donde el sistema va a funcionar (la red), con el propósito de establecer los planes de contingencia adecuados y presupuestar los planes de adquisición correspondiente (15).

Prueba de Fiabilidad de Sistema

Este tipo de prueba considera la fiabilidad de un sistema como la probabilidad de que el mismo funcione o desarrolle una cierta función, bajo condiciones fijadas y durante un período de tiempo determinado. De la misma forma se define la fiabilidad de un dispositivo, aparato o persona, como casos particulares de "sistemas" en el ámbito de la ingeniería.

1.7.3.4 Pruebas de Aceptación

Estas pruebas se realizan para permitir que el cliente valide todos los requisitos. Las realiza el usuario final en lugar del responsable del desarrollo del sistema, una prueba de aceptación puede llegar hasta la ejecución sistemática de una serie de pruebas bien planificadas. La prueba de aceptación puede tener lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema.

Prueba Alfa y Beta

Es un proceso que llevan a cabo los desarrolladores para descubrir errores que parezca que solo el usuario final puede descubrir. La prueba alfa se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado. La prueba beta se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador.

1.7.4 Métodos de prueba

Existen dos métodos para la ejecución de las pruebas, el de pruebas de caja blanca y el de caja negra, a continuación se definen cada uno de ellos.

1.7.4.1 Pruebas de Caja Blanca

Las pruebas de caja blanca consisten en proporcionar que se cumpla al menos uno de los caminos básicos del módulo que se prueba.

Estas pruebas verifican la implementación interna de la unidad. Para cada componente se estudiará su implementación interna y se tratará de verificar su correcto comportamiento algorítmico. Se comprobarán los caminos comunes, los críticos, los menos conocidos y otros asociados con riesgos altos.

En las pruebas de caja blanca se observa siempre el código, las mismas se realizan para probarlo todo. Estas pruebas están basadas en estudiar el código fuente y se utilizan, principalmente, desde una perspectiva interna en el desarrollo de software, además se aplican mediante diferentes métodos.

Para esta prueba se consideran tres puntos importantes:

1. Conocer el desarrollo interno del programa, determinante en el análisis de coherencia y consistencia del código.
2. Considerar las reglas predefinidas por cada algoritmo.
3. Comparar el desarrollo del programa en su código con la documentación pertinente. La primera parte de esta prueba es el análisis estático.

Técnicas de Caja Blanca

- **Prueba de Condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- **Prueba de Flujo de Datos:** Es un método en el que se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **Prueba de Bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.
- **Prueba del Camino Básico:** Esta permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

  Para aplicar esta técnica se deben seguir los siguientes pasos:

  1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
  2. Se calcula la complejidad ciclomática del grafo.
  3. Se determina un conjunto básico de caminos independientes.

4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

1.7.4.2 Pruebas de Caja Negra

Las pruebas de caja negra son utilizadas para verificar que el sistema cumpla con los requisitos funcionales descritos. Estas pruebas se centran en el estudio de la especificación del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas. Las pruebas de caja negra están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario (ejemplo: canales de comunicaciones, etc.). Estas pruebas se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se corresponde con su especificación.

- El enfoque aleatorio consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

Técnicas de caja negra

Las técnicas de prueba de caja negra son:

- **Particiones o Clases de Equivalencia:** Es el método que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba, cada caso debe cubrir el máximo número de entradas, debe tratarse el dominio de valores de entrada dividido en un número finito de clases de equivalencia.

  La prueba de un valor representativo de una clase permite suponer razonablemente que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro valor de la clase.

  La identificación de las clases se realiza basándose en el principio de igualdad de tratamiento: todos los valores de la clase deben ser tratados de la misma manera por el programa.
- **Análisis de Valores Límite:** Pressman plantea que no hay una identificación clara de los valores límite, afirma que suele haber más errores en los valores límites que en los típicos. (17)

  Cuando se habla de valores límites lo que se dice es que se eligen casos de prueba en los límites o bordes de la clase que se está probando.

- **Conjetura de Errores:** Se enumera una lista de posibles equivocaciones típicas que pueden cometer los desarrolladores y de situaciones propensas a ciertos errores:
  - El valor cero es una situación propensa a error tanto en la salida como en la entrada.
  - En situaciones en las que se introduce un número variable de valores, conviene centrarse en el caso de no introducir ningún valor y en el de un solo valor. También puede ser interesante una lista que tiene todos los valores iguales.
  - Es recomendable imaginar que el programador pudiera haber interpretado algo mal en la especificación.
  - También interesa imaginar lo que el usuario puede introducir como entrada a un programa.

1.7.5 Estrategia de Prueba de software

El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca un fallo humano son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos pueden estar especificados de forma errónea o imperfecta así como en posteriores pasos de diseño y desarrollo (2).

Para obtener un producto de alta calidad, algunos autores como Pressman, plantean que las pruebas de software se deben realizar durante todo el ciclo de vida del mismo. Además para lograr el éxito total se deberá dar seguimiento a través de una estrategia de prueba.

La estrategia de prueba de software integra un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba: la planificación, el diseño de casos de prueba, la ejecución y los resultados, tomando en consideración cuánto esfuerzo y recursos se van a requerir, con el fin de obtener como resultado una correcta construcción del software (2).

Cada prueba que se realice (ya sea unitaria, integración, sistema, validación y verificación, estrés, configuración y/o instalación, aceptación) debe ser planificada con antelación, para de esta manera lograr que los resultados alcanzados sean los esperados y evitar la improvisación a la hora de ejecutarla.

Para esto, es necesario llevar a cabo un Plan de Prueba, en el cual se prevean los recursos necesarios (humanos, materiales y/o financieros), se definen los procedimientos y plantillas que serán utilizados, así como quién debe aplicar las pruebas y cuándo estas deben aplicarse (26).

Para lograr un mejor control del trabajo a realizar se deben confeccionar listas de chequeos, las cuales sirven para verificar el grado de cumplimiento de cada regla con un fin determinado.

La planificación de la estrategia de prueba puede reducir significativamente el esfuerzo necesario para el desarrollo de las pruebas adecuadas, reducir el tiempo de realización y ejecución de las mismas y disminuir los altos costos que se generan.

1.8 Visores de imágenes

Visor de imágenes es un programa de ordenador que puede mostrar gráficos de la imagen almacenada, sino que a menudo también pueden manejar varios formatos de archivos gráficos. Este tipo de software por lo general hace la imagen de acuerdo a las propiedades de la pantalla, tales como profundidad de color, resolución de la pantalla y el perfil de color.

Los visores de imágenes dan la máxima flexibilidad para el usuario, proporcionando una visión directa de la estructura de directorios disponibles en un disco duro. La mayoría de los visores de imágenes no proporcionan ningún tipo de organización automática de las imágenes y por lo tanto, la carga se mantiene en el usuario para crear y mantener su estructura de carpetas. Sin embargo, algunos visores de imágenes también tienen características de las imágenes, especialmente la organización de una base de datos de imagen, y por tanto también pueden ser utilizados como organizadores de la imagen.

Algunas de las características típicas de los visores de imágenes son los siguientes:

- Zoom y rotación
- Visualización a pantalla completa
- Presentación de diapositivas
- Imagen de pantalla
- Impresión
- Captura de pantalla

1.8.1 Ejemplos de Visores de imágenes médicas

- **Alma VISOR2D:** Es un visor de imágenes médicas obtenidas a partir de exploraciones radiológicas, desde aparatos de captación de diferentes modalidades. Alma VISOR2D es un completo visor radiológico, que incorpora las herramientas de visualización y valoración radiológica más avanzadas (27).
- **AMALTEA:** Plataforma integrada por un conjunto de aplicaciones desarrolladas como herramienta para el CIMES (Centro de Investigaciones Médico Sanitarias). AMALTEA nace bajo el paraguas de una necesidad perentoria en los servicios de salud pública: reducir los tiempos de respuestas entre las peticiones y la obtención de los resultados de las pruebas, y en el caso de los servicios externos: la comunicación directa entre proveedor y demandante de los servicios (28).
- **3Dicom Viewer:** Es uno de los últimos sistemas comercializados por 3Dicom, dedicada a la implantación y soporte de sistemas de código abierto para el mundo de la medicina. Tecnología para la transmisión y visión on-line de imágenes radiológicas que por primera vez permite conservar toda la información médica necesaria cuando se emplea la teleradiología en los centros sanitarios. Este visor de imágenes radiológicas funciona sobre cualquier navegador web y optimiza la descarga de las imágenes con la mínima información necesaria, utilizando la técnica del teselado (formación de la imagen en mosaico) y distintos niveles de zoom, empleada ya en los visores de mapas on-line actuales (29).

1.8.2 Formato DICOM para las imágenes médicas

En medicina se han realizado grandes avances tecnológicos, tanto en hardware como en software. En los últimos años el software ha adquirido mucha importancia tratando de estandarizar la comunicación entre equipos de tecnología propietaria. Gracias a muchas instituciones a nivel mundial que se han organizado han logrado crear un estándar de comunicación llamado DICOM (Digital Imaging and Comunications in Medicine).

En 1983, el Colegio estadounidense de Radiología (ACR) y la asociación nacional de fabricantes eléctricos (NEMA, National Electrical Manufacturers Association), formó un comité cuya misión era hallar o desarrollar una interface entre el equipamiento y cualquier otro dispositivo que el usuario quisiera conectar. Además de las especificaciones para la conexión del hardware, el estándar se desarrollaría para incluir un diccionario de los elementos necesarios para la interpretación y exhibición de las imágenes.

DICOM es un estándar médico oficial y libre, se utiliza en la mayoría de los hospitales de todo el mundo y su misión consiste en facilitar un medio de almacenamiento y digitalización de imágenes médicas, el almacenamiento se hace mediante un protocolo bajo TCP/IP (18).

Un archivo DICOM consiste en una cabecera, seguido de los datos de píxeles. La cabecera se compone, entre otras cosas, en el nombre del paciente, otros datos del paciente y los detalles de la imagen. Importante entre los detalles de la imagen son las dimensiones de anchura y altura, y los bits por pixel. Todos estos detalles están ocultos en el interior del archivo DICOM en forma de etiquetas y sus valores.

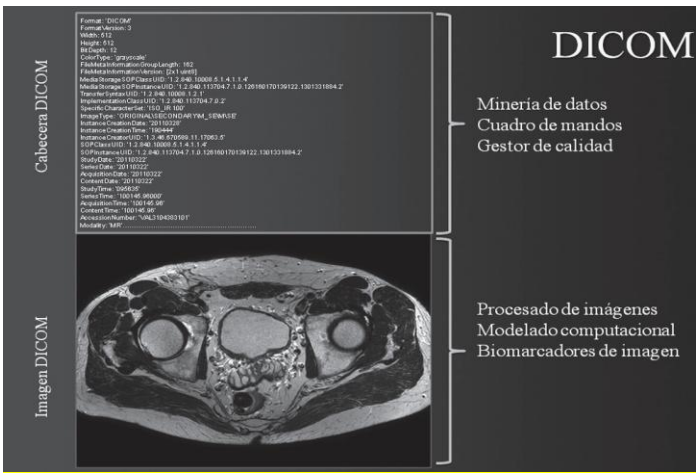

Figura 6 "DICOM".

En el estándar DICOM toda información asociada a la imagen médica comprende dos aspectos fundamentales para su tratamiento. La parte de imagen nos ayudará a desarrollar biomarcadores que permitan conocer y medir los parámetros específicos y relevantes a un proceso biológico, puede estar comprimida con distintos estándares., y la parte de datos con multitud de campos estandarizados que especifican tanto datos administrativos como datos sobre la imagen que permite explotar la información de las cabeceras DICOM para cuantificar y evaluar la calidad de los procesos implicados en la utilización de la imagen médica.

El estándar DICOM facilita la interoperabilidad de los equipos de imágenes médicas especificando:

- Para las comunicaciones de red, un conjunto de protocolos que deben ser seguidos por los dispositivos conformes al estándar.
- Sintaxis y semántica de los comandos y la información asociada que puede ser intercambiada usando estos protocolos.
- Para la comunicación de datos, un conjunto de servicios para el almacenamiento de datos que deben ser seguidos por los dispositivos conformes al estándar, así como un formato de fichero y una estructura de directorio médico para facilitar el acceso a las imágenes y a la información relacionada almacenada en medios de intercambio.
- La información que debe suministrarse con una implementación conforme al estándar.

El estándar DICOM no especifica:

- Detalles de implementación o cualquier característica del estándar de un dispositivo conforme a él.
- El conjunto global de características y funciones que se esperan de un sistema implementado integrando un grupo de dispositivos conformes al estándar DICOM.
- Un procedimiento de testeo/validación para calcular la conformidad de una implementación con el estándar.

El estándar DICOM en la actualidad se constituye en la columna vertebral de todo sistema de información de un hospital, definiendo formatos y protocolos de comunicación.

El estándar DICOM está ahora mismo en su versión 3.0 y es mantenido por los miembros del Comité de Estándares DICOM (DICOM Standards Committee), que está formado por organizaciones, vendedores de hardware y software para PACS y otros grupos de interés general (19).

1.9 Caracterización General del Proyecto Vismedic.

1.9.1 Inicios del proyecto Vismedic

Las investigaciones que convergen al proyecto Vismedic, se iniciaron a partir del proyecto “Simulador Quirúrgico”, hace al menos 4 años por dos estudiantes de la Facultad 5, que plasmaron estas investigaciones en su tesis de grado. Luego se continuó avanzando en este tema con el propósito de

hacer investigaciones sobre la visualización de imágenes para la salud, esta investigación siempre contó con la guía de los médicos especialistas. Luego de algunos años ya se contaba con resultados investigativos para formalizar la creación de un producto. La dirección del CEDIN decidió liberar un producto oficial del centro con estos resultados investigativos el cual tiene por nombre Visor de Imágenes 2D. Para la elaboración de este producto se contó con la guía de un especialista en radiología del Instituto de Oncología de La Habana.

La situación actual del proyecto es que urge la necesidad de entregar a tiempo este producto con todas las funcionalidades pactadas con el cliente y la calidad requerida, pero no cuenta el CEDIN con una estrategia de prueba que garantice lo anteriormente planteado para este tipo de producto.

1.9.2 Características generales del producto

El producto Visor de Imágenes 2D posee una interfaz moderna, sencilla e intuitiva que permite al usuario familiarizarse rápidamente con el entorno de trabajo. Este producto se ha desarrollado con la participación activa de un equipo de profesionales formado por jefe de línea, arquitecto, analistas y desarrolladores. Se desarrollaron una serie de funcionalidades que son de interés para el equipo médico de radiólogos del Hospital Oncológico de la Habana, que traen como beneficios el descubrimiento de enfermedades o padecimientos que pueden ser mortales para los pacientes.

El equipo de desarrollo hizo uso de las librerías DCMTK (Digital Comunication Standarization), con las cuales se pueden gestionar imágenes radiológicas, utilizando el estándar DICOM. Esta es una colección de librerías y aplicaciones, incluye software para examinar, construir y convertir archivos de imagen DICOM, gestión de medios en línea, enviar y recibir imágenes a través de una conexión de red, así como almacenamiento de imágenes demostrativas y servidores de listas de trabajo. DCMTK está escrito en una mezcla de ANSI, C y C + +. Se presenta el código fuente completo y está disponible como “open source”.

El producto es una aplicación de escritorio, no está conectada a un servidor como comúnmente suele estar este tipo de producto, aunque en un futuro próximo se llegará a la implementación de otros módulos que contengan esta característica.

El Visor 2D cuenta con una funcionalidad que permite guardar filtros aplicados a una imagen, esta configuración es guardada en un fichero, el cual se genera en tiempo de ejecución en la carpeta

home/vismedicdata para el caso de Linux, este sistema no hace uso de bases de datos, aunque utiliza QSQLite para la salva en este fichero de los filtros aplicados a las imágenes.

Funcionalidades desarrolladas:

- Notas en las imágenes: En esta funcionalidad el sistema permitirá al usuario registrar anotaciones digitales en las imágenes. El sistema permitirá al usuario hacer observaciones significativas (el estado o el movimiento del paciente) durante la visualización de la imagen médica. Estas observaciones serán registradas como anotaciones digitales para el radiólogo en el contexto del software.
- Indicar regiones en la imagen mediante flechas: El sistema permitirá indicar las regiones en la imagen mediante dibujos. En este caso el dibujo que va a indicar la región de interés en la imagen será una flecha.
- Realizar mediciones sobre la imagen: El sistema permitirá realizar mediciones de distancias, ángulos, diámetros, áreas circulares y rectangulares de forma visual sobre la imagen.
- Realizar operaciones de transformación y énfasis sobre la imagen: El sistema permitirá realizar operaciones de zoom, translación, rotación, reflejo, ancho y nivel de ventana (W/L) y Lupa interactiva sobre la imagen de forma visual.
- Aplicar herramienta Lupa de forma interactiva sobre subregiones rectangulares de la imagen elegidas por el usuario: El sistema debe permitir al usuario tener una visualización interactiva de subregiones de interés de la imagen con un nivel de aumento predeterminado.
- Previsualizar información de un directorio de imágenes o de una imagen en particular en el momento en que se va a importar: El sistema debe mostrar la información de una imagen una vez que se desee importar la misma como vista previa del resultado que se obtendrá una vez que se desee importar.
- Navegar por el estudio DICOM: El sistema permitirá navegar por el estudio DICOM visualizando manual y automáticamente dichas imágenes.

Para la implementación de estas funcionalidades fue necesario tener en cuenta ciertas características técnicas que posibiliten un alto nivel de eficiencia, rendimiento, con un rápido grado de velocidad de procesamiento y cálculo de respuesta, así como alta recuperación y disponibilidad, por tanto las

computadoras deben tener al menos un microprocesador Pentium IV 3.0 GHz y como mínimo una memoria RAM de 1GB y una RAM de video de 128 Mg, además debe tener instalado el sistema operativo Ubuntu 11.04 de 64 bit.

En la implementación se utilizó el lenguaje de programación C++, bajo el paradigma de Programación Orientada a Objetos (POO) y para la interfaz gráfica se empleó Qt Framework, además en la programación solo se utilizaron bibliotecas libres. El formato que deben tener las imágenes de este sistema es DICOM, y para el acceso a estas se usará DCMTK. Para un mejor entendimiento de las funcionalidades del sistema se ha confeccionado un manual de usuario, que le proporcionará a los mismos toda la información que necesite.

Conclusiones del capítulo

El estudio realizado a los visores de imágenes médicas con respecto a la calidad permite concluir que para estos sistemas es de vital importancia tener un control de la calidad el cual pueda respaldar el correcto funcionamiento de este. En el capítulo se detallan los diferentes tipos de pruebas propuestos por RUP, de los cuales se seleccionarán las pruebas a aplicar al software teniendo en cuenta sus características. La investigación realizada arrojó convincentes argumentos para realizar una propuesta de estrategia de pruebas a seguir.

Capítulo 2: Descripción de la Solución Propuesta

2.1 Introducción del capítulo

En el siguiente Capítulo se detalla la estrategia de prueba en general que se propone para el aseguramiento de la calidad del producto Visor de Imágenes 2D, donde se definen el flujo de trabajo y sus actividades, los artefactos de entrada y salida. El flujo de trabajo que se definió para llevar a cabo esta estrategia es el que propone RUP ya que el proyecto se guió por esa metodología para su desarrollo. A este proyecto solo se le aplicarán pruebas de sistema ya que no hay tiempo para la ejecución de las demás, y estas se van a realizar de acuerdo a las características del producto así como los recursos requeridos para garantizar la calidad requerida por este. En este capítulo, además se validará la solución propuesta, donde se recogerán los resultados obtenidos después de la ejecución de las pruebas en la plantilla de No Conformidades y se evaluarán dichos resultados, obteniendo un informe final de las pruebas.

2.2 Flujo de Trabajo de Pruebas

Entre los objetivos que se persiguieron para desarrollar las pruebas al producto Visor de Imágenes 2D, se encuentran los siguientes:

- La correcta instalación del producto.
- El correcto funcionamiento de las funcionalidades previstas en los requisitos.
- El correcto funcionamiento de la base de datos utilizada para almacenar las imágenes.

El flujo de trabajo que a continuación se muestra describe las distintas actividades que se llevarán a cabo para realizar cada una de las pruebas. De manera general, se debe comenzar con la planificación, en esta actividad se genera el plan de pruebas un artefacto de vital importancia para el desarrollo de las mismas puesto que en este se tienen en cuenta aspectos como los recursos, requerimientos y los resultados que arrojan las pruebas. Posteriormente se desarrollan de forma paralela el diseño y la configuración del ambiente de pruebas. Luego de concluidas estas actividades, se procede a ejecutar las mismas, donde se documentan las no conformidades detectadas y se registran los problemas surgidos. Finaliza el flujo con la evaluación de las pruebas, actividad en la que

se concilian las no conformidades detectadas con el equipo de desarrollo y se emite el expediente de pruebas. En caso de que los resultados de las pruebas no fueran factibles, se procede a ejecutar otra iteración del flujo de prueba.

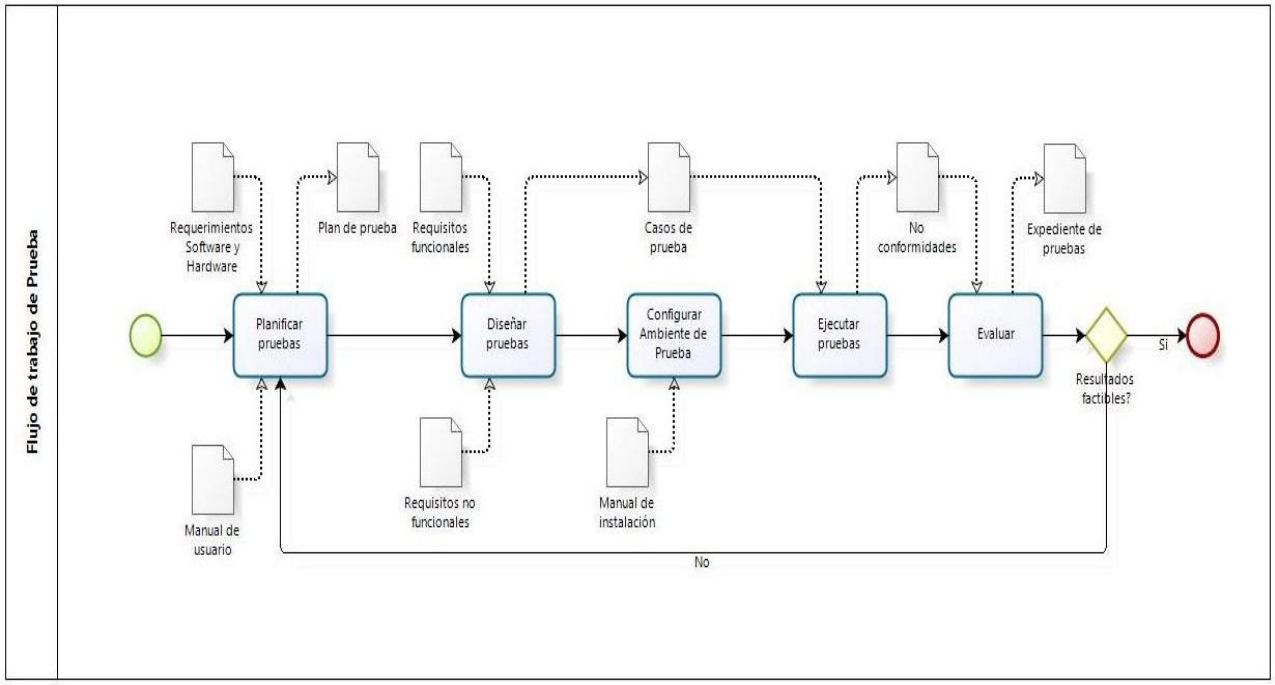

Figura 7 "Flujo de Trabajo Pruebas".

2.3 Descripción de las Actividades del flujo de trabajo

A continuación se describen cada una de las actividades que contiene el flujo de trabajo de pruebas representado anteriormente.

2.3.1 Planificar pruebas

El propósito de esta actividad es definir el alcance que van a tener las pruebas y sus objetivos teniendo en cuenta los recursos necesarios para su realización. Cuando se habla de recursos se deben tener en cuenta tanto los materiales como los humanos. El plan de pruebas es el artefacto generado en esta

actividad, el cual, puede estar sujeto a cambios a lo largo de toda la vida del proyecto. El administrador de pruebas es el encargado de realizar esta actividad.

Pasos:

1. Identificar los recursos necesarios para realizar las pruebas.
   - Recursos Humanos.
   - Recursos Software.
   - Recursos Hardware.

2. Confeccionar Plan de Pruebas.
3. Aprobación del Plan de pruebas por parte del equipo de desarrollo y el equipo de calidad.

   Entradas:
   - Manual de usuario.
   - Requerimientos Software y Hardware.

   Salidas:
   - Plan de pruebas.

2.3.2 Diseñar pruebas

El objetivo de esta actividad es identificar, describir y diseñar los casos de prueba y el procedimiento de la prueba, estos son los artefactos generados en esta actividad, el responsable de realizarla es el diseñador de pruebas. Luego de identificadas las pruebas, el diseñador procede a seleccionar las técnicas que se deben utilizar para el tipo de software con el que se trabaja, analiza los objetivos de las pruebas que se han planificado y para el desarrollo de los casos de prueba es necesario la plantilla de la descripción de los requisitos.

Pasos:

1. Diseñar los casos de pruebas, determinando las condiciones del software a probar.
2. Aprobación de los casos de pruebas a aplicar, por parte del equipo de aseguramiento de la calidad.

   Entradas:

- Plan de Prueba.
- Requisitos funcionales.

Salidas:

- Casos de prueba.
- Especificación de los casos de prueba.

2.3.3 Configurar ambiente de prueba

Esta actividad tiene como propósito asegurar todas las condiciones de hardware y software necesarias para la ejecución de las pruebas, además de garantizar las herramientas necesarias. El encargado de realizar esta actividad es el probador.

Pasos:

1. Crear y configurar ambiente de pruebas.
2. Capacitar al equipo de probadores.

Entradas:

- Plan de Prueba.
- Requisitos no funcionales.
- Manual de Instalación.

Salidas:

- Entorno de prueba configurado.

2.3.4 Ejecutar pruebas

Esta actividad tiene como objetivo llevar a cabo las pruebas de software. Durante esta actividad se ejecutan las pruebas previamente planificadas y diseñadas. El probador es el encargado de ejecutar todas las pruebas, durante la ejecución de las mismas se registran los resultados de las pruebas y se realizan reportes de No Conformidades (NC) por parte del probador.

Pasos:

1. Ejecutar casos de pruebas y listas de chequeo.
2. Registrar las No Conformidades.

Entradas:

- Plan de Prueba.
- Casos de prueba.
- Especificación de los casos de prueba.

Salidas:

- Reporte de No Conformidades.

2.3.5 Evaluar pruebas

En esta actividad se realiza un análisis final de las pruebas aplicadas y se confeccionan informes finales. Los diseñadores de pruebas llevan a cabo esta actividad revisando y evaluando los resultados de las pruebas. Luego el administrador de pruebas, se encarga de emitir el documento donde se registran las no conformidades finales y se decide si el resultado de las pruebas fueron factibles o no.

Pasos:

1. Evaluar el Reporte de No Conformidad.
2. Confeccionar Informe final.

   Entradas:

   - Reporte de No Conformidades.

   Salidas:

   - Informe final.

2.4 Descripción de los artefactos de flujo de trabajo

2.4.1 Plan de pruebas

El propósito del plan de pruebas es explicitar el alcance, enfoque, recursos requeridos, calendario, responsables, organización y la estrategia de las pruebas que se van a aplicar. Se puede definir un plan de pruebas de forma global y uno para cada tipo de prueba, en este caso se utilizó uno de forma global, el mismo se realizará haciendo uso de la plantilla propuesta por el Centro de Calidad para Soluciones Informáticas de la UCI (Calisoft). (Ver **Anexo 1**).

2.4.2 Expediente de pruebas

El propósito de este artefacto es archivar todos los documentos generados durante la etapa de pruebas. El expediente de pruebas debe incluir:

1. Plan de prueba.
2. Diseño de Casos de pruebas.
3. Reporte de No Conformidades.
4. Informe final.

Todos estos documentos se deben redactar siguiendo las plantillas propuestas y se deben recoger en una carpeta con el nombre de expediente de pruebas.

2.4.3 Casos de prueba

El propósito de este artefacto es definir un conjunto de condiciones o variables bajo las cuales se determinará si el requisito de una aplicación es parcial o completamente satisfactorio. Los casos de pruebas se deben diseñar a partir de la plantilla propuesta por la dirección de calidad de la universidad. (Ver **Anexo 2**).

2.4.4 Especificación de los casos de prueba

Es un documento donde se describe de forma detallada el alcance de cada caso de prueba así como sus flujos principales y alternos. Este documento se debe desarrollar a partir de la plantilla propuesta. (Ver **Anexo2**). Según la IEEE Std 829-1998 (IEEE Standard for Software Test Documentation), este documento debe contener (30):

1. Descripción.
2. Elementos a probar.
3. Condiciones de ejecución.
4. Entradas.
5. Salidas.
6. Configuración de Ambiente de pruebas.
7. Evaluación de la Prueba.

2.4.5 Reporte de No Conformidades (NC)

En esta tabla se deben recoger las NC detectadas por cada uno de los probadores durante la ejecución de cada caso de prueba. El mismo se debe desarrollar siguiendo el formato propuesto por la universidad. (Ver **Anexo 3**).

2.4.6 Informe final

Es un documento en el cual se registran las no conformidades finales, se redacta una evaluación de los resultados donde se decide si los resultados de las pruebas realizadas fueron positivos o no.

2.5 Propuesta de pruebas a aplicar

Para la estrategia de pruebas a aplicar se realizó un estudio de las pruebas de software que propone RUP y se analizaron las características generales del producto Visor de Imágenes 2D y sus funcionalidades. No es necesario incluir en la estrategia de prueba todas las pruebas que se proponen en RUP, ya que la aplicación no almacena datos estructurados, no presenta requisitos de seguridad, no gestiona usuarios y sesiones, no se conecta a una infraestructura de red, es un sistema sencillo que no consume grandes recursos, las funcionalidades implementadas no necesitan de gran procesamiento de cálculo y de datos, a pesar de lo anteriormente expuesto el sistema es de vital importancia para el sistema de salud en cuba, las funcionalidades implementadas resuelven un problema objetivo en el procesamiento de imágenes médicas, permitiendo aplicar filtros , contrastes, mediciones de área y distancias a las imágenes, por lo que se concluye que se aplicarán las pruebas de sistema: instalación, compatibilidad, funcionalidad, interfaz de usuario y usabilidad que se detallan a continuación.

2.5.1 Prueba de instalación

Descripción: Esta prueba se encarga de verificar que el sistema pueda ser instalado en la plataforma del cliente y funciona correctamente.

Objetivo: Instalar el sistema correctamente en el ambiente del cliente.

Técnica: Se realizará mediante la instalación del producto, verificando que el producto se instale correctamente.

Responsable de ejecutar la prueba: Probador.

Responsable de diseñar la prueba: Diseñador de Pruebas.

Procedimiento para realizar la prueba: El diseñador es el encargado de diseñar los casos de pruebas para medir que el sistema se haya instalado correctamente. El administrador de pruebas se encarga de aprobarlos, para que luego el probador los ejecute. Este debe verificar que el sistema se ejecute según los requisitos planteados.

2.5.2 Prueba de funcionalidad

Descripción: Esta prueba se encarga de verificar que el software cumpla con los requerimientos funcionales definidos.

Objetivo: Evaluar las funcionalidades del sistema funcionando como un todo.

Técnica: Se van a realizar mediante el chequeo del cumplimiento de todas las funcionalidades descritas en la especificación de requisitos funcionales mediante el método de caja negra aplicando la técnica de partición de equivalencia.

Responsable de ejecutar la prueba: Probador.

Responsable de diseñar la prueba: Diseñador de Pruebas.

Procedimiento para realizar la prueba: El diseñador es el encargado de diseñar los casos de pruebas para medir la funcionalidad del sistema. El administrador de pruebas se encarga de aprobarlos, para que luego el probador los ejecute. El probador verifica que las funcionalidades implementadas funcionen de forma correcta, para así concluir las pruebas documentando todos los resultados que se obtengan.

2.5.3 Prueba de interfaz de usuario

Descripción: Son caracterizadas por verificar la consistencia en la interfaz de usuario.

Objetivo: Determinar si la organización de los contenidos y las funcionalidades que se ofrecen son correctos.

Técnica: Se realiza mediante una lista de chequeo.

Responsable de ejecutar la prueba: Probador.

Responsable de diseñar la prueba: Diseñador de pruebas.

Procedimiento para realizar la prueba: En esta prueba se comprueba si la interfaz del software es clara y entendible, se revisan una serie de factores con el fin de establecer si el sistema cumple con las necesidades requeridas utilizando una lista de chequeo. (Ver **Anexo 5**).

2.5.4 Prueba de compatibilidad

Descripción: La prueba consiste en asegurar que el software sea compatible con versiones determinadas del sistema operativo.

Objetivo: El objetivo de esta prueba es comprobar que el sistema sea compatible con el sistema operativo a utilizar.

Técnica: Se realiza mediante la instalación del producto en diferentes entornos.

Responsable de ejecutar la prueba: Probador.

Responsable de diseñar la prueba: Diseñador de pruebas.

Procedimiento para realizar la prueba: Para la ejecución de esta prueba es necesaria la instalación del sistema operativo requerido y comprobar que el software funciona correctamente sobre este.

2.5.5 Prueba de usabilidad

Descripción: Las pruebas de usabilidad consisten en que se lleven a cabo las tareas para las cuales fue diseñada, se toma nota de la interacción del usuario con el sistema, particularmente de los errores y dificultades con las que se encuentren.

Objetivo: Validar el grado de usabilidad empírico del sistema. Medir la facilidad de aprendizaje con la que nuevos usuarios desarrollan una interacción efectiva con el sistema o producto.

Técnica: Se realizarán probando la usabilidad del producto mediante un cuestionario.

Responsable de ejecutar la prueba: Probador.

Responsable de diseñar la prueba: Diseñador de pruebas.

Procedimiento para realizar la prueba: Esta prueba se realiza a través de un cuestionario, donde cada pregunta evaluará un aspecto clave de usabilidad, una vez que el usuario haya interactuado con la aplicación, posterior se evalúa el resultado del cuestionario. (Ver **Anexo 4**).

2.6 Recursos requeridos

A continuación se definen los recursos que se necesitan para lograr la correcta ejecución de las pruebas que se le aplicarán al producto, mencionadas anteriormente.

2.6.1 Recursos humanos

Los roles y las responsabilidades definidas para este equipo de pruebas, están pensadas para realizar un desarrollo del proceso de pruebas más eficiente y eficaz. Para llevar a cabo las pruebas es necesario que este equipo este compuesto por al menos una persona que desempeñe el rol de Administrador de Pruebas, una persona que desempeñen el rol de diseñador de pruebas, y al menos dos probadores.

Tabla 1 "Roles y responsabilidades del equipo de pruebas".

Rol	Responsabilidades	Mínimo recomendados

Administrador de pruebas	➢ Asegurar que la aplicación producida se ajusta a las especificaciones y está razonablemente libre de errores. ➢ Coordinar la ejecución de las pruebas. ➢ Elaborar el plan de prueba. ➢ Evaluar los resultados que se obtienen en las pruebas.	1 Persona
Diseñador de Pruebas	➢ Diseñar los casos de prueba. ➢ Documentar el resultado de las pruebas realizadas al software. ➢ Definir listas de chequeo.	1 Persona
Probador	➢ Ejecutar las pruebas diseñadas. ➢ Anotar los resultados obtenidos.	2 Persona

2.6.2 Recursos software

En esta sección se describe los recursos de software necesarios para desarrollar las pruebas de manera exitosa.

Tabla 2 "Recursos de software".

Recurso	Software Instalado	Descripción
PC1	➢ Ubuntu 11.04 de 64 bit ➢ Visor de Imágenes 2D	PC donde se instalará el producto: Visor de Imágenes 2D y se realizarán las pruebas.
PC2	➢ Windows XP ➢ Microsoft Office 2007	PC donde se diseñarán los casos de prueba y se registrarán las No Conformidades.

2.6.3 Recursos hardware

En esta sección describe los recursos necesarios para el desarrollo de las pruebas. A continuación se exponen los recursos mínimos a tener en cuenta a la hora de realizar las pruebas.

Tabla 3 "Recursos de hardware".

Recurso	Configuración	Descripción
PC1	➢ Procesador: Pentium IV 3.0 GHz	Requisitos mínimos de la PC, donde se

	➢ Memoria RAM: 1GB ➢ Tarjeta de video: 128 MB ➢ Disco duro: 80 GB	ejecutarán las pruebas.
PC2	➢ Procesador: Pentium IV 3.0 GHz ➢ Memoria RAM: 256 MB ➢ Tarjeta de video: 64 MB ➢ Disco duro: 40 GB	Requisitos mínimos de la PC, donde se registrarán las pruebas y se diseñarán los casos de prueba.

2.7 Criterios de evaluación

Los criterios de evaluación estarán dados de forma independiente para cada tipo de pruebas; el siguiente cuadro muestra los criterios de evaluación generales de las pruebas ejecutadas.

Tabla 4 "Criterios de evaluación generales".

Tipos de prueba	Criterio de evaluación
Prueba de instalación	➢ Instalación y desinstalación correcta del software.
Prueba de funcionalidad	➢ Detectar errores en la ejecución de las pruebas. ➢ El 90% de las pruebas realizadas deben ser exitosas. ➢ El resultado de cada caso de prueba debe ser igual al resultado de salida esperado.
Prueba de interfaz de usuario	➢ La aplicación cumple con los requerimientos funcionales especificados. ➢ El 90% de las pruebas realizadas deben ser exitosas. ➢ Detectar errores en la ejecución de las pruebas.
Prueba de compatibilidad	➢ La aplicación cumple con los requerimientos no funcionales especificados.
Prueba de usabilidad	➢ La aplicación cumple con los requerimientos funcionales especificados. ➢ Más del 70% de las respuestas deben ser positivas y regulares.

2.8 Validación

Una de las etapas más importantes en el desarrollo de las pruebas la constituye los resultados que se obtienen en las mismas, los cuales permiten medir la calidad con la que se encuentra el producto probado, a continuación se describen los resultados de la aplicación de la estrategia de prueba.

Se realizaron todas las pruebas que fueron seleccionada en la estrategia de prueba definida, estas pruebas fueron ejecutadas haciendo uso de los artefactos vinculados a cada uno de ellas, para las pruebas de funcionalidad se utilizaron los casos de pruebas diseñados, **Anexo 2**, apoyados en la especificación requisitos funcionales entregados por la analista del proyecto.

Como resultados de las pruebas quedó conformado el Documento de No Conformidades, **Anexo 3**, donde se ponen de manifiesto los resultados obtenidos en la realización de las pruebas.

Se encontraron problemas en la ejecución del caso de prueba *01_DCP Abrir imagen DICOM* escenario 2 donde la no conformidad encontrada es de importancia baja y en el caso de prueba *02_DCP Agregar notas en las imágenes claves* en la sección “Gestionar Texto” en los escenarios 3 y 4, y en la sección “Gestionar Flechas” en el escenario 2, donde las 3 no conformidades encontradas son de importancia media, por lo que se evalúa de insatisfactoria al no superar las pruebas de funcionalidad el 90% requerido para este tipo de pruebas.

En el momento que se realiza esta evaluación solo se ha realizado una sola iteración, el equipo de proyecto se encuentra arreglando las no conformidades recogidas en el **Anexo 3**.

Para la realización de las pruebas de usabilidad se hizo uso de la encuesta, presente en el **Anexo 4**, donde los resultados arrojaron algunos criterios donde se le sugiere al equipo de desarrollo que modifiquen algunos aspectos encontrados para mejorar la interacción del usuario con la aplicación, para la evaluación de la misma se toma que 1 es mal, 2 es regular y 3 bien.

PREGUNTA	CRITERIOS DE EVALUACIÓN
1. ¿Hay términos en idiomas diferentes mezclados?	1 = Se encuentran en todo el sistema
2. ¿Es simple el vocabulario utilizado?	3 = El vocabulario es completamente comprensible.
3. ¿Hay algún tipo de asistencia para los usuarios que hacen uso del sistema por primera vez?	1 = No existe ninguna ayuda.

4. ¿El sistema es fácil de operar para alguien que no recibió capacitación en su operación?	3 = El sistema es completamente fácil de operar.
5. ¿Se entienden la interfaz y su contenido?	3 = La interfaz es completamente entendible.
6. ¿Resulta fácil entender el resultado de una acción?	3 = Todos los resultados de las acciones son entendibles.
7. ¿El sistema informa claramente sobre los errores presentados?	3 = El sistema informa de forma adecuada todos los errores cometidos por el usuario.
8. ¿Se utiliza mensajes y textos descriptivos?	2 = La mayoría de los textos son descriptivos o fáciles de interpretar.
9. ¿Se permite al usuario personalizar la interfaz?	1 = La interfaz no es personalizable.
10. ¿Existe atajos del teclado?	3 = Todas las opciones presentan atajos por teclado.

La respuesta de algunas de las preguntas no afecta en la liberación del producto, pero sí debería tener en cuenta su corrección y mejora de la aplicación. Los resultados finales fueron 6 preguntas evaluadas de bien, una pregunta evaluada de regular y 3 preguntas evaluadas de mal.

Para el caso de la pregunta 1, se sugiere cambiar el idioma al lenguaje español o permitir al usuario seleccionar el idioma deseado.

Para el caso de la pregunta 3, se sugiere la creación de la Ayuda al usuario, brindándole un mejor entendimiento de las funcionalidades del producto.

En el caso de la pregunta 9 no tiene incidencia alguna ya que esta no está recogida dentro de las funcionalidades del sistema, esta solo proporcionaría valor agregado al producto.

Como resultado final se evalúa la prueba de usabilidad como satisfactoria teniendo como resultado el 70% de los resultados positivos.

Para la realización de la prueba de interfaz de usuario, se hizo uso de la lista de chequeo, presente en el **Anexo 5**, donde los resultados fueron positivos.

No	Evaluación	Si	No	Comentario
1.	¿Está la información libre de errores gramaticales, ortográficos y de los errores tipográficos?	x		

2.	¿Los íconos representan las acciones a las que están asociados?	x		
3.	¿Todas las funcionalidades del sistema están representadas en el menú?	x		
4.	¿Cada una de las opciones da las respuestas apropiadas?	x		
5.	¿Se tiene ausencia de errores críticos al probar todas las opciones?	x		
6.	¿Cada comando del menú tiene una secuencia hot-key?	x		

La prueba de instalación se ejecutó correctamente aunque no se contó con el manual de instalación previsto en la ejecución de esta prueba, ya que el proyecto aún no ha culminado y no tienen ese artefacto terminado para esta fase. Las pruebas de compatibilidad fueron ejecutadas sobre el sistema operativo Ubuntu 11.04 y 11.10, no se detectaron problemas, la aplicación se ejecutó correctamente, el proyecto tiene previsto hacer el producto compatible también para otros sistemas operativos aunque en esta fase este requisito no está listo.

Conclusiones del capítulo

En este capítulo se elaboró un plan de prueba donde se definió una estrategia de prueba a seguir, se seleccionaron los tipos de pruebas que van a ser aplicados de acuerdo a las características del producto y los requisitos funcionales, además se tuvo en cuenta los recursos necesarios para la ejecución de las pruebas. La estrategia de prueba se planteó siguiendo la metodología que ya se había adoptado en el proyecto, es decir, la metodología tradicional RUP, haciendo un estudio detallado de los diferentes roles que plantea la misma para el flujo de trabajo de prueba, las actividades y artefactos utilizados. Se diseñaron los casos de prueba a partir de las especificaciones de los casos de usos entregados por la analista de sistema, utilizando para ello las plantillas certificadas por Calisoft. La validación de la Estrategia de Prueba fue satisfactoria cumpliendo con los objetivos trazados, obteniendo de esta forma la principal meta del proyecto Vismedic, la cual era lograr un software con la calidad requerida,

racionalizando las pruebas a ejecutar en dependencia de las características específicas del producto Visor de Imágenes 2D.

Conclusiones

Mediante el presente trabajo se logró dar cumplimiento a su principal objetivo, el cual consistía en diseñar y aplicar una estrategia de pruebas al producto Visor de Imágenes 2D del proyecto Vismedic. Para darle inicio al trabajo de diploma se realizó un estudio de los conceptos fundamentales relacionados con la investigación, se realizó una caracterización del producto logrando obtener una mejor comprensión del problema y de las funcionalidades que requerían ser evaluadas en las pruebas dando paso a la definición de la estrategia de pruebas. Se ajustó el flujo de trabajo que propone RUP a las características específicas del producto definiendo las actividades, roles, artefactos y recursos necesarios para la correcta ejecución de las pruebas. La aplicación de la estrategia de pruebas fue satisfactoria, cada uno de los defectos encontrados fueron evaluados y recogidos en la plantilla de No Conformidades para ser entregados al proyecto para su posterior mitigación, logrando obtener un producto con mejor calidad.

Recomendaciones

Por lo planteado en dicha investigación y los resultados obtenidos se recomienda:

- Seguir aplicando la estrategia de prueba desarrollada en el proyecto Vismedic a otras iteraciones del producto.
- Aplicar las pruebas de unidad y las pruebas de integración al módulo Visor de imágenes 3D.
- Ampliar la lista de chequeo y cuestionario en las pruebas de prueba de interfaz de usuario y usabilidad para una mejor evaluación del módulo Visor de imágenes 3D.
- Utilizar la estrategia propuesta para la aplicación de las pruebas del proyecto Vismedic en otros proyectos productivos de este tipo.

Bibliografía

1- IEEE. Standard 610. Computer Dictionary. 1990.

2- Pressman, R. S. Ingeniería de Software. Un enfoque práctico. Mc Graw Hill. 2005.

3- *ISO 9001.* 2000.

4- Buades, G. Calidad en Ingeniería del Software. 2002.

5- Kruchten, P. The Rational Unified Process: An Introduction. 2000.

6- Jacobson, Booch, y Rumbaugh. El Proceso Unificado de Desarrollo de Software. 2000.

7- Juristo, N., Moreno, A. y Vegas, S. Limitations of Empirical Testing Technique Knowledge. New Jersey. 2003.

8- Cigitallabs. [En línea] 05 de 11 de 2002. [Citado el 15 de Abril del 2012.] http://:www.cigitallabs.com/resources/definitions/software_testing.html.

9- Kendall, K. Análisis y diseño de sistemas. **3ª edición**: 796- 800. 1997.

10- Software, C. d. a. D. d. I. y. G. d. Fase de Elaboración. FT Prueba (procedimientos genéricos y aplicación de algunos tipos de pruebas simples). 2005.

11- Myers, G. The art of software testing. 1979.

12- ISO 9126. 2001.

13- Carnegie Mellon University. Capability Maturity Model Integration (CMMI) Versión 1.1. 2006.

14- Davis, A. 201 Principles of Software Development. s.l.: MaGraw-Hill, 1995.

15- Avatar. Fase de transición de RUP. 2007.

16-Torres, M. C. Modelos de Evaluación de Mejora. 2006.

17- Pressman, Ingeniería de Software, un enfoque practico. DISEÑO DE CASOS DE PRUEBA. 2006.

18- Gregorio, Rodrigo Herrera. Diseño básico de un sistema ris/pacs “Imagenología médica de radiología bajo un protocolo dicom y tcp/ip”. 2007.

19- Medina, David del Río. La imagen 2D en el estándar DICOM. Universidad de Málaga. España.2009.

20- Carrasco, O. M. Un enfoque actual sobre la calidad del software. 1995.

21- RUP. Ayuda del Rational Unified Process. 2003.

22- Jr, F. A. [En línea] 21 de Abril del 2008. Jaipan Group. [Citado el 3 de Abril del 2012] http://www.yaipan.com/joomla/index.php/editorial/61

23- Dustin E. Effective Software testing. Pearson Education. 2003.

24- Zuyu J., Tsao J., Wu Y. Testing y Quality assurance for component-based software. 2003.

25- Ilene Burnstein. Practical Software Testing.Springer. NY Estados Unidos. 2005.

26- ISO/IEC 29119. Software Testing. 2008.

27- Alma It Systems. Ficha de producto AlmaVisor2D. 2008.

28- Inergis. Amaltea, Soluciones en Salud. [Citado el 21 de Mayo del 2012] http://www.inergis.com/contenidos/general.action?idsupersection=1&idselectedsection=29&selectedsection=Amaltea,%20Soluciones%20en%20Salud&typetable=informacion_general

29- 3Dicom Viewer, Visor Web DICOM [En línea] 15 de Marzo del 2011. [Citado el 24 de mayo del 2012] http://www.3dicom.es

30- IEEE Standard 829. Test Documentation. 1998.

31- EcuRed. Metodologías de desarrollo de software. [Citado el 10 de mayo del 2012] http://www.ecured.cu/index.php/Metodolog%C3%ADas_de_desarrollo_de_software

32- Figueroa, Roberth G. Metodologías tradicionales vs. Metodologías ágiles. Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación. 2007.

33- Rodríguez, Pilar. Estudio de la aplicación de metodologías ágiles para la evolución de productos software, Tesis de Máster en Tecnologías de la Información. Facultad de Informática Universidad Politécnica de Madrid. 2008.

Glosario de términos

C

CMMI: Modelo de Capacidad y Madurez.

D

Documento de No Conformidades: Documento en el que se recogen los incumplimientos de los procedimientos definidos formalmente para el software y que pueden repercutir directa o indirectamente en la satisfacción del cliente y en el cumplimiento de las metas del proyecto.

E

Estrategia de prueba: Un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba.

Especificación de requisitos de software: Documentación de requisitos fundamentales (necesarios, esenciales e indispensables) de funcionalidades, rendimiento, restricciones y atributos del software, y sus interfaces externas. Su acrónimo inglés es SRS.

I

ISO: Organización Internacional para la Estandarización.

Informática: La palabra “Informática” está compuesta por los vocablos información y automatización, y fue empleada por primera vez en el año 1962 por Philipe Dreyfus. Se refiere al conjunto de técnicas destinadas al tratamiento lógico y automático de la información, con el fin de obtener una mejor toma de decisiones.

Interfaz de usuario: Interfaz que permite la comunicación entre un usuario y un sistema, o los componentes de un sistema.

L

Lista de Chequeo: Listado de preguntas en forma de cuestionario que sirve para verificar el grado de cumplimiento de determinadas reglas establecidas a priori con un fin determinado.

P

Pruebas: Una actividad en la cual un sistema o uno de sus componentes se ejecutan en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto.

Prueba funcional: Prueba que ignora la mecánica interna de un sistema o un componente y centra la atención sólo en las salidas generadas como respuesta a determinadas entradas y condiciones de ejecución.

Plan de Prueba: Informe en el cual se recogen todas las pruebas necesarias que se llevan a cabo en un proyecto.

Plantilla: Forma de dispositivo que proporciona una separación entre la forma o estructura y el contenido.

R

Realidad Virtual: Ciencia basada en el empleo de ordenadores y otros dispositivos, cuyo fin es producir una apariencia de realidad que permita al usuario tener la sensación de estar presente en ella. Se consigue mediante la generación por ordenador de un conjunto de imágenes que son contempladas por el usuario a través de un casco provisto de un visor especial.

S

Sistema: Conjunto de procesos, hardware, software, instalaciones y personas necesarios para realizar un trabajo o cumplir un objetivo.

Software: Los programas de ordenador, procedimientos, y opcionalmente la documentación y los datos asociados que forman parte de un sistema.

V

Visor de imágenes: Aplicación informática que permite mostrar imágenes digitales guardadas es un disco local o -también- remotamente, asimismo puede manejar varios formatos de imágenes. Este tipo de software dibuja la imagen de acuerdo a las propiedades disponibles de la pantalla tales como profundidad de color y resolución de pantalla.