UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD #5

Propuesta de técnicas de prueba de software para elevar la calidad en proyectos productivos de Entornos Virtuales.

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

Autores

Annerys Aguiar Gálvez

Alain Bedoya Reyes

Tutor

Ing. Amado Espinosa Hidalgo

Asesora

Aniuska Aguilera González

Ciudad de la Habana, Cuba

Junio 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 5 de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

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

Annerys Aguiar Gálvez	**Alain Bedoya Reyes**	**Amado Espinosa Hidalgo**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Tutor:

Ingeniero Amado Espinosa Hidalgo, graduado de Ingeniería Informática en el 2004, profesor instructor con 3 años de experiencia docente y 4 años de experiencia en el desarrollo de software.

Asesores:

Licenciada Aniuska Aguilera González, graduada de Licenciatura en Contabilidad y Finanzas en el año 2004, profesora instructora con 3 años de experiencia.

AGRADECIMIENTOS.

Annerys

Agradezco la elaboración de esta tesis a todas las personas que de una forma u otra colaboraron en ello, a mis amigos, a mi familia y a toda aquella persona que en un momento determinado me dedicó alguna palabra ofreciéndome ánimos y confió en mí.

Agradezco también a la Revolución y a Fidel por darme la oportunidad de cumplir este sueño.

Alain

Gracias en primer lugar a toda mi familia...que siempre han estado presente con su todo apoyo y cariño, en especial a mis padres, mis tíos (Raúl y Héctor) y a mi abuelita (Ana) por todo lo que han hecho por mi y por inculcarme que el estudio siempre es de gran importancia en la vida, a mi prima (Liset) que ojala y el presente trabajo, le sirva de inspiración y estudie una carrera universitaria. A Carlos mi padrastro, María Antonia y Felo por ayudarme y darme aliento durante los años de la carrera. A mis amigos (Leo) y (Wilmer) que siempre han sido como hermanos para mi y nunca a faltado un consejo cuando lo he necesitado.

Agradezco a todos los que han tenido que ver de una forma u otra con la realización de este trabajo, a todos mis amigos por todos los momentos que compartimos durante los 5 años de la carrera.

Un agradecimiento especial a nuestro Comandante en Jefe y a la Revolución por darnos la oportunidad de estudiar en esta Universidad y formarnos como profesionales.

DEDICATORIA.

Annerys

A mami y papi por su inmenso amor, comprensión, apoyo y por confiar en mí; por su sacrificio, entrega y por ser mi guía en este largo camino recorrido.

A Yaniel mi novio, por esperar todos estos años lejos, por todas las alegrías vividas, por acompañarme en buenos y malos momentos.

A mis amigos por entregarme su amor y apoyo siempre.

Alain

A papi y mami por todo el cariño y apoyo que me han dado y por esmerarse tanto en ayudarme en mis estudios y mi formación como profesional.

A mi abuela (Ana) y mis tíos (Raúl y Héctor), que siempre me han dado todo su cariño y ayuda desde mis primeros años y han confiado en mí en todo momento.

Especialmente a mi abuelo (Ernesto,) que aunque no este, siempre me guió en mis primeros pasos.

RESUMEN

El presente trabajo surge como necesidad de impulsar la mejora de calidad en los proyectos productivos de la facultad 5 de la Universidad de la Ciencias Informáticas (UCI), basado en el estudio del proceso de pruebas para aplicaciones de software.

El objetivo principal de la aplicación de técnicas de pruebas es conseguir una confianza aceptable en que se detectarán los errores o defectos existentes sin necesidad de consumir una cantidad excesiva de recursos.

La prueba de software es un proceso que corre en paralelo al proceso de desarrollo de software, y que se realiza por el convencimiento de que todo sistema debe ser inspeccionado o probado con el objetivo de establecer si el nivel de calidad requerido es alcanzado y si se cumplen los requisitos establecidos por el cliente.

El presente trabajo hace referencia a las principales técnicas de pruebas existentes y a la importancia de las mismas en el nivel de calidad de los productos.

Es por ello que se expone como propósito describir las debilidades que existen en los proyectos productivos de la Facultad 5 dedicada al perfil de “Entornos Virtuales”. A partir de la experiencia obtenida y del estudio de técnicas de prueba de software, se enuncia una propuesta de cuáles deben ser las técnicas de prueba más adaptables a las características de estos proyectos y cómo deben ser planificadas en función de potenciar la estimulación de los procesos de calidad de los productos, de evitar a toda costa que el proyecto se salga del tiempo y presupuestos planeados, de reducir los esfuerzos en mantenimiento y de mejorar la satisfacción de usuarios y clientes.

Palabras Claves: calidad de software, prueba de software, plan de prueba, técnicas de prueba.

ÍNDICE

INTRODUCCIÓN

La producción de software en nuestro país es una ciencia joven la cual se ha impulsado rápidamente gracias al desarrollo de las ciencias informáticas en las universidades de nuestro país y a entidades surgidas debido al Ministerio de la Informática y de las Comunicaciones (MIC) con *"el objetivo preciso de impulsar, facilitar y ordenar el uso masivo de servicios y productos de las Tecnologías de la Información, las Comunicaciones, la Electrónica y la Automatización para satisfacer las expectativas de todas las esferas de la sociedad".*

Este desarrollo en la producción de software se ha estimulado con la fundación de la Universidad de las Ciencias Informáticas (UCI), con lo cual se espera un progreso acelerado de las Tecnologías de la Información y las Comunicaciones (TIC).

El aseguramiento de la calidad se ha convertido en una necesidad de prioridad para las organizaciones que desarrollan software porque en la medida que avanza la tecnología, es más exigente la calidad del software requerida actualmente por las empresas encargadas de medir la calidad, y para satisfacer estos requerimientos de calidad uno de los métodos más utilizados por los desarrolladores es el proceso de desarrollo de Pruebas de Software.

Algunas de las sorpresas que suelen encontrar los programadores que se inician es la enorme cantidad de tiempo y esfuerzo que requiere el desarrollo de pruebas. Se estima que la mitad del esfuerzo de desarrollo de un programa (tanto en tiempo como en gastos) lo ocupa este proceso. Si se refiere a programas que involucran vidas humanas (medicina, equipos nucleares) el costo de un proceso de pruebas puede fácilmente superar el 80 por ciento.

Actualmente la prueba de software se utiliza en todos los desarrollos de la industria, siendo inconcebible que un cliente reciba un sistema software que no haya sido probado. Pese a su enorme impacto en el coste de desarrollo, es una de las líneas de trabajo que muchos programadores aún consideran clasificable como un arte y, por tanto, como difícil de conceptualizar.

En el mundo existen múltiples metodologías para el desarrollo de software, pero sin duda alguna todas tienen implícito el proceso de desarrollo de pruebas, el cual está centrado en el objetivo de encontrar defectos a un software; puede ser por razones de depuración o de aceptación del mismo.

El principal reto de la industria del software en la actualidad es transformar el desarrollo de productos y soluciones en un proceso acelerado y disciplinado que consiga la alineación de equipos, tecnologías y procesos, y maximice el valor añadido que las empresas obtienen del software. Este debe ser, además, uno de los principales retos de la industria del software naciente en nuestro país; por lo que la UCI debe imponerse este reto.

Durante los casi 5 años de vida de la UCI se han implementado numerosas aplicaciones que han dado una visión muy clara de que es posible llevar a cabo con gran éxito el objetivo por el cual fue creada nuestra universidad, ejemplo de ello es la creación del simulador de auto y el evaluador teórico para conducción en la Facultad 5; además se ha trabajado en direcciones como juegos y otras aplicaciones.

A pesar de esto, en ocasiones se han presentado irregularidades entorno a las fechas de entrega de los productos y han tenido que proporcionarse trabajos de mantenimiento y soporte a usuarios insatisfechos.

En la actualidad la Facultad 5, enfocada en el perfil de desarrollo de Entornos Virtuales, no se ha establecido una cultura en cuanto a la utilización de pruebas de software para mejorar la calidad de los productos y no se cuenta con un estudio que resuma un conjunto de técnicas de prueba que se adapten a las características de los productos desarrollados y que garanticen minimizar las no conformidades surgidas debido a la ausencia de un proceso organizado de prueba. La puesta en práctica de las pruebas de software sería un paso esencial, debido a que cuanto más pronto se apliquen mecanismos de prueba en el proceso de desarrollo, más rápido se detectarán los problemas originados, más fácilmente podrá evitarse que el proyecto se salga del tiempo y presupuesto planeado, y además se podrían evitar los costos de corregir errores en etapas de mantenimiento, estimados en el mundo entre 60 y 100 veces más que el costo de corregirlos en etapas tempranas de producción.

Por lo que el presente trabajo orienta su **problema científico** a: ¿Cómo propiciar la realización de pruebas de software con el fin de elevar la calidad del proceso de desarrollo en los proyectos de Entornos Virtuales en la Universidad de las Ciencias Informáticas?

La calidad dentro del proceso de desarrollo de software define las características tanto funcionales como no funcionales que describen al producto, tales como el correcto funcionamiento del código hasta el estilo creativo y comprensible de la interfaz de usuario.

La elaboración de un software con calidad implica la utilización de metodologías o estándares de desarrollo y pruebas de software que permitan elevar la productividad, tanto para la labor de desarrollo como para el control de la calidad. Esto da paso a que el presente trabajo enmarque específicamente el **objeto de estudio** en: El desarrollo de pruebas de software, por lo cual se define el **campo de acción** en: Las técnicas y estrategias existentes para el desarrollo de pruebas de software.

El presente trabajo de diploma tiene como **objetivo**: Determinar las técnicas existentes sobre el desarrollo de pruebas de software para su aplicación en la Facultad 5 de la Universidad de las Ciencias Informáticas.

Se proponen las siguientes **tareas de investigación:**

- Estudiar de las teorías sobre la aplicación y control de pruebas de software, así como también de las técnicas y metodologías existentes.
- Caracterizar las técnicas y estrategias más significativas de desarrollo de pruebas de software existentes en el mundo.
- Identificar el desarrollo de pruebas de software en los proyectos productivos de la Facultad 5 de la Universidad de las Ciencias Informáticas.
- Confeccionar un plan de pruebas de software que sirva de guía a los proyectos de la Facultad 5.

El presente documento consta de tres capítulos:

En el **Capítulo 1** se describe la fundamentación teórica y estado actual de las pruebas de software, haciendo referencia a conceptos, al proceso de pruebas de software, a los principios de las pruebas y al desarrollo de pruebas en metodologías tales como RUP y XP.

En el **Capítulo 2** se realiza una caracterización de técnicas y estrategias para el desarrollo de pruebas documentándose, aspectos fundamentales relacionados con métodos y técnicas para el desarrollo exitoso de las mismas.

En el **Capítulo 3** se describe una caracterización del desarrollo de pruebas en proyectos productivos en los entornos de la Facultad 5 y se proporciona una propuesta de técnicas de prueba a ser utilizadas en estos proyectos. Además, se recomienda un plan de prueba para los proyectos de Entornos Virtuales y la aplicación de estimaciones del esfuerzo de la prueba.

CAPÍTULO I. Fundamentación teórica y actualidad de las pruebas de software.

1.1 INTRODUCCIÓN

El objetivo fundamental de este capítulo es detallar conceptualmente los temas a los que se hará referencia durante el avance del trabajo, visualizar las diferentes tendencias actuales referentes a las pruebas de software y a metodologías que utilizan tales prácticas de pruebas en su desarrollo.

Se conceptualizará el tema de software, ingeniería de software y calidad de software para una mejor comprensión al ser planteado alguno de estos temas en la evolución del trabajo.

Se tratará el tema de las pruebas de software, abordando conceptos fundamentales, tales como prueba de software, caso de prueba, estrategia de prueba, defecto, fallo y error. Se hará una referencia a las tendencias actuales sobre estos temas reflejados en el contexto nacional e internacional y reflejado en algunas de las metodologías más significativas que existen, tales como Rational Unified Process (RUP) y Extreme Programming (XP).

1.2 CONCEPTOS FUNDAMENTALES

¿Qué es un Software?

Software es el elemento lógico del ordenador. Constituye el conjunto de programas que puede ejecutar el hardware para la realización de las tareas de computación a las que se destina. Es el conjunto de instrucciones que cuando se ejecutan facilitan la función y el rendimiento que permite la utilización del equipo.

También conocido como soporte lógico, comprende todo tipo de programas, utilidades, aplicaciones, sistemas operativos, que hacen posible que el usuario pueda manejar adecuadamente la información y los documentos que representan la operación y uso de los programas al trabajar con la máquina.

El término está totalmente integrado en nuestro idioma ya que, al igual que sucede con hardware, no ha habido nadie capaz de encontrar una traducción capaz de englobar el concepto en una sola palabra.

¿Qué es Ingeniería de Software?

La **Ingeniería de Software** es la rama de la ingeniería mediante la cual aplicando un conjunto de técnicas y/o metodologías, herramientas y conocimientos informáticos es posible la creación y control organizado y eficiente, de aplicaciones de software.

El IEEE (Institute of Electrical and Electronics Engineers) define la ingeniería de software como la rama de la ingeniería que aplica los principios de la ciencia de computación y las matemáticas para lograr soluciones eficaces económicamente a los problemas de desarrollo de software.

¿Qué es Calidad del Software?

La calidad del software es el grado en que el software satisface los requerimientos funcionales y no funcionales definidos por el cliente y que está fuertemente vinculada a las cualidades que caracterizan y que determinan la utilidad y existencia del software con los estándares de desarrollos explícitos y documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente.

La calidad de software se fundamenta en la utilización de métodos de desarrollo adecuados sobre herramientas que den soporte a estos. "*El proceso se basa en la práctica de revisiones de los productos obtenidos para garantizar la adecuación de la construcción, y en la realización de pruebas tendentes a la búsqueda y depuración de errores. El resultado de dichas revisiones y pruebas es la elaboración de un informe que indique el grado de calidad del producto obtenido, los problemas detectados y las soluciones propuestas*" (PRESSMAN 2005).

¿Qué es una Prueba de Software?

La **Prueba o Testeo de Software**, es un procedimiento llevado a cabo para identificar posibles fallos de implementación, calidad, o usabilidad de un programa. Básicamente es una fase en el desarrollo de software cuyo objetivo es probar las funcionalidades de la aplicación construida.

La prueba de software es un proceso que corre en paralelo al proceso de desarrollo de software, y que se realiza por el convencimiento de que todo sistema debe ser inspeccionado o probado con el objetivo de establecer si el nivel de calidad requerido es alcanzado.

Es un elemento que a menudo se refiere como verificación y validación. En (PRESSMAN 2005) se plantea que "*la **verificación** se refiere al conjunto de actividades que aseguran que el software*

*implementa correctamente una función específica. La **validación** se refiere a un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente"*.

¿Qué es un caso de prueba?

Un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados de un caso desarrollado para cumplir un objetivo en particular ó una función esperada.

En la Ingeniería del software, los casos de prueba o Test Case son un conjunto de condiciones o variables bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio.

Se pueden realizar muchos casos de prueba para determinar que un requisito es completamente satisfactorio. Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber al menos un caso de prueba para cada requisito a menos que un requisito tenga requisitos secundarios. En ese caso, cada requisito secundario deberá tener por lo menos un caso de prueba. Algunas metodologías como RUP recomiendan crear por lo menos dos casos de prueba para cada requisito.

¿Qué es una técnica de prueba?

Las técnicas de prueba "*facilitan una guía sistemática para diseñar pruebas que: (1) comprueben la lógica interna de los componentes software, y (2) verifiquen los dominios de entradas y salidas del programa para descubrir errores en la funcionalidad, el comportamiento y rendimiento*". (PRESSMAN 2005)

La técnica de prueba es un procedimiento que conlleva a la detección de errores en una aplicación software y que está basado en la elaboración de casos de prueba con diferentes fines de detección.

¿Qué es una estrategia de prueba?

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

La estrategia de prueba de software se constituye de 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 cuenta cuánto esfuerzo y recursos se van a requerir.

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

¿Qué es un fallo?

Un fallo ocurre cuando un programa no se comporta de la forma adecuada. El IEEE define un fallo como "la incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados".

¿Qué es un defecto?

Un defecto existe en el código si se produce un fallo, no hay defecto si el sistema no puede fallar*. "Un defecto en un sistema o componente del sistema hace que el sistema no pueda realizar una función requerida"* (BURNSTEIN 2003).

El IEEE plantea que un defecto puede definirse como *"un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa"*. Un defecto puede ser, además, una característica no conforme con los requerimientos de la especificación y que puede afectar de forma negativa a la calidad del producto o servicio que se ofrece.

¿Qué es un error?

Un error es una acción cometida inconscientemente por un programador que puede dar lugar a que el software contenga un defecto, haciéndolo fallar.

Los errores tienden a introducirse con mayor probabilidad en el programa en que se trabaja cuando se diseñan e implementan funciones o condiciones complejas o que se encuentran fuera de lo normal.

¿Que es una Metodología o un Proceso 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 aplicaciones software.

Un proceso de desarrollo va indicando por pasos todas las acciones a realizar para lograr el producto deseado; indicando qué personas deben participar en el desarrollo de las actividades y el rol que debe realizar.

Además la metodología debe detectar y corregir los errores cuanto antes. Uno de los problemas más frecuentes y costosos es el aplazamiento de la detección y corrección de errores en las etapas finales del proyecto. Cuanto más tarde sea detectado el error más caro será corregirlo. Por lo tanto cada fase del proceso de desarrollo de software deberá incluir una actividad de verificación y/o validación explícita.

1.3 ACTUALIDAD DE LAS PRUEBAS DE SOFTWARE.

La creación de software tiene sólo algunas décadas de desarrollo; la industria del software en nuestro país aún está definiendo la manera más apropiada de desarrollarse para brindar todo su potencial al desarrollo de la economía. Para la Universidad de las Ciencias Informáticas considerar las propuestas basadas en la experiencia de los demás investigadores, permitirá evolucionar en la industria del software con mayor velocidad; por ello en esta sección del documento se tendrán en cuenta las principales tendencias en el desarrollo de las pruebas de software planteadas por prestigiosos especialistas en el tema.

Al inicio del desarrollo de la industria del software, las pruebas se consideraban sólo una actividad que realizaba el programador para encontrar defectos en sus productos; con el paso de los años se ha determinado la importancia que tienen para garantizar el tiempo, el costo y la calidad del producto, de tal forma que actualmente las pruebas definen un procedimiento mediante el cual se evalúa la funcionalidad del software respecto a los requerimientos establecidos por los usuarios.

¿Cuál es la nueva tendencia en las pruebas establecida en (MYERS 2004)? Iniciarlas antes, dentro del proyecto, y capacitar a especialistas responsables de esta actividad. El primer punto quiere decir que actualmente las especificaciones de pruebas se realizan al mismo tiempo que el diseño del software; la propuesta es iniciar el análisis de la prueba junto al análisis del software. El segundo punto habla de

crear conciencia acerca de la importancia de las pruebas y tener un equipo de personas dedicadas a esta actividad que puedan integrarse a un proyecto y sean responsables de su calidad.

Los objetivos actuales de las pruebas no sólo tienen que ver con corregir errores, sino con prevenirlos influyendo y controlando el diseño y desarrollo del software. Las pruebas deben ser empleadas como modelos de los requerimientos de la aplicación que se ha de construir; por tanto, en las especificaciones de software deben incluirse especificaciones de pruebas, ambas deberán revisarse de conjunto, y en esta revisión deberá participar un especialista en pruebas.

Las pruebas se deben realizar, en el entorno en el que se utilizará el sistema, lo que incluye el personal que lo maneja, pero además la etapa de pruebas no debe ser posterior a la confección de un programa, tiene que ser paralela a la programación.

Por los años de 1970, una regla muy conocida era, que en un proyecto de programación típico, el 50 por ciento del tiempo empleado en el mismo y más del 50 por ciento del costo total, era utilizado en las pruebas o en desarrollo del sistema.

En la actualidad la historia se repite. Existen nuevos sistemas de desarrollo, lenguajes y programadores que están acostumbrados a desarrollar más rápidamente. Pero hoy por hoy las pruebas todavía juegan un rol importante en cualquier proyecto de desarrollo de software.

Hoy en día, se podría decir que las pruebas de software han llegado a ser más fáciles y más difíciles al mismo tiempo. Las pruebas de software son más difíciles por la gran cantidad de lenguajes de programación, sistemas operativos y las plataformas de hardware en las que se encuentran envueltas. Las pruebas de software son más fáciles, en algunos casos, por que los sistemas operativos y software, son mucho más sofisticados y proveen intrínsicamente buenas rutinas de pruebas, que pueden ser incorporadas en las aplicaciones, sin la necesidad que el programador las desarrolle desde cero.(MYERS 2004)

La realización de pruebas de software es un proceso centrado en el objetivo de encontrar defectos a un software; puede ser por razones de depuración al software o de aceptación del mismo. A pesar que generalmente en el mundo se está de acuerdo en que es mejor prevenir defectos que encontrarlos y corregirlos, la realidad es que actualmente el ser humano es incapaz de producir sistemas libres de defectos. La prueba es un elemento esencial en el desarrollo de software, el cual ayuda a mejorar la calidad de este.

Las pruebas no mejoran directamente la calidad del sistema, pero sí lo hacen indirectamente, previendo un panorama claro de las debilidades observadas del sistema y de los riegos asociados para la organización. Esto permite a las administraciones tomar decisiones respecto a la asignación de recursos para mejorar la calidad del sistema.

Para lograr estos objetivos, todo proceso de prueba contiene actividades para planear lo que se necesita, especificando qué debe ser probado. Hay una regla universal, la cual indica que es imposible encontrar todos los defectos y que nunca hay suficiente tiempo, personal o dinero para probar todo. Se deben tomar decisiones sabias de cómo distribuir los recursos disponibles.

1.3.1 El proceso de pruebas de software tratado en el contexto nacional e internacional.

Actualmente, en los países desarrollados existe una industria de prueba de software fuertemente establecida y organizada, constituida entre otras cosas por una buena cantidad de profesionales especializados, proveedores de herramientas, congresos, publicaciones periódicas, y múltiples alternativas de capacitación y certificación para los ingenieros interesados en este campo (ejemplo de ello es la empresa e-Quallity). La industria de software de esos países puede utilizar esa especialización debido al desarrollo que presentan.

Nuestro país, actualmente, está intentando insertarse en este mercado globalizado y altamente competitivo de las tecnologías de información y las comunicaciones. Aún no se ha logrado una industria de software suficientemente grande y especializada con una oferta de servicios de alta calidad y valor agregado; pero si se quiere alcanzar este perfeccionamiento se debe tener en cuenta que sin duda alguna la prueba de software es hoy más solicitada que hace algunos años debido a las demandas de calidad del cliente.

Aunque, en nuestro país, la prueba de software no sea un servicio ofrecido, como en otros, de manera integral, especializada e independiente, la calidad del software es un reto por el que se está trabajando, pues se tiene bien definido que para insertarse en el ámbito global de la industria de software la calidad no representa un valor agregado, sino más bien una exigencia obligatoria para permanecer en el mercado internacional.

Algunas de las consecuencias que se obtienen de un producto entregado con baja calidad y que dan razón de ser a la prueba de software son:

- Las empresas beneficiadas no pagan por un producto hasta que no se alcance el nivel de calidad esperado.
- Liberar un producto inmaduro trae como consecuencia una labor penosa y costosa de soporte a usuarios insatisfechos, además del gasto para proveer a estos usuarios de una nueva versión con los problemas resueltos.
- La mala calidad de un producto no tarda en verse reflejada en una baja en las ventas.
- La insatisfacción del cliente trae como consecuencia pérdida de imagen de la empresa desarrolladora, así como una baja de la credibilidad en la misma.
- En sistemas críticos la falla del sistema puede traer como consecuencia pérdidas de vidas humanas y/o monetarias, con repercusiones económicas muy severas.

1.3.2 Principios de la prueba de software enunciados por Myers.

En un principio del desarrollo de software, la prueba se llevaba a cabo para ganar confianza en el sistema, o sea para comprobar sí “corría bien”. Los trabajos de Glenford J. Myers en los años 70 comenzaron a cambiar este criterio, ya que desde su punto de vista, el objetivo de la prueba es demostrar que el sistema “no satisface los requerimientos”.

Estos son algunos de los principios que Myers enunció en su libro (MYERS 2004) al respecto de la prueba de software:

Principio 1: Una parte necesaria de un caso de prueba es una definición de la salida o el resultado esperado.

Este principio obvio es uno de los errores ocurridos con más frecuencia en la prueba del software. Si el resultado previsto de un caso de prueba no se ha predefinido, las probabilidades son que un resultado creíble pero erróneo será interpretado como un resultado correcto debido al fenómeno de que "el ojo ve lo que desea ver".

En otras palabras, a pesar de la definición destructiva adecuada de la prueba, todavía hay un deseo subconsciente de ver el resultado correcto. Una forma de combatir esto es animar un examen detallado de toda la salida explicando, por adelantado, la salida prevista del programa. Por lo tanto, un caso de prueba debe consistir en dos componentes:
1. Una descripción de los datos de entrada del programa.

2. Una descripción exacta de la salida correcta del programa para la que se determinaron los datos de entrada.

Principio 2: Un programador debe evitar probar el código de su propio programa.

Cualquier escritor sabe, o debe saber, que no es lo más correcto procurar corregir o corregir su propio trabajo. Usted sabe lo que se supone que dice una parte y puede no reconocer esa parte cuando lo dice de otra manera, y realmente no desea encontrar errores en su propio trabajo. Esto es igualmente aplicable a los programadores de software.

Otro problema se presenta con un cambio en la forma de ver un proyecto de software. Después de que un programador haya diseñado y cifrado de forma constructiva un programa, es extremadamente difícil para él cambiar repentinamente la perspectiva para ver el programa de forma destructiva. La mayoría de los programadores no pueden probar sus propios programas eficazmente, porque no pueden abstraerse para cambiar los mecanismos mentales al intentar exponer los errores. Además, un programador puede subconscientemente evitar el hallazgo de errores por el miedo a la retribución que hará el supervisor, el cliente, o el dueño del programa o sistema que se desarrolla.

Además de estas condiciones psicológicas, hay un segundo problema significativo: El programa puede contener errores debido a una equivocación del programador en la declaración o la especificación del problema. Si éste es el caso, es probable que el programador incluya el mismo error en las pruebas de su propio programa.

Esto no significa que es imposible que un programador pruebe su propio programa, sino que algo implica que la prueba es más eficaz y acertada si alguna otra persona la ejecuta. Observe que esta cuestión no se aplica para eliminar errores; el eliminar errores es realizado más eficientemente por el programador original.

Principio 3: La organización desarrolladora debiera evitar probar sus propios sistemas.

La polémica en este principio es similar a la anterior. Una organización de un proyecto de programación es, en muchos sentidos, una organización viva, con problemas psicológicos similares a los de programadores individuales. Hay una barrera mental que dificulta que una persona encuentre defectos en aquello que construyó con esmero y dedicación. Por lo tanto, es difícil que una organización de programación sea objetiva en la prueba de sus propios programas, porque el proceso

de prueba, si está acertado con la definición apropiada, puede verse como la disminución de la probabilidad de resolver el horario y los objetivos de coste.

Una vez más esto no dice que es imposible que una organización de programación encuentre algunos de sus errores, porque las organizaciones logran esto con cierto grado de éxito, sin embargo, algo implica que es más económico que la prueba sea realizada por una parte objetiva, independiente.

Principio 4: Examinar completamente los resultados de cada prueba.

Este es probablemente el principio más obvio, pero es otra vez algo que se pasa por alto a menudo.

Se han visto numerosos experimentos que demuestran que muchos fallan a la hora de detectar ciertos errores, incluso cuando los indicios de esos errores eran claramente observables en los listados de salida. Visto de otra forma, los errores que se encuentran en las últimas pruebas no se ven frecuentemente en los resultados de pruebas anteriores.

Principio 5: Los casos de prueba deben ser escritos no sólo para las condiciones de entrada que son inválidas e inesperadas, sino también para las que sean válidas y esperadas.

Hay una tendencia natural al probar un programa en concentrarse en entrar condiciones válidas y esperadas, y en el abandono de las condiciones inválidas e inesperadas. Muchos de los errores que se descubren repentinamente en programas de la producción se presentan cuando el programa se utiliza de una nueva o inesperada manera.

Por lo tanto, los casos de prueba que representan condiciones inesperadas e inválidas de entradas parecen tener una producción más alta de detección de errores que los casos que prueban las condiciones válidas.

Principio 6: Examinar un programa para ver si no hace lo que se supone que debe hacer es solamente la mitad de la batalla; la otra mitad es ver si el programa hace lo que no se supone que debe hacer.

Esto es un corolario al principio anterior. Los programas se deben examinar para controlar los efectos secundarios indeseados.

Principio 7: Evitar los casos desechables y triviales de prueba a menos que el programa sea verdaderamente un programa sencillo.

Este problema se ve muy a menudo con los sistemas interactivos para probar programas. Una práctica común es colocar un término e inventar casos de prueba en marcha, y después enviar estos casos de prueba terminando con el programa.

El problema principal es que los casos de prueba representan una inversión valiosa que, en este ambiente, desaparece después de que se haya terminado la prueba.

Siempre que el programa tenga que ser probado otra vez (por ejemplo, después de corregir un error o de llevar a cabo una mejora) las pruebas deben ser rediseñadas, puesto que este rediseño requiere una cantidad considerable de trabajo las personas tienden a evitarlo.

Por lo tanto, la contra-prueba del programa es raramente tan rigurosa como la prueba original, significando eso si la modificación causa que una parte previamente funcional del programa falle, este error va a ser a menudo desapercibido. Los casos y el funcionamiento de pruebas después de cambios realizados a otros componentes del programa son conocidos como pruebas de regresión.

Principio 8: No debe planearse un esfuerzo de prueba bajo el supuesto de que no se encontrarán defectos.

Esto es a menudo el error que realizan los líderes de proyecto y es una señal del uso incorrecto de la definición de prueba: el supuesto de que la prueba es el proceso de demostrar que el programa funciona correctamente. Una vez más, la definición de prueba es el proceso de ejecutar un programa con la intención de encontrar errores.

Principio 9: La probabilidad de existencia de más errores en una sección de un programa es proporcional al número de los errores encontrados ya en esa sección.

Los errores tienden a presentarse en racimos dado que en el programa típico, algunas secciones parecen ser mucho más propensas a los errores que otras, aunque nadie ha dado una buena explicación de porqué ocurre esto. El fenómeno es útil pues nos da la visión en el proceso de prueba.

Si una sección particular de un programa es mucho más propensa a los errores que otras, entonces este fenómeno quiere decir que, en términos del rendimiento en nuestra inversión de prueba, los

esfuerzos de prueba adicionales están enfocados lo mejor posible contra esta sección propensa a error.

Principio 10: La prueba es una tarea extremadamente creativa e intelectualmente desafiadora.

Es muy probable que la creatividad requerida en probar un programa grande exceda la creatividad requerida en diseñar ese programa. Se ha visto ya que es imposible probar un programa lo suficiente para garantizar la ausencia de todos los errores, por ello es considerable encauzar a algunos de los mejores programadores en el campo de las pruebas de software.

1.3.3 Utilización del trabajo con pruebas en metodologías tales como RUP y XP.

En los últimos años se han desarrollado dos corrientes en lo referente a los procesos o metodologías de desarrollo de software, los llamados métodos pesados y los métodos ligeros.

La diferencia fundamental entre ambos es que mientras los métodos pesados intentan conseguir el objetivo común por medio de orden y documentación (ejemplo de ello es la metodología RUP); los métodos ligeros (también llamados ágiles) tratan de mejorar la calidad del software por medio de una comunicación directa e inmediata entre las personas que intervienen en el proceso (ejemplo de ello es la metodología XP).

1.3.3.1 RUP

Es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, no tan solo de software.

Un proceso realizado siguiendo RUP se divide en cuatro fases que se comportan de la siguiente forma:

- Inicio (puesta en marcha): El desarrollo del prototipo exploratorio de demostración; no requiere la elaboración de pruebas.
- Elaboración (definición, análisis y diseño): Probar los componentes ejecutables que se han implementado y que deben corresponderse con la arquitectura básica de la aplicación.
- Construcción (implementación y prueba): Desarrollar los casos de prueba y procedimientos de prueba para hacerlos.
- Transición (fin del proyecto y puesta en producción): El producto en su entorno de operación por lo que es probado por usuarios reales.

En cada fase de RUP se realizarán una o varias iteraciones y dentro de cada una de ellas se seguirá un modelo de cascada para los flujos de trabajo que requieren cada una de las fases anteriormente citadas.

RUP define nueve flujos de trabajo a utilizar en cada fase. Estos son:
- Modelado del Negocio
- Captura de Requisitos
- Análisis y Diseño
- Implementación

- Prueba

- Distribución
-Gestión de la Configuración y Cambios
- Gestión del Proyecto
- Gestión del Entorno

En RUP el proceso de prueba tiene una particular relación con las restantes disciplinas, destacada de la siguiente manera:

Figura 1.1: La prueba de software en el ámbito de la metodología RUP.

- La disciplina Requerimientos, captura los requisitos para el producto de software, el cual es una de las entradas principales para identificar qué pruebas se van a ejecutar y es la base de una ejecución de pruebas debido a la captura clara y eficiente de estos.
- La disciplina Análisis y Diseño, determina el diseño apropiado para el software, la cual es otra de las entradas principales para identificar qué pruebas se van a ejecutar, con el objetivo de establecer el diseño apropiado del software.
- La disciplina de Implementación produce versiones operacionales del sistema (builds) que son validados por la prueba. Dentro de una iteración, múltiples builds serán probados.
- La disciplina Despliegue entrega el producto de software completo al usuario final. Antes el software es validado en el proceso de prueba, aunque las pruebas de aceptación y las pruebas betas son ejecutadas como parte del despliegue.
- La Gestión de proyecto planifica el proyecto y las iteraciones, el de Plan de Iteración es un artefacto que es una importante entrada usada cuando se va a definir la misión de evaluación para la prueba.
- La disciplina Gestión de la Configuración y Cambios controla los cambios dentro del proyecto. La prueba verifica que cada cambio ha sido completado apropiadamente.

RUP es un proceso muy documentado alrededor de las pruebas y su adecuada aplicación al software en construcción principalmente a partir de casos de pruebas.

Un caso de prueba en RUP es un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular. Son un conjunto de condiciones o variables bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio. Se pueden realizar muchos casos de prueba para determinar que un requisito es completamente satisfactorio. Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber al menos un caso de prueba para cada requisito.

El ciclo de vida de prueba en RUP se puede definir de la siguiente manera:

El software es refinado a través de iteraciones en el ciclo de vida. El ciclo de vida de prueba se beneficia siguiendo un proceso iterativo equivalente. En cada iteración el equipo de desarrollo produce uno o más builds, cada build es un candidato potencial para probar. Los objetivos del equipo de desarrollo difieren de una iteración a otra. El equipo de prueba estructura su prueba de acuerdo a los objetivos de la iteración.

En una iteración n, usted puede implementar y ejecutar determinadas pruebas. Algunas de estas pruebas son conservadas y acumuladas, las cuales son usadas para pruebas de regresión. Cualquiera de las pruebas desarrolladas en la iteración n son candidatas para pruebas de n+1, cuando hay pruebas que son repetidas varias veces, vale la pena considerar automatizarlas.

La prueba se alinea con la iteración que el resto del equipo de desarrollo sigue. Generalmente, cada iteración contiene al menos un ciclo de prueba. La iteración comienza con una investigación por el equipo de prueba, quienes negocian con el líder de proyecto, y otros stakeholders en lo que se refiere a las pruebas más útiles para emprender en las iteraciones.

Un subconjunto de los miembros del equipo de prueba puede estar investigando nuevas técnicas de pruebas. Este esfuerzo intenta probar qué técnicas son factibles de usar. Así que el equipo de prueba puede contar con estas técnicas en iteraciones posteriores.

Para cada build construido por el equipo de desarrollo, se realizan los cuatro siguientes flujos de trabajo de la disciplina de prueba. Se valida que el build es lo suficiente estable para comenzar la evaluación. Se realiza la evaluación. A medida que se está probando y evaluando, se entrega un resultado útil de la evaluación de las pruebas para los stakeholder; este resultado está determinado en términos de la misión de evaluación.

Los problemas encontrados durante una iteración pueden ser solucionados dentro de la misma iteración o pospuesto hasta próximas iteraciones. Una de las tareas principales para el equipo de prueba y los administradores de proyecto es medir cómo finalizó la iteración, verificando que los objetivos de la misma expuestos en el Plan de Iteración están cumplidos.

El propósito de cada actividad es el siguiente:

Actividad: Definir la misión de la evaluación. Su función fundamental es identificar el método apropiado de la prueba para la iteración y estar de acuerdo con los stakeholder de las metas correspondientes que dirigirán las pruebas.

A partir del Plan de Iteración y el Plan de Desarrollo de Software se define el Plan de Prueba, se identifica los objetivos y la estrategia de pruebas. Se define además cómo monitorear y evaluar el progreso de las pruebas. Para cada iteración, se enfoca este flujo de trabajo en:

- ✓ Identificar los objetivos y los entregables de las pruebas.

- ✓ Identificar una buena estrategia de utilización de los recursos.
- ✓ Definir el alcance y la frontera de las pruebas.
- ✓ Esbozar la estrategia de prueba a usar.
- ✓ Definir cómo se va a monitorear y valorar el progreso de las pruebas.

Actividades desarrolladas:

- ✓ Identificar los motivadores de pruebas.
- ✓ Identificar los objetivos de las pruebas y las ideas de pruebas.
- ✓ Definir las necesidades de traceabilidad y valoración.
- ✓ Definir la estrategia de prueba.
- ✓ Llegar a un acuerdo de la misión.

Artefactos claves elaborados en esta actividad

- ✓ Plan de prueba.
- ✓ Estrategia de prueba.

Actividad: Verificar el enfoque de prueba. Su función fundamental es demostrar que diferentes técnicas facilitarían la prueba planificada, es verificar demostrando que el enfoque trabajará, producirá resultados precisos y es apropiado para los recursos disponibles.

El objetivo es lograr un entendimiento de las restricciones y limitaciones de cada técnica al aplicarlas en un contexto del proyecto dado: encontrar una solución de implementación apropiada para cada técnica o encontrar técnicas alternativas que puedan usarse. Esto ayuda a mitigar el riesgo de descubrir muy tarde técnicas que no son factibles aplicarlas en el proyecto.

Para cada iteración, enfocar el trabajo para:

- ✓ Verificación temprana de qué estrategia de prueba trabajará y producirá resultado de valor.
- ✓ Establecimiento de la infraestructura básica disponible y de apoyo a la estrategia de prueba.
- ✓ Obtener un compromiso del equipo de desarrollo de software que reúna los requerimientos necesarios para lograr la estrategia de prueba.
- ✓ Identificar el alcance, frontera, limitaciones y restricciones de cada técnica.

Actividades desarrolladas:

- ✓ Definir la configuración del ambiente de prueba.
- ✓ Identificar los mecanismos a probar.

- ✓ Definir elementos a ser probados.
- ✓ Definir detalles de prueba.
- ✓ Implementar prueba.

Artefactos claves elaborados en esta actividad:

- ✓ Arquitectura de automatización de la prueba.
- ✓ Configuración del entorno de la prueba.
- ✓ Caso de prueba.
- ✓ Especificación de interfaz de la prueba.

Actividad: Validar la estabilidad de build. Su función fundamental es validar que el build es suficientemente estable para la prueba detallada y para comenzar la evaluación.

Es importante definir los detalles de la prueba, implementarla, ejecutarla, analizar los fallos y determinar los resultados de la prueba.

Para cada Build que va ser probado, enfocar el trabajo a:

- ✓ Valorar la estabilidad y probar el build.
- ✓ Lograr una comprensión inicial o confirmación de las expectativas del trabajo de desarrollo entregado en el build.
- ✓ Tomar la decisión de aceptar el build como adecuado para usarlo (guiado por la misión de evaluación) en más pruebas.

Actividades desarrolladas:

- ✓ Definir los detalles a probar.
- ✓ Ejecutar los casos de prueba.
- ✓ Implementar prueba.
- ✓ Analizar fallos.
- ✓ Determinar los resultados de las pruebas.
- ✓ Valorar y abogar por la calidad.

Artefactos claves elaborados en esta actividad:

- ✓ Resumen de evaluación de la prueba
- ✓ Resultado de la prueba.

Actividad Mejorar la calidad de las pruebas: Su función fundamental es mantener y mejorar la calidad de las pruebas. Esto es especialmente importante si la intención es volver a usar las ventajas desarrolladas en el actual ciclo de prueba en ciclos de pruebas posteriores.

RUP clasifica las pruebas de software en distintos tipos que serán ampliados para una mejor comprensión en el capítulo 2. Algunos de estos son:

Prueba de Unidad

Se plantea a pequeña escala, y consiste en ir probando uno a uno los diferentes módulos que constituyen una aplicación. [Para mayor información ver capítulo 2]

Prueba de Integración

Se centran en probar la coherencia semántica entre los diferentes módulos, tanto de semántica estática (se importan los módulos adecuados; se llama correctamente a los procedimientos proporcionados por cada módulo), como de semántica dinámica (un módulo recibe de otro lo que esperaba). Normalmente estas pruebas se van realizando por etapas, englobando progresivamente más y más módulos en cada prueba.

Las pruebas de integración se pueden empezar en cuanto se tienen unos pocos módulos, aunque no terminarán hasta disponer de la totalidad. En un diseño descendente se empieza a probar por los módulos más generales; mientras que en un diseño ascendente se empieza a probar por los módulos de base. [Para mayor información ver capítulo 2]

Prueba de sistema

El objetivo de las pruebas de sistema es la comprobación del sistema global, realizándose para comparar el sistema con sus objetivos originales.

- ✓ Se busca demostrar que objetivos no se cumplen.
- ✓ No es posible realizarlo si no existen objetivos documentados.
- ✓ Se usan como base los objetivos originales y la documentación del usuario para ampliarla.
- ✓ No existe un método, se dan lineamientos, a la hora de preparar los casos de prueba.

[Para mayor información ver capítulo 2]

Pruebas de regresión

El objetivo es comprobar que los cambios sobre un componente, no generan errores adicionales en otros componentes no modificados. El conjunto de pruebas de regresión contiene tres clases diferentes de casos de prueba:

- ✓ Una muestra representativa de pruebas que ejercite todas las funciones del software.
- ✓ Pruebas adicionales que se centran en las funciones del software que se van a ver probablemente afectadas por el cambio.
- ✓ Pruebas que se centran en los componentes del software que han cambiado.

No es práctico ni eficiente volver a ejecutar cada prueba de cada función del programa después de un cambio.

Pruebas de aceptación

Son las que se plantea el cliente final, que decide qué pruebas va a aplicarle al producto antes de darlo por bueno y pagarlo. El objetivo de la persona que prueba es encontrar los fallos lo antes posible, en todo caso antes de pagarlo y antes de poner el programa en producción.

RUP plantea diferentes Métodos de Pruebas que serán ampliados para su mejor comprensión en el capítulo 2 como son:

- ➢ **Pruebas de Caja Negra**

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

- ➢ **Pruebas de Caja Blanca**

Se comprueban los caminos lógicos del software proponiendo casos de pruebas que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

1.3.3.2 XP

Como toda metodología ágil intenta reducir la complejidad del software a través de la estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y la actividad actual, por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción.

XP define UserStories como base de software a desarrollar. Estas son historias que escribe el cliente y que describen escenarios sobre el funcionamiento del software. A partir de las UserStories y de la arquitectura perseguida se crea un plan de versiones entre el equipo de desarrollo y el cliente.

Junto a los UserStories están los escenarios de pruebas que describen el escenario contra el que se comprueba la realización de las UserStories. UserStories y casos de pruebas son las bases sobre la que se asienta el trabajo de desarrollador en XP.

Como primer paso de cada iteración se escribirán las pruebas, de tal forma que puedan ser ejecutadas automáticamente, de manera que pueda comprobarse la corrección del software antes de cada versión. Esto es de vital importancia en XP debido a su apuesta por las iteraciones cortas que generan software que el cliente puede ver y por la refactorización para mejorar el código constantemente, que hacen más deseable una cantidad considerables de pruebas lo más automatizables posible. Así pues, la funcionalidad concreta del software solo se escribe cuando las pruebas para su corrección estén preparadas.

Características esenciales de XP; la metodología se basa en:

- **Pruebas Unitarias**: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándose en algo hacia el futuro, se pueda hacer pruebas de las fallas que pudieran ocurrir. Es como si se adelantaran a obtener los posibles errores.
- **Refabricación (refactorización)**: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- **Programación en pares**: consiste en que dos desarrolladores participen en un proyecto, en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

Las técnicas de XP se dividen en cuatro ámbitos: planificación, diseño, codificación y **pruebas**.

El proceso de pruebas en XP:

La diferencia más grande entre los proyectos de Programación Extrema o XP (Extreme Programming) y la mayoría proyectos tradicionales de desarrollo de software es el concepto del Desarrollo Dirigido por Pruebas TDD (por sus siglas en inglés *Test Driven Development*). Con XP, cada pedazo del código es cubierto por las pruebas de unidad.

«Cualquier característica de un programa para la que no haya una prueba automatizada, simplemente no existe». Y es que éste es el pilar básico sobre el que se sustenta XP. Otros principios son susceptibles de ser adaptados a las características del proyecto, de la organización, del equipo de desarrollo; pero este especifica que si no se realizan pruebas, no se está haciendo XP.

TDD, también es una técnica de diseño, ya que aclara la forma de atacar los problemas, valida las decisiones y promueve buenas prácticas de diseño.

Procedimiento de TDD:

- ✓ Escriba una prueba para la nueva funcionalidad.
- ✓ Ejecutar y observar fallos.
- ✓ Escribir el código suficiente para que pase la prueba.
- ✓ Ejecutar la prueba y observar el cumplimiento de la misma.
- ✓ Refactorizar para eliminar duplicación.
- ✓ Ejecutar la prueba nuevamente.
- ✓ Repetir hasta que el código haga lo que se requiere.

TDD se hace cargo de diversos aspectos que en otras metodologías suceden en el desarrollo de un sistema tales como:

- ➢ Normalmente las pruebas son realizadas después que el código esta escrito, lo cual provoca una disminución del alcance de las pruebas, en especial en códigos fuentes de cierto tamaño.
- ➢ Las pruebas algunas veces son realizadas por otras personas que no escribieron el código, esto provoca que ciertos caminos del código no se prueben, provocando con esto una cobertura menor.
- ➢ El diseño de las pruebas se basa en la documentación de lo que el sistema debe hacer. Si la documentación está actualizada lo más que se podrá hacer es conocer que el sistema hace lo que esta escrito.

En TDD estos aspectos cambian, ya que la producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada UserStories que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial.

Las UserStories dan lugar a las pruebas de aceptación formales en XP. Estas pruebas se hacen de acuerdo con el cliente, y solo se consideraría una UserStories acabada cuando esta se cumpla. Por tanto, en una relación comercial se debe formalizar en qué van a consistir estas pruebas. En un proyecto de software libre, los objetivos los marca una comunidad, así que no es tan interesante formalizarlos. Interesa más facilitar a los usuarios la aportación de opiniones sobre cómo se están cubriendo sus necesidades.

Sin embargo, las pruebas de unidad sí son útiles. Cada módulo que se desarrolle puede tener una batería de pruebas de unidad que comprueben que funciona correctamente a lo largo del tiempo. Esto combinado con un proceso de integración continua permite conocer el grado de funcionamiento del proyecto completo. Si las pruebas de unidad se crean antes de desarrollar las funcionalidades que prueban, también valen como indicativo de cuánto falta para completar una UserStories.

Uno de los pilares de la metodología XP es el uso de pruebas para comprobar el funcionamiento de los códigos que se implementan.

El uso de las pruebas en XP es el siguiente:

- Se deben crear las aplicaciones que realizarán las pruebas con un entorno de desarrollo específico para pruebas.
- Hay que someter a pruebas las distintas clases del sistema omitiendo los métodos más triviales.
- Se deben crear las pruebas que pasarán los códigos antes de implementarlos.

Un punto importante es crear pruebas que no tengan ninguna dependencia del código que en un futuro evaluará. Hay que crear las pruebas abstrayéndose del futuro código, de esta forma se asegurará la independencia de la prueba respecto al código que evalúa.

El uso de las pruebas es adecuado para observar la refactorización. Las pruebas permiten verificar que un cambio en la estructura de un código no tiene porqué cambiar su funcionamiento.

Las pruebas mencionadas anteriormente sirven para evaluar las distintas tareas en las que ha sido dividida una historia de usuario o Userstories. Para asegurar el funcionamiento final de una determinada historia de usuario se deben crear la “Prueba de aceptación”; estas pruebas son creadas y usadas por los clientes para comprobar que las distintas historias de usuario cumplen su cometido.

Los usuarios en XP realizan varios roles relacionados con la prueba, tales como:

- **Programador**

Escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

- **Cliente/Usuario**

El cliente escribe las UserStories y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

- **Encargado de pruebas (Tester)**

El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

En XP aumentar la calidad conduce a que el proyecto pueda realizarse en menos tiempo. En efecto, en cuanto el equipo de desarrollo se habitúe a realizar pruebas intensivas y se sigan estándares de codificación, poco a poco comenzará a avanzar mucho más rápido de lo que lo hacía antes, mientras la calidad del proyecto se mantiene asegurada por las pruebas, lo que conlleva mayor confianza en el código y, por tanto, mayor facilidad para adaptarse al cambio, sin estrés, lo que hace que se programe más rápido.

Las técnicas de eXtreme Programming proporcionan un camino para obtener productos de calidad medible en el desarrollo de proyectos de software libre. El hecho de que sea una metodología ligera, la hace especialmente idónea para un entorno heterogéneo con un grupo grande de desarrolladores.

COSIDERACIONES FINALES

A partir del estudio y análisis de la documentación existente sobre el desarrollo de pruebas de software, se ha llegado a las siguientes conclusiones:

- El objetivo de la prueba de software es descubrir errores. Para conseguir este objetivo se planifica y se ejecuta una serie de pasos que van revisando todos los elementos del software. En todas las fases del desarrollo del proyecto hay que probar el software que se va construyendo.

- La prueba de software es una actividad que forma parte de la labor del ingeniero informático y exige un perfil y una formación muy particular, en la que se utilizan técnicas y herramientas para detectar niveles inadecuados de calidad, aplicando una cantidad de recursos limitados, en especial tiempo y dinero, de forma tal que genere un valor agregado en el proceso de desarrollo de software.

CAPÍTULO II. Caracterización de técnicas y estrategias para el desarrollo de pruebas.

2.1 INTRODUCCIÓN.

En este capítulo se documentan los aspectos fundamentales relacionados con los métodos y las técnicas más eficientes para el desarrollo exitoso de pruebas. Como se ha analizado en el capítulo anterior, ello requiere de personal especializado y capacitado en los procesos de planificación, diseño y ejecución de pruebas de aplicaciones de software.

Según un análisis de lo documentado en el capítulo anterior, para la mejora de un proceso de pruebas se recomienda poner atención a tres aspectos:

- ✓ Metodología: Define los procedimientos, guías y plantillas que serán utilizados. Esto permitirá un marco de trabajo que podrá ser entrenado, verificado, medido, repetido y mejorado. El repetir un proceso permitirá mejorar continuamente el rendimiento del personal utilizando las experiencias anteriores.
- ✓ Técnicas y Herramientas: Proveen los elementos necesarios para realizar una planificación, estimación, análisis, diseño y ejecución sistemática de las actividades de pruebas que serán realizadas.
- ✓ Personal Capacitado: Ingenieros y técnicos que han recibido entrenamiento formal y cuentan con la experiencia y habilidades necesarias para utilizar eficientemente la Metodología y las Técnicas y Herramientas que forman parte del proceso de pruebas de la organización.

Debido a lo anteriormente planteado en los epígrafes siguientes se analizan con detalle una serie de métodos, modelos y técnicas de prueba a utilizar para una mejora en el desarrollo de pruebas de software en proyectos productivos.

2.2 Técnicas y estrategias de prueba.

Las técnicas de diseño de casos de prueba tienen como objetivo conseguir una confianza aceptable en que se detectarán los defectos existentes sin necesidad de consumir una cantidad excesiva de recursos (por ejemplo, tiempo para probar o tiempo de ejecución). Toda la disciplina de pruebas debe moverse, por lo tanto, en un equilibrio entre la disponibilidad de recursos y la confianza que aportan los casos para descubrir los defectos existentes. Este equilibrio no es sencillo, lo que convierte a las

pruebas en una disciplina difícil que está lejos de parecerse a la imagen de actividad rutinaria que suele sugerir.

2.2.1 Pruebas de requerimientos o revisión técnica de requisitos.

La prueba de requerimientos pretende comprobar los tres principales atributos de calidad de los requisitos: corrección (carencia de ambigüedad), coherencia (especificación completa y clara del problema) y consistencia o exactitud de los requisitos (que no haya requisitos contradictorios). Estos aspectos se verifican con el fin de prevenir tantos errores como sea posible cuanto antes y de que se facilite el diseño y ejecución de los casos de prueba.

Durante el proceso de inspección de los requerimientos, una persona, designada por el equipo de Aseguramiento de Calidad del Software o Software Quality Assurance (SQA), revisará el documento de especificación de requerimientos, con la lista de chequeo general del documento y la lista de chequeo de requerimientos. La corrección del contenido del documento será responsabilidad del analista y el usuario, quienes son los encargados de aprobar los requerimientos definidos en el documento.

Procedimiento para desarrollar el proceso de inspección de requerimientos.

1. Se determina si los objetivos son claros, verificables y necesarios. El resultado de esta revisión se consigna en la lista de chequeo de objetivos.
2. Mediante un proceso iterativo se define la funcionalidad esperada del software, y se consigna usando el documento de requisitos del sistema.
3. Debe verificar el documento de requerimientos usando la lista de chequeo general del documento de especificación de requerimientos, la lista de chequeo general del documento de inspección y análisis de requerimientos.
4. Revisar cada requerimiento (consistencia, ambigüedad, coherencia), usando para ello la lista de chequeo de requerimientos.

2.2.2 Niveles de Prueba.

Existen 5 niveles o etapas de pruebas, que a la vez constituyen estrategias establecidas:

2.2.2.1 Prueba de unidad.

Es el proceso de verificación en la menor unidad del diseño del software: el módulo, normalmente realizada por el propio personal de desarrollo en su entorno. Usando la descripción del diseño del procedimiento como guía, se prueban los caminos de control importantes, con el fin de descubrir errores dentro del límite del módulo.

Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del módulo. Si los datos no fluyen correctamente, todas las demás pruebas no tienen sentido. (PRESSMAN 2005)

Lista de comprobaciones para la prueba de Interfaces.

- ✓ ¿Es igual el número de parámetros de entrada al número de argumentos?
- ✓ ¿Coinciden los atributos de los parámetros y los argumentos?
- ✓ ¿Coinciden los sistemas de unidades de los parámetros y de los argumentos?
- ✓ ¿Son iguales los números de los argumentos transmitidos a los módulos de llamada que el número de parámetros?
- ✓ ¿Son iguales los atributos de los argumentos transmitidos a los módulos de llamada y los atributos de los parámetros?
- ✓ ¿Son iguales los sistemas de unidades de los argumentos transmitidos a los módulos de llamada y de los parámetros?
- ✓ ¿Son correctos el número de los atributos y el orden de los argumentos de las funciones incorporadas?
- ✓ ¿Existen referencias a parámetros que no estén asociados con el punto de entrada actual?
- ✓ ¿Entran sólo argumentos alterados?
- ✓ ¿Son consistentes las definiciones de variables globales entre los módulos?
- ✓ ¿Se pasan las restricciones como argumentos?

La pruebas del camino básico y de bucles son técnicas muy efectivas para descubrir una gran cantidad de errores en los caminos.

Este nivel de prueba se deben descubrir errores tales 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.

Procedimiento de la Prueba de Unidad. (PRESSMAN 2005)

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

Los controladores y los resguardos son una sobrecarga de trabajo. Es decir, ambos son software que debe desarrollarse (normalmente no se aplica un diseño formal) pero que no se entrega con el producto de software final. Si los controladores y resguardos son sencillos, el trabajo adicional es relativamente pequeño. Desgraciadamente, muchos componentes no pueden tener una adecuada prueba unitaria con un sencillo software adicional. En tales casos, la prueba completa se pospone hasta que se llegue al paso de prueba de integración (donde también se usan controladores o resguardos).

La prueba de unidad se simplifica cuando se diseña un módulo con un alto grado de cohesión. Cuando un módulo sólo realiza una función, se reduce el número de casos de prueba y los errores se pueden predecir y descubrir más fácilmente.

Ventajas de la utilización de este tipo de prueba.

Las ventajas de usar este tipo de pruebas son muchas, entre ellas se pueden plantear:

- ➢ Los errores serán más fáciles de localizar.
- ➢ Los errores estarán más acotados, ya que se sabrá qué módulos no están pasando las pruebas unitarias.
- ➢ Se reducen los errores ocurridos como consecuencia de la eliminación de otros errores, ya que aplicando las pruebas unitarias se pueden ejecutar nuevamente las pruebas y comprobar que el módulo funciona de la forma esperada.

- Con pruebas unitarias, la mayoría de los errores de programación se detectan durante la propia etapa de programación, lo que tiene gran valor, ya que mientras más tiempo permanezca un error en el sistema, más tiempo requerirá eliminarlo y más repercusiones acarreará en otras secciones del programa.
- Las pruebas funcionales se hacen más sencillas, ya que solo se comprobarán la correcta relación entre las distintas unidades.
- El programador escribe código de una forma más lógica, pues lo diseña mucho más simple y accesible para poder realizarle las pruebas.

2.2.2.2 Prueba de integración.

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es coger los módulos probados mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. (PRESSMAN 2005)

A menudo hay una tendencia a intentar una integración no incremental; es decir, a construir el programa mediante un enfoque de <*big bang*>*.* Se combinan todos los módulos por anticipado. Se prueba todo el programa en conjunto. ¡Normalmente se llega al caos! Se encuentran un gran conjunto de errores. La corrección se hace difícil, puesto que es complicado aislar las causas al tener delante el programa entero en toda su extensión. Una vez que se cogen esos errores aparecen otros nuevos y el proceso continua en lo que parece ser un ciclo sin fin.

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

Existen dos estrategias de integración incremental:

Integración Descendente (TOP-DOWN):

Se integran los módulos moviéndose hacia abajo por la jerarquía de control. Comenzando por el módulo principal, los módulos subordinados se van incorporando a la estructura bien, en forma *primero en profundidad,* que integra todos los módulos de un camino de control principal de la estructura, o

primero en anchura, que incorpora todos los módulos directamente subordinados a cada nivel, moviéndose por la estructura de forma horizontal.

El proceso de integración se realiza en una serie de cinco pasos (PRESSMAN 2005):

- 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 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.

Al aplicar esta estrategia pueden surgir algunos problemas, el más común se da cuando se requiere un proceso de los niveles más bajos de la jerarquía para poder probar adecuadamente los niveles superiores. Al principio de la prueba descendente, los módulos de bajo nivel se reemplazan por resguardos; por tanto, no pueden fluir datos significativos hacia arriba por la estructura del programa.

Para solucionar esto se tienen tres opciones: (PRESSMAN 2005)

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 (BOTTOM-UP):

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

Se puede implementar una estrategia de integración ascendente mediante los siguientes pasos planteados en (PRESSMAN 2005):

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

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

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

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

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

Los módulos críticos deben probarse lo antes posible.

2.2.2.3 Prueba de Validación.

En la prueba de validación, el software totalmente ensamblado, se prueba como un todo para comprobar si cumple los requisitos funcionales y de rendimiento, facilidad de mantenimiento, recuperación de errores, etcétera.

La validación puede definirse de muchas formas, pero una simple (aunque vulgar) definición es que la validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente. Las expectativas están definidas en la Especificación de Requisitos del Software, un documento que describe todos los atributos del software visibles para el usuario. (PRESSMAN 2005)

La validación del software se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de pruebas que se va a llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisface todos los requisitos funcionales, que se alcanzan todos los requisitos de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requisitos (por ejemplo, portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento). (PRESSMAN 2005)

2.2.2.4 Prueba de sistema.

La prueba del sistema es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados.

Tipos de Pruebas del Sistema. (PRESSMAN 2005)

- **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.
- **Prueba de Seguridad:** Intenta verificar que los mecanismos de protección incorporados en el sistema lo protegerán, de hecho, de acceso impropios.
- **Prueba de Resistencia:** Están diseñadas para enfrentar a los programas con situaciones anormales, se prueba la robustez del sistema frente al uso intensivo de la aplicación por parte de los usuarios.
- **Prueba de Rendimiento:** Está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. Eficiencia medida en velocidad de proceso y recursos consumidos.

Una actividad asociada al proceso de prueba es la depuración de programas. Esta actividad consiste en la localización del origen de los errores encontrados en el proceso de prueba. Esta actividad se lleva a cabo, fundamentalmente, mediante la aplicación de tres técnicas:

- Fuerza bruta: No se sigue ningún criterio en concreto (poco aconsejable).
- Retroceso: A partir del momento en que el error se manifiesta se retrocede en los subprocesos hasta encontrar el origen del error.

- Eliminación de causas: Se introducen marcas especiales en el programa para averiguar por donde pasa el código, eliminando trozos de programa en cada paso (donde no se encuentra la causa del error).

2.2.2.5 Prueba de aceptación.

En la prueba de aceptación el usuario comprueba en su propio entorno de explotación si acepta el producto desarrollado o no.

Para el planeamiento de la prueba de aceptación, los usuarios desempeñan un papel de soporte. Los clientes junto con planificadores de la prueba diseñan los casos reales de prueba que funcionarán durante prueba de aceptación.

Los desarrolladores de prueba deben tener presente que el software se está desarrollando para satisfacer las exigencias de los consumidores, y no importa cuán elegante es su diseño, él no será aceptado por los usuarios a menos que les ayude a alcanzar sus metas según lo especificado en los requisitos. Las pruebas de aceptación permiten que los usuarios evalúen el software en términos de sus expectativas y metas.

Cuando el software se está desarrollando para un cliente específico, las pruebas de aceptación se realizan después de prueba del sistema. Los casos de la prueba de aceptación se basan en los requisitos.

2.2.3 Prueba de caja blanca.

La prueba de caja blanca del software comprueba los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coinciden con el esperado o mencionado.

Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código.

En (PRESSMAN 2005) se plantea que mediante los métodos de prueba de la caja blanca, el ingeniero de software puede obtener casos de prueba que garanticen que:

1. Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.

2. Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
3. Ejecuten todos los bucles en sus límites y con sus límites operacionales.
4. Se ejerciten las estructuras internas de datos para asegurar su validez.

La prueba de la caja blanca, a primera vista, podría parecer impracticable puesto que no es posible aplicarla exhaustivamente para grandes sistemas, sin embargo no se debe desechar, ya que se puede elegir y ejercitar una serie de caminos lógicos importantes, que invoquen además las estructuras de datos más importantes para comprobar su validez.

La realización de este tipo de pruebas se fundamenta en los siguientes puntos:

- Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa.
- A menudo se cree que un camino lógico tiene pocas posibilidades de ejecutarse cuando, de hecho se puede ejecutar de forma regular.

2.2.3.1 Métodos de la Prueba de Caja Blanca. (PRESSMAN 2005)

- **La prueba del camino básico:** Esta prueba permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.
- **La prueba de condición o estructura de control:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- **La prueba de flujo de datos:** Se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **La prueba de bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

Los métodos de caja blanca pueden aplicarse a las operaciones que se definen para una clase. Sin embargo, la concisa estructura de muchas operaciones de la clase provoca que algunos argumenten que el esfuerzo aplicado en la prueba de tipo caja blanca pudiera redirigirse mejor hacia pruebas a nivel de clase.

2.2.4 Prueba de caja negra.

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

- Verifican las especificaciones funcionales y no consideran la estructura interna del programa.
- Es hecha sin el conocimiento interno del producto.
- No validan funciones ocultas (por ejemplo funciones implementadas pero no descritas en las especificaciones funcionales del diseño) por tanto los errores asociados a ellas no serán encontrados.

Los métodos de prueba de caja negra son apropiados para los sistemas OO (Orientados a Objetos). Los casos de uso brindan datos de entrada muy útiles en el diseño de pruebas de caja negra y basada en estados.

2.2.4.1 Métodos de pruebas de caja negra.

1. Partición equivalente o clases de equivalencia.

La partición equivalente es un método de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de pruebas. Un caso de prueba ideal descubre de forma inmediata una clase de errores (por ejemplo, un proceso incorrecto de todos los datos de carácter) que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de pruebas que hay que desarrollar.(PRESSMAN 2005)

El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada.

Si un conjunto de objetos puede unirse por medio de relaciones simétricas, transitivas y reflexivas, entonces existe una clase de equivalencia, la cual representa un conjunto de estados válidos o no, para condiciones de entrada. Típicamente una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.(PRESSMAN 2005)

Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices establecidas en (PRESSMAN 2005):

1. Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas
3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.
4. Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.

La partición equivalente se ejecuta en dos pasos. El primero es identificar las clases de equivalencia y el segundo es identificar los casos de pruebas. Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada.

Identificación de clases de equivalencia

Para cada entrada externa:

- ✓ Si la entrada específica un rango de valores válidos, se define una clase de equivalencia válida (dentro del rango) y dos no válidas (una a cada lado del rango).

Ejemplo: Si la entrada requiere un mes en el rango de 1 a 12, define una clase de equivalencia válida para los meses del 1 al 12 y dos no válidas (mes<1 y mes>12)

- ✓ Si la entrada especifica un número N de valores válidos, se define una clase de equivalencia válida y dos clases de equivalencias no válidas (ninguno y más de N).

Ejemplo: Si la entrada requiere los títulos de al menos 3 pero no más de 8 títulos de libros, entonces se define un valor de clase de equivalencia válido y dos clases de equivalencias no válidas (<3 y 8 libros).

- ✓ Si la entrada específica un conjunto de valores válidos, se define un valor de clase de equivalencia válido (dentro del conjunto) y una clase de equivalencia no válida (fuera del conjunto).

Ejemplo: Si la entrada requiere uno de los nombres TOM, DICK o HARRY, entonces se define una clase de equivalencia válida (usando uno de los nombres válidos) y uno no válido (usando el nombre JOE).

- ✓ Si hay una razón para creer que el programa maneja cada valor de entrada diferentemente, entonces se define un valor de clase de equivalencia válido por cada valor de entrada válido.
- ✓ Si la entrada específica una situación de tiene que ser, define un valor de clase de equivalencia válido y otro no válido.

Ejemplo: Si el primer carácter de la entrada tiene que ser numérico, entonces se define un valor de clase de equivalencia válida donde el primer carácter es un número y una clase de equivalencia no válida donde el primer carácter no es un número.

- ✓ Si hay una razón para creer que los elementos en una clase de equivalencia no son manejados de manera idéntica por el programa, se subdivide la clase de equivalencia en clases de equivalencias más pequeñas.

Identificar casos de prueba a partir de las clases de equivalencia

- ✓ Asignar un único número para cada clase de equivalencia.
- ✓ Hasta que todas las clases de equivalencias válidas no hayan sido cubiertas por casos de prueba, escribir un nuevo caso de prueba cubriendo tantas clases de equivalencia no cubiertas como sea posible.
- ✓ Hasta que todas las clases de equivalencia no válidas no hallan sido cubiertas por casos de prueba, escribir un caso de prueba que cubra una, y solo una, de las clases de equivalencia no válidas.
- ✓ Si múltiples clases de equivalencia no válidas son ejecutadas en el mismo caso de prueba, algunos casos puede que nunca sean ejecutados porque el primer caso puede bloquear a otros casos o terminar la ejecución del caso de prueba.

2. Análisis de Valores Límites. (PRESSMAN 2005)

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

El análisis de valores límites es una técnica de casos de prueba que complementa la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de pruebas en los extremos de la clase en lugar de centrarse solamente en las condiciones de entrada y obtiene casos de pruebas también para el campo de salida.

Diferencias de AVL con particiones de equivalencia:

1. No se elige "cualquier elemento" de la clase de equivalencia, sino uno o más, de manera que los márgenes se sometan a prueba.
2. Los casos de prueba se generan considerando también el espacio de salida.

Reglas para identificar casos en AVL:

- ✓ Si una condición de entrada especifica un rango delimitado por los valores a y b, se deben generar casos para a y b y casos no válidos justo por debajo y justo por encima de a y b, respectivamente.
- ✓ Si una condición de entrada especifica un número de valores, se deben desarrollar casos de prueba que ejerciten los valores máximo y mínimo, uno más el máximo y uno menos el mínimo.
- ✓ Aplicar las directrices 1 y 2 a las condiciones de salida.
- ✓ Si las estructuras de datos internas tienen límites preestablecidos, hay que asegurarse de diseñar un caso de prueba que ejercite la estructura de datos en sus límites.

  Rango de entrada: [-1.0, 1.0]

  - Casos de prueba para –1.0, +1.0, -1.001, +1.001.
- ✓ Si una condición de entrada especifica un número de valores, se deben desarrollar casos de prueba que ejerciten los valores máximo y mínimo, uno más el máximo y uno menos el mínimo.

  "El fichero de entrada tendrá de 1 a 255 registros"

  - Casos para 0, 1, 254, 255 registros.
- ✓ Aplicar las directrices 1 y 2 a las condiciones de salida.

  "El programa podrá mostrar de 1 a 4 listados"

  - Casos para intentar generar 0, 1, 4 y 5 listados.

2.2.5 Una descripción del Modelo de Madurez de Pruebas (TMM).

Hacia Finales de 1996, en el Instituto de Tecnología de Illinois, se creó el TMM o Testing Maturity Model; el objetivo principal de TMM es conducir las evaluaciones y mejoras de los procesos de pruebas en las organizaciones que desarrollan software.

En principio, TMM está guiado por el conjunto de los conceptos básicos que dieron origen al Modelo de Capacidad y Madurez o CMM (Capability Maturity Model), y está compuesto por dos componentes

principales: un conjunto de niveles de madurez y un modelo de evaluación. De este modo, en el TMM se describe la posición que un determinado proceso de pruebas ocupa en la jerarquía de madurez de las pruebas.

En (BURNSTEIN 2003) se expone que en el TMM la mejora del proceso de prueba es apoyada por un sistema de niveles y de metas de la madurez. El logro de las metas de la madurez da lugar a la mejora incremental del proceso de prueba en una organización.

El desarrollo de la versión 1.0 del TMM fue dirigido por el trabajo hecho en el modelo para el software (CMM), un modelo de madurez de la capacidad de mejora del proceso que ha recibido la ayuda extensa de la industria del software en los Estados Unidos. El CMM es, arquitectónicamente clasificado, como el modelo de mejora de proceso organizado. Este tipo de arquitectura del modelo de mejora de proceso prescribe las fases que una organización debe proceder de manera ordenada para mejorar su proceso de desarrollo de software.

Se deben considerar las siguientes como características generales del Modelo TMM:

- Es un modelo complementario y compatible a CMM y por consecuencia al Modelo de Capacidad y Madurez para el Software o CMM-SW (Capability Maturity Model for Software).
- Está basado en una validación de la situación actual del proceso de pruebas a través de reglas claras y objetivas.
- Estimula la mejora continua de los procesos de pruebas de software.
- Es un modelo basado en las mejores prácticas de pruebas de software existentes en el mercado.

2.2.5.1 Niveles del TMM.

La estructura interna del TMM es rica en prácticas de prueba que se pueden aprender y aplicar de manera sistemática para apoyar un proceso de calidad de la prueba que mejore en pasos incrementales.

Hay cinco niveles en el TMM que prescriben una jerarquía de la madurez y una trayectoria evolutiva para la mejora del proceso. Las características de cada nivel se describen en términos de las metas de la capacidad organizacional de prueba, y roles o responsabilidades para los jugadores dominantes en el proceso de prueba, los encargados, desarrolladores o probadores, y usuarios o clientes.

Cada nivel, a excepción del nivel 1, tiene una estructura que consiste en lo siguiente:

- **Un sistema de metas de la madurez.** Las metas de la madurez identifican las metas de mejora para la prueba que se deben tratar para alcanzar la madurez en ese nivel. Para estar clasificado en un nivel, una organización debe satisfacer las metas de madurez de ese nivel.
- **Submetas de soporte de la madurez.** Definen el alcance, los límites y las actividades necesarias para un nivel particular.
- **Actividades, tareas y responsabilidades (ATR).** Se pone en práctica de la dirección de ATRs y las ediciones de organización de la adaptación en cada nivel del TMM. Se identifican las actividades y las tareas de soporte, y las responsabilidades se asignan a los grupos apropiados.

Nivel 1- Inicial: (No existen metas de madurez)

En el nivel 1 del TMM, la prueba es un proceso caótico; es mal definido, y no distinguido en eliminar errores. Generalmente no existe un sistema documentado de las especificaciones para el comportamiento del software. Las pruebas se desarrollan a propósito de la terminación de la codificación. La prueba y el eliminar errores se intercalan para conseguir los fallos del software.

Los productos de software se lanzan a menudo sin garantía de calidad. Hay una carencia de recursos, de herramientas y de personal correctamente entrenado. Este tipo de organización estaría en el nivel 1 del CMM.

Nivel 2- Definición de fase: (Meta 1: Desarrolle la prueba y las metas de eliminación de errores; Meta 2: Inicie un proceso de planificación de prueba; Meta 3: Institucionalice las técnicas y los métodos básicos de prueba)

El nivel 2 de la prueba en el TMM se separa la eliminación de errores y se define como la fase que sigue a la codificación. Es una actividad prevista, sin embargo, el planeamiento de la prueba en el nivel 2 puede ocurrir después de codificar por razones relacionadas con la inmadurez del proceso de prueba.

La meta fundamental de la prueba a este nivel de la madurez es demostrar que el software resuelve las especificaciones establecidas. Las técnicas y los métodos básicos de prueba están en su lugar; por ejemplo, el uso de estrategias de pruebas de caja negra y de caja blanca. La prueba tiene varios

niveles: existen de unidad, de integración, de sistema, y niveles de aceptación. Muchos problemas de calidad en este nivel del TMM ocurren porque el planeamiento de la prueba sucede tarde en el ciclo de vida del software. Además, los defectos son propagados desde los requisitos y desde fases que se diseñan en el código. No hay programas de revisión, hasta ahora, para tratar esta edición importante. Con un código consolidado, la prueba basada en ejecución todavía se considera una actividad de prueba primaria.

Nivel 3- Integración: (Meta 1: Establezca una organización de prueba de software; Meta 2: Establezca un programa de entrenamiento técnico; Meta 3: Integre la prueba al ciclo de vida del software; Meta 4: Controle y supervise la prueba)

En el nivel 3 del TMM, la prueba no es solo una fase que sigue a la codificación, pues se integra completamente en el ciclo de vida del software Las organizaciones pueden fundamentarse en las habilidades experimentales de planificación de pruebas que han adquirido en el nivel 2. A diferencia del nivel 2, en el nivel 3 del TMM, la planificación de la prueba comienza en la fase de los requisitos y continúa a través del ciclo de vida. Los objetivos de la prueba se establecen con respecto a los requisitos basados en necesidades del usuario o cliente, y se utilizan para el diseño de casos de prueba.

Hay una organización de prueba, y se reconoce la prueba como una actividad profesional. Hay una organización técnica de entrenamiento con enfoque a la prueba. La prueba es supervisada para asegurarse que se está trabajando según el plan y que pueden tomarse acciones si ocurren desviaciones. Las herramientas básicas apoyan actividades de prueba claves, y el proceso de prueba es visible en la organización.

Aunque las organizaciones a este nivel comienzan a jugar un papel importante en las revisiones de control de calidad, no hay un programa formal de revisión, y las revisiones no tienen un lugar aún en el ciclo de vida. Todavía no se ha establecido un programa formal de la medida de la prueba para cuantificar un número significativo de cualidades del proceso y del producto.

Nivel 4- Gestión y medición: (Meta 1: Establezca un programa amplio de revisión en la organización; Meta 2: Establezca un programa de medición de la prueba; Meta 3: Evaluación de la calidad del software)

La prueba en el nivel 4 se convierte en un proceso que es medido y cuantificado. Las revisiones en todas las fases del proceso del desarrollo se reconocen ahora como las actividades del control de la prueba/calidad.

Los productos de software son probados para cualidades de calidad tales como confiabilidad, utilidad, y capacidad de mantenimiento. Los casos de prueba de todos los proyectos se recogen y se registran en una base de datos de casos de pruebas con el fin de la reutilización del caso de prueba y de realizar la prueba de regresión.

Los defectos son registrados y se les da un nivel de severidad. Algunas de las deficiencias que ocurren en el proceso de prueba ocurren debido a la carencia de una filosofía de prevención de defectos, y de apoyo automatizado para la colección, análisis, y diseminación de prueba relacionada a la métrica.

Nivel 5- Control de Optimización /Prevención de defecto /Calidad: (Meta 1: Prevención de defectos; Meta 2: Control de calidad; Meta 3: Prueba de optimización del proceso).

Debido al lugar en que está la infraestructura con el logro de las metas de madurez en los niveles del 1 al 4 del TMM, el proceso de prueba ahora, puede decirse que, está definido para ser manejado; su coste y eficacia pueden ser supervisados. En el nivel 5, los mecanismos están en un punto que permite mejorar continuamente la prueba. Se practican la prevención y el control de calidad del defecto.

Las herramientas automatizadas apoyan totalmente el funcionamiento y la reejecución de los casos de prueba. Las herramientas también proporcionan la ayuda para el diseño de casos de pruebas, mantenimiento de artículos relacionados con pruebas, y la colección y análisis de defectos.

La colección y el análisis de prueba relacionada a la métrica también obtienen una ayuda en la herramienta. La reutilización del proceso es también una práctica en el nivel 5 del TMM apoyado por una biblioteca de recursos del proceso.

El foco del TMM está en la prueba como proceso en sí mismo que pueda ser evaluado y mejorado. En dominio de prueba las ventajas posibles de la mejora de proceso de la prueba son las siguientes:

- probadores más elegantes.
- un software de más alta calidad.
- la capacidad de satisfacer el presupuesto y metas a programar.
- planeamiento mejorado.

- la capacidad de satisfacer metas de prueba cuantificables.

2.2.6 Técnicas de diseño de casos de prueba.

Antes de las pruebas propiamente, es necesario identificar los escenarios de negocios, definir los casos de pruebas y los datos de pruebas.

En la práctica resulta imposible definir y aplicar todos los posibles casos para una aplicación de software. El número de combinaciones de un conjunto de variables es "2^n" siendo "n" el número de variables. Más aún, el número de errores posibles es 3^n-1. Una aplicación sencilla podría tener 20 variables. Entonces, el número de combinaciones posibles es 1.048.576 y el número de errores posibles 3.486.784.400. Esto hace necesario aplicar técnicas de generación de casos y conjetura de errores que permitan reducir los casos teóricos a una cantidad de casos razonables para obtener una cierta calidad del producto de software.

Ya que no se pueden probar todas las posibilidades de funcionamiento del software, la idea fundamental para el diseño de casos de prueba consiste en elegir algunas de ellas que, por sus características, se consideren representativas del resto. Así, se asume que, si no se detectan defectos en el software al ejecutar dichos casos, se puede tener un cierto nivel de confianza (que depende de la elección de los casos) en que el programa no tiene defectos. La dificultad de esta idea es saber elegir los casos que se deben ejecutar. De hecho, una elección puramente aleatoria no proporciona demasiada confianza en que se puedan detectar todos los defectos existentes. Por eso es necesario recurrir a ciertos criterios de elección que se verán a continuación.

Existen tres enfoques principales para el diseño de casos:

a) El enfoque **estructural** o de **caja blanca**. Consiste en centrarse en la estructura interna (implementación) del programa para elegir los casos de prueba. En este caso, la prueba ideal (exhaustiva) del software consistiría en probar todos los posibles caminos de ejecución, a través de las instrucciones del código, que puedan trazarse.

b) El enfoque **funcional** o de **caja negra**. Consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos. Aquí, la prueba ideal del software consistiría en probar todas las posibles entradas y salidas del programa.

c) El enfoque **aleatorio** consiste en utilizar estadísticas que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba. La prueba exhaustiva consistiría en probar todas las posibles entradas al programa.

Estos enfoques no son excluyentes entre sí, ya que se pueden combinar para conseguir una detección de defectos más eficaz.

2.2.6.1 Diseño de casos de prueba. Enfoque de prueba de caja blanca.

El diseño de casos tiene que basarse en la elección de caminos importantes que ofrezcan una seguridad aceptable en descubrir los defectos, y para ello se utilizan los llamados criterios de cobertura lógica. Se debe destacar que los criterios de cobertura están enumerados en orden de exigencia y, por lo tanto, de coste económico. Es decir, el criterio de cobertura de sentencias es el que ofrece una menor seguridad de detección de defectos, pero es el que cuesta menos en número de ejecuciones del programa.

1. **Cobertura de sentencias.** Se trata de generar los casos de prueba necesarios para que cada sentencia o instrucción del programa se ejecute al menos una vez.
2. **Cobertura de decisiones.** Consiste en escribir casos suficientes para que cada decisión tenga, por lo menos una vez, un resultado verdadero y, al menos una vez, uno falso. En general, una ejecución de pruebas que cumple la cobertura de decisiones cumple también la cobertura de sentencias.
3. **Cobertura de condiciones.** Se trata de diseñar tantos casos como sea necesario para que cada condición de cada decisión adopte el valor verdadero al menos una vez y el falso al menos una vez. No se puede asegurar que si se cumple la cobertura de condiciones se cumple necesariamente la de decisiones.
4. **Criterio de decisión/condición.** Consiste en exigir el criterio de cobertura de condiciones obligando a que se cumpla también el criterio de decisiones.
5. **Criterio de condición múltiple.** En el caso de que se considere que la evaluación de las condiciones de cada decisión no se realiza de forma simultánea (por ejemplo, según se ejecuta en el procesador), se podría considerar que cada decisión multicondicional se descompone en varias decisiones unicondicionales. Es decir, una decisión como IF ((a=1) AND(c=4)) THEN se convierte en una concatenación de dos decisiones: IF(a=1) y IF(c=4). En este caso, se debe conseguir que todas las combinaciones posibles de resultados (verdadero/falso) de cada condición en cada decisión se ejecuten al menos una vez.

La cobertura de caminos (secuencias de sentencias) es el criterio más elevado: cada uno de los posibles caminos del programa se debe ejecutar al menos una vez. Se define **camino** como la secuencia de sentencias encadenadas desde la sentencia inicial del programa hasta su sentencia final.

Si se trabaja con los caminos de prueba, existe la posibilidad de cuantificar el número total de caminos utilizando algoritmos basados en matrices que representan el grafo de flujo del programa. Saber cuál es el número de caminos del grafo de un programa ayuda a planificar las pruebas y a asignar recursos a las mismas, ya que indica el número de ejecuciones necesarias. También sirve de comprobación a la hora de enumerar los caminos.

Los bucles constituyen el elemento de los programas que genera un mayor número de problemas para la cobertura de caminos. Su tratamiento no es sencillo ni siquiera adoptando el concepto de camino de prueba.

2.2.6.2 Diseño de casos de prueba. Enfoque de prueba de caja negra.

La prueba funcional o de caja negra se centra en el estudio de la especificación del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas. Lamentablemente, la prueba exhaustiva de caja negra también es generalmente impracticable.

Ya que no se pueden ejecutar todas las posibilidades de funcionamiento y todas las combinaciones de entradas y de salidas, se deben buscar criterios que permitan elegir un subconjunto de casos cuya ejecución aporte una cierta confianza en detectar los posibles defectos del software. Para fijar estas pautas de diseño de pruebas, se hará un soporte en las siguientes dos definiciones de Myers que definen qué es un caso de prueba bien elegido:

- El que reduce el número de otros casos necesarios para que la prueba sea razonable. Esto implica que el caso ejecute el máximo número de posibilidades de entrada diferentes para así reducir el total de casos.
- Cubre un conjunto extenso de otros casos posibles, es decir, indica algo acerca de la ausencia o la presencia de defectos en el conjunto específico de entradas que prueba, así como de otros conjuntos similares.

A continuación se tratarán distintas técnicas de diseño de casos de caja negra de las que ya se han documentado sus características en secciones anteriores.

Particiones o clases de equivalencia (Ver epígrafe 2.2.4.1)

Esta técnica utiliza las cualidades que definen un buen caso de prueba de la siguiente manera:

- 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 que cumplan la siguiente propiedad: 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.

Conjetura de errores.

La idea básica de esta técnica consiste en enumerar una lista de equivocaciones que pueden cometer los desarrolladores y de las situaciones propensas a ciertos errores. Después se generan casos de prueba en base a dicha lista. Esta técnica también se ha denominado generación de casos especiales, ya que no se obtienen en base a otros métodos sino mediante la intuición o la experiencia.

No existen directrices eficaces que puedan ayudar a generar este tipo de casos, ya que lo único que se puede hacer es presentar algunos ejemplos típicos que reflejan esta técnica. Algunos valores a tener en cuenta para los casos especiales son los siguientes:

- 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 (por ejemplo una lista), 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. Se dice que se debe prever toda clase de acciones de un usuario como si fuera “completamente tonto” o, incluso, como si quisiera sabotear el programa.

2.2.6.3 Diseño de casos de pruebas. Enfoque de pruebas aleatorias.

En las pruebas aleatorias se simula la entrada habitual del programa creando datos de entrada en la secuencia y con la frecuencia con las que podrían aparecer en la práctica, de forma continua sin parar. Esto implica usar una herramienta denominada generador de pruebas, la que se alimenta con una

descripción de las entradas y las secuencias de entradas posibles y su probabilidad de ocurrir en la práctica.

Este enfoque de prueba es muy común en la prueba de compiladores en la que se generan aleatoriamente códigos de programas que sirven de casos de prueba para la compilación.

Si el proceso de generación se ha realizado correctamente, se crearán eventualmente todas las posibles entradas del programa en todas las posibles combinaciones y permutaciones. También se puede conseguir, indicando la distribución estadística que siguen, que la frecuencia de las entradas sea la apropiada para orientar correctamente nuestras pruebas hacia lo que es probable que suceda en la práctica. No obstante, esta forma de diseñar casos de prueba es menos utilizada que las técnicas de caja blanca y de caja negra.

2.2.7 Inspecciones de código.

El objetivo de las inspecciones establecidas en (MYERS 2004) es implementar un proceso formal de revisión detallada, con el propósito de encontrar defectos en una etapa muy temprana del desarrollo del producto.

Una inspección del código es un sistema de procedimientos y técnicas de detección de errores para la lectura de código en grupo. La mayoría de las discusiones de inspecciones de código se enfocan en los procedimientos, formularios a ser rellenados.

Un equipo de inspección consta generalmente de cuatro personas. Una de estas realiza el rol de moderador. Se espera que el moderador sea un programador competente, pero este no es el autor del programa y no necesita conocimiento de los detalles del programa. Los deberes del moderador incluyen:

- Distribución de materiales para orientar la sesión de la inspección.
- Conducir la sesión.
- Registrar todos los errores encontrados.
- Asegurarse de que los errores estén corregidos posteriormente.

El moderador es como un ingeniero de control de calidad. El segundo miembro del equipo es el programador. Los miembros restantes del equipo son, generalmente, el diseñador del programa (si es diferente al programador) y el especialista de prueba.

El moderador distribuye la especificación del listado y del diseño del programa a los otros participantes varios días antes de la sesión de inspección. Se espera que los participantes se familiaricen con el material antes de la sesión. Durante la sesión, ocurren dos actividades:

- El programador narra, declaración por declaración, la lógica del programa. Durante el discurso, los otros participantes deben plantear preguntas que deben ser seguidas para determinar si existen errores. Es probable que el programador encuentre, más que los otros miembros del equipo, muchos de los errores encontrados durante esta narración. Es decir, el acto simple de leer en voz alta un programa a una audiencia parece ser una técnica eficaz para la detección de errores.
- El programa se analiza con respecto a una lista de comprobación de errores de programación históricamente comunes.

El moderador es responsable de asegurarse que las discusiones procedan a lo largo de líneas productivas y que los participantes centran su atención en encontrar los errores, no corrigiéndolos.

Después de la sesión, al programador se le da la lista de los errores encontrados. Si no fueron encontrados pocos errores, o si cualquiera de los errores requiere una corrección substancial, el moderador puede tomar medidas para reinspeccionar el programa después de que se corrijan los errores. Esta lista de errores también se analiza, se categoriza, y se utiliza para refinar la lista de comprobación de errores para mejorar la eficacia en inspecciones futuras.

El tiempo y local de la inspección deben ser planeadas para evitar interrupciones exteriores. La cantidad de horas óptima para una sesión de inspección debe ser de 90 a 120 minutos. Puesto que la sesión es una experiencia mentalmente extenuante, sesiones más largas tienden para ser menos productivas. La mayoría de las inspecciones proceden a una velocidad de aproximadamente 150 declaraciones del programa por hora. Por esa razón, los programas grandes se deben examinar en inspecciones múltiples, cada inspección ocupa desde uno a varios módulos o subprogramas.

Para que el proceso de inspección sea eficaz, debe ser establecida una actitud adecuada. Si el programador ve la inspección como un ataque contra él y adopta una postura defensiva, el proceso será ineficaz. Más bien, el programador debe acercarse al proceso y debe ver el proceso con luz positiva y constructiva: El objetivo de la inspección es encontrar errores en el programa, para así mejorar la calidad del trabajo.

Finalmente, el proceso de inspección es una manera de identificar temprano las secciones del programa más propensas a error, ayudando a centrar más atención en estas secciones durante los procesos de prueba computarizados.

CONSIDERACIONES FINALES

Existen muchas técnicas y/o estrategias de pruebas de software que pueden ser aplicadas a múltiples proyectos productivos desarrolladores de software, pero para logar una mejora potencial de la calidad del software basada en la eliminación y/o prevención de los defectos y errores, cada una de estas técnicas deben ser adecuadas o modeladas a cada uno de estos proyectos.

Debido a esto, en el transcurso del próximo capítulo se detallarán las características relacionadas al trabajo con pruebas correspondientes a los proyectos productivos de la facultad 5 de la UCI, y se hará una valoración de cuáles de las técnicas de pruebas deben ser aplicadas a estos y en qué condiciones.

CAPÍTULO III. Caracterización de proyectos productivos de la Facultad 5 de la Universidad de las Ciencias Informáticas entorno a la utilización de pruebas de software.

3.1 INTRODUCCIÓN.

En este capítulo se seleccionarán, mediante un análisis riguroso, las técnicas de pruebas de software que son más factibles para la utilización en los proyectos productivos de la Facultad 5 de la Universidad de las Ciencias Informáticas. Primeramente se realizará un estudio y caracterización de algunos proyectos que conforman el perfil productivo de la facultad, basado en el uso que estos le dan a las pruebas de software. Con este análisis y el estudio de técnicas de prueba realizado, se establecerán las metas de pruebas de software que deben conformar estos proyectos.

El proceso de prueba comienza con la generación de un plan de pruebas en base a la documentación sobre el proyecto y la documentación sobre el software a probar. A partir de dicho plan, se entra en detalle diseñando pruebas específicas basándose en la documentación del software y se toma la configuración del software que se va a probar para ejecutar sobre ella los casos.

Como parte de este capítulo se describirá un plan general como propuesta para los proyectos de “Entornos Virtuales”. Los resultados de aplicar un plan y técnicas eficaces de prueba pueden servir para realizar predicciones de la fiabilidad del software y para detectar las causas más habituales de error y mejorar los procesos de desarrollo futuros.

Para establecer los objetivos correctos del plan y de la propuesta de técnicas se ha diseñado un formato de encuesta que permite determinar, en alguna medida, el conocimiento y la importancia que se tiene en los proyectos productivos sobre el desarrollo de pruebas.

Los resultados de la encuesta sirven de retroalimentación dentro del proceso de pruebas de software, y son utilizados para instituir las medidas que se tengan que tomar para solucionar los problemas identificados.

3.2 Caracterización de la Facultad 5 de la UCI.

La Facultad 5 está dedicada al desarrollo de entornos virtuales. Su misión es formar un profesional integral, especializado en la rama de la informática y altamente comprometido con su patria, desarrollar

la investigación y la producción de software para entornos virtuales, en correspondencia con las aspiraciones de desarrollo sostenido y sustentable de la sociedad cubana. (Domínguez 2006)

Actualmente la Facultad 5 cuenta con los siguientes proyectos:

Proyectos internacionales.

- ✓ Supervisory Control And Data Acquisition (SCADA)
- ✓ Auto Teórico - Práctico

Proyectos Nacionales.

- ✓ Laboratorios Virtuales
- ✓ Paseos Virtuales
- ✓ Tiro
- ✓ Simulador de Medicina Quirúrgica
- ✓ Diseño 3D
- ✓ Juegos de Mesa
- ✓ Juegos de Consola
- ✓ Calidad de Software
- ✓ Herramienta para Entornos Virtuales
- ✓ Sistemas de Gestión
- ✓ Elementos Inteligentes

Los principales resultados que se han obtenido con estos proyectos son (Domínguez 2006):

- ✓ Versión de auto teórico, República Dominicana.
- ✓ Simulador Práctico, Guatemala.
- ✓ 1er Volumen multimedia de la Constitución de Venezuela.
- ✓ Guardería en dos idiomas.
- ✓ A jugar (MINED-Primaria).
- ✓ Demo del Simulador de Tiro.

3.2.1 Situación actual de los proyectos productivos según la encuesta (Ver Anexo 9).

Tabla 3.1: Características respecto a la utilización de pruebas de software en los proyectos productivos de la Facultad 5.

Proyectos	Auto Teórico-Práctico	Tiro	Medicina Quirúrgica	Laboratorios Virtuales	SCADA

Tipo de software que desarrolla	Simulador de auto para la obtención de la licencia de conducción	Simulador de Tiro	Simulador de Medicina	Software educativos	Software para la automatización
Productos realizados	✓ Simulador Práctico, Guatemala ✓ Multimedia para examen teórico de conducción, República Dominicana	Engine de Tiro	Demo de habilidades para cirujanos	Videos interactivos sobre enseñanza de la Programación	Solo prototipos y algunos entregables de todo el sistema
Metodologías base utilizadas	RUP	RUP	RUP	RUP	RUP
Técnicas de pruebas utilizadas	Pruebas de unidad	No se utilizan	No se utilizan	No se utilizan	Pruebas de unidad basadas en el método de particiones equivalentes
Elaboración de planes de prueba	No se utilizan	No se utilizan	Se están elaborando	No se utilizan	No se utilizan
Análisis de riesgos para la planificación de pruebas	No	No	No	No	No
Herramientas de pruebas usadas	No se utilizan	No se utilizan	No	No se utilizan	CxxTest, para la automatización de pruebas de unidad
Aplicación de soporte a usuarios insatisfechos	Sí	No	No	No	No
Utilización de pruebas de	Sí	No se utilizan	No se utilizan	No se utilizan	Sí

unidad					
Utilización de pruebas de estrés	No se utilizan	No se utilizan	No se utilizan	No se utilizan	No se utilizan
Utilización de pruebas de rendimiento	No se utilizan	No se utilizan	No se utilizan	No se utilizan	No se utilizan

En la **Figura 3.1**, se evidencia el porcentaje de los resultados de la encuesta aplicada a la muestra de proyectos de la facultad 5 seleccionada, donde se plasma claramente que solo 2 de los 5 proyectos encuestados emplean alguna técnica de prueba en su proceso de verificación y/o validación del producto, lo que representa un 40% de la totalidad de proyectos encuestados.

Ninguno de los proyectos encuestados confecciona planes de prueba para la ejecución de las mismas y solo uno tiene en proceso de elaboración algún plan para la prueba, lo que representa el 0% en la utilización de planes y solo un 20% en proceso de confección de algún plan.

Absolutamente ninguno de los proyectos evalúa los riesgos del negocio para la confección de planes de prueba que se enfoquen principalmente en las características críticas del sistema, lo que representa un 0% de utilización.

Solo uno de los proyectos encuestados utiliza alguna herramienta automatizada para facilitar y ahorrar en tiempo el desarrollo de las pruebas, lo que representa solo el 20% de los proyectos encuestados.

Solo 2 de los proyectos encuestados utilizan la técnica de prueba de unidad en el progreso de la prueba, lo que representa solo el 40% de los encuestados.

Absolutamente ninguno de los proyectos encuestados establece una estrategia de prueba para comprobar la validez de su producto, lo que representa el 0% de los proyectos.

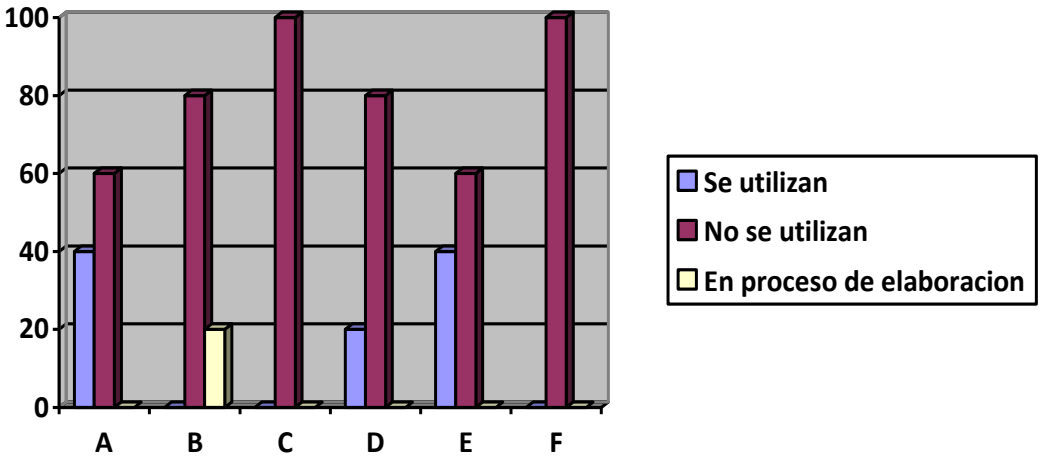


Figura 3.1: Análisis del porcentaje de los resultados en la encuesta aplicada a proyectos productivos de la Facultad 5.

Leyenda:

A: Utilización de técnicas de pruebas.

B: Elaboración de planes de pruebas.

C: Realización de planes basados en riesgos.

D: Utilización de herramientas de pruebas.

E: Realización de técnicas de pruebas de unidad.

F: Utilización de estrategias de pruebas.

Mediante el estudio realizado, las entrevistas sostenidas con los líderes de proyectos y estudiantes, la encuesta aplicada y las características observadas de cada uno de los proyectos productivos de la facultad, se detectaron los siguientes problemas:

1. Insuficiencias en la planificación y control del trabajo.(Domínguez 2006)
2. Las pruebas realizadas no hacen énfasis en las áreas de alto riesgo.
3. La planeación y ejecución de pruebas solo se realiza una vez que el software ha sido desarrollado o cuando una parte significativa de este ha sido implementada.
4. Solo uno de los proyectos utilizan alguna herramienta para la automatización de las pruebas.
5. No son diseñados ni administrados los casos de pruebas a ejecutar.
6. Hay falta de un proceso básico de pruebas y de conocimiento de qué es lo que se debe probar.
7. No se mide calidad de software con la exigencia requerida para software de estos tipos.(Domínguez 2006)

8. Los responsables de prueba no exigen lo suficiente por la calidad y ejecución de estas.
9. No se utiliza un proceso de mejora continua para el desarrollo de pruebas.
10. Existe desconocimiento de las necesidades de los clientes.(Domínguez 2006)
11. No se conocen las principales normas de producción y calidad de software. (Domínguez 2006)

Todos estos problemas en su conjunto traen consigo que existan atrasos en la terminación del software, provocando que para poder cumplir con la misión encomendada, se haga todo a última hora en plazos muy cortos y sin la planificación debida, con lo cual la calidad del producto obtenido resulta afectada, lo que puede traer como consecuencia, entre otras molestias, el tedioso y costoso trabajo de soporte a usuarios insatisfechos.

Algunas de las condiciones que han acarreado en estos problemas son:

- La no mucha importancia que habitualmente se le da a la fase de pruebas.
- La falta de herramientas, debida fundamentalmente a su elevado precio y al desconocimiento de los beneficios que aportan estas.
- Los proyectos no tienen conocimiento del beneficio que aportan las técnicas de prueba en la mejora de la calidad del producto
- Falta de organización del equipo independiente de pruebas.
- Falta de conocimiento del proceso de desarrollo de pruebas.
- No se gestiona el conocimiento.
- Mala planificación del tiempo.
- No se tienen bien establecidos los roles.

Las acciones correspondientes a la calidad de software que se estaban llevando a cabo eran acciones aisladas no contempladas en un plan formal.(Domínguez 2006)

A pesar de los beneficios que se pueden obtener de la correcta aplicación de las técnicas y estrategias de pruebas de software, la utilización de estas técnicas en los proyectos productivos de la Facultad 5 (sobre todo en aquellos que son críticos pues involucran vidas humanas tales como Medicina Quirúrgica, Auto Teórico-Práctico y Tiro), no es suficiente para el logro de los beneficios potenciales y la calidad que realmente requieren estos tipos de software.

El presente trabajo propone para su aplicación consecuente la utilización estricta de planes de prueba basados en el análisis de los riesgos del negocio, la utilización de un equipo independiente para el desarrollo de pruebas y la utilización de algunas técnicas de prueba que se adaptan a las

características específicas de los proyectos productivos de la Facultad 5, tales como pruebas de estrés, de rendimiento, de mutación y pruebas para sistemas de tiempo real. Estas técnicas deben brindar un mejoramiento visible al proceso de pruebas y a la calidad de software en la Facultad 5.

3.3 Propuesta de un plan de prueba guía para los proyectos de Entornos Virtuales.

Como consecuencia a las dificultades ya explicadas surge la propuesta de un plan de prueba guía para los proyectos de la Facultad 5. Un plan de pruebas debe describir qué se debe probar y cómo.

Para obtener una planificación adecuada del proceso de pruebas, aumentando el aprovechamiento del tiempo, realizando en software de forma ordenada y organizando los procesos de prueba con el fin de descubrir los errores en cada una de las etapas de desarrollo, se recomienda que se reúna el líder de proyecto con el responsable de pruebas, que puede ser el responsable de la calidad del software a desarrollar. Debe fluir concientemente una comunicación constante entre ambas partes involucradas, con lo que se ganará en confiabilidad a la hora de realizar la prueba, ya que ambos estarán al tanto de los cambios, resultados y situaciones que surjan inesperadamente para poder actuar. Además se recomienda que se use una lista de chequeo (Anexo 11) para la creación eficiente del plan. Esta lista proporcionará confiabilidad de que el plan de pruebas está completo y correctamente confeccionado.

El equipo de desarrollo del proyecto debe reunirse y planificar los cronogramas de entrega y liberación de cada artefacto de software con el equipo de prueba; esta colaboración debe ser firmada por ambas partes como constancia de la toma de acuerdos para la verificación de los deberes.

Para la confección del plan de pruebas óptimo, deben ser cumplidos una serie de condiciones de gran importancia para los responsables de la prueba, tales como:

- **Obtener los requerimientos en forma clara.**

La lista de requerimientos puede ser comprobada para establecer la medida en que estos cumplen con las necesidades concretas de los clientes. En este aspecto, el equipo de prueba puede solicitar la lista de requerimientos establecida por el grupo de desarrollo para verificar que se implemente un programa con las características esperadas. (Ver Capítulo 2, Prueba de requerimientos)

- **Obtener la planificación de desarrollo.**

Esta medida, permitirá a los probadores establecer el máximo de tiempo que requerirán para realizar las pruebas y para entregar el producto en el plazo de tiempo preestablecido con el cliente.

- **Identificar aplicaciones de alto riesgo o con prioridad de prueba.**

Las prioridades de los planes y los casos de pruebas se deben clasificar como de importancia alta, media o baja. Cuando las pruebas deben llevarse a cabo en poco tiempo, esto ayudará a decidir qué casos deben probarse y cuáles pueden omitirse sin problemas.

Es de vital importancia identificar los riesgos que pueda tener el proceso, con el fin de establecer planes de contingencia para ellos. Entre los riesgos que se deben considerar están:

- ✓ No disponibilidad del ambiente de pruebas (software o hardware) en las fechas requeridas.
- ✓ Cambios en el alcance funcional de la aplicación.
- ✓ No disponibilidad de la sección del producto requerida para la ejecución de pruebas.
- ✓ No disponibilidad de herramientas automáticas para ejecución de pruebas en los casos que se requiera.
- ✓ Complejidad de los procesos a evaluar.
- ✓ Insuficiencia de tiempo, acorde con lo planificado para la entrega del producto, lo que conlleva a que se desfavorezca la etapa de pruebas.

Se recomienda que se analicen las áreas críticas del producto, o sea aquellas partes que si fallan suponen una pérdida de prestigio por la parte desarrolladora ya que determinan pérdidas sustanciales, pueden ser vidas humanas o importantes pérdidas económicas.

- **Determinar métodos de prueba.**

Para la elaboración del plan de pruebas deben estar bien especificados los métodos que se usarán y determinados quienes los ejecutarán y en qué momento del desarrollo se hará.

- **Identificar el alcance de la prueba.**

Deben definirse todos los componentes y aspectos del sistema que deberán ser evaluados, guiándose por los límites de tiempo y los riesgos establecidos.

- **Estimar tiempo de prueba.**

Con el tiempo de desarrollo establecido, los métodos de prueba a utilizar y las áreas de alto riesgo identificadas, se deberá establecer una aproximación del tiempo y esfuerzo a realizar para el desarrollo de la prueba. Esto será de vital importancia para establecer el costo de la prueba y consecuentemente la calidad de la sección del producto a probar.

- **Clasificar y documentar errores del programa.**

Obtener un registro de defectos con la clasificación de cada uno, permitirá que este registro sea empleado en aplicaciones similares para obtener una medida de los errores que pueden ocurrir más frecuentemente y enfocarse en que estos no se repitan o sean más fáciles de localizar y corregir.

- **Redactar los casos de prueba que encontraron fallas.**

Esto permitirá que los casos sean reutilizables en otras partes del programa que contienen condiciones equivalentes o en programas similares, y aportará en el ahorro de tiempo a la hora de diseñar las pruebas y de encontrar los errores introducidos.

- **Evaluar resultados en reportes.**

Las dificultades o no conformidades encontradas en la ejecución de los casos de prueba, deberán ser evaluadas en reportes que serán emitidos al equipo de desarrollo para tomar medidas correctivas, ser usadas como retroalimentación del equipo y evitar que ocurran nuevamente.

- **Volver a probar si es necesario.**

Al ser documentadas las dificultades encontradas en el desarrollo de los casos de prueba y ser informadas a los desarrolladores y corregidas por los mismos, el equipo de prueba deberá establecer si es necesaria la ejecución de alguno o todos los casos de prueba nuevamente, para establecer la conformidad con los nuevos resultados y evaluar si los nuevos cambios no han introducido comportamientos indeseados en el programa.

- **Actualizar el Plan de Pruebas.**

Con cada ejecución de un grupo de casos de prueba debe ser actualizado el plan de pruebas, el cual puede ser reutilizado en futuros procesos de desarrollo.

3.3.1 Propuesta de confección del plan de pruebas para los proyectos de Entornos Virtuales.

<Caso de estudio de un Proyecto de “Entornos Virtuales”>

PLAN DE PRUEBAS

Versión < número Versión>

Revisión histórica

Fecha	Versión	Descripción	Autor

1. Introducción

El plan de pruebas se elabora acorde con los plazos de tiempo establecidos, este guiará el desarrollo de las pruebas y deberá dejar claro, para cada prueba, qué tipo de propiedades se quieren probar, cómo se mide el resultado; debe especificar en qué consiste la prueba y definir cuál es el resultado que se espera. Además el plan debe reflejar el énfasis que hacen las pruebas en las áreas de alto riesgo y debe ser constituido con la seguridad de que es compatible con el plan de desarrollo.

1.1. Propósito

Este documento describe el Plan de pruebas para un sistema de Gestión del perfil “Entornos Virtuales”. En específico define los siguientes objetivos:

- ✓ Elementos a probar.
- ✓ Estrategia de pruebas a seguir en el proceso de prueba.
- ✓ Recursos necesarios para llevar a cabo el proceso de prueba y estimación de los esfuerzos que conlleva.
- ✓ Lista los resultados que se obtienen de las actividades de prueba.

1.2. Ámbito

Este Plan de Pruebas describe las técnicas de pruebas de unidad, integración, sistema y aceptación que se aplicarán al sistema software desarrollado, con sus respectivos métodos de caja negra, estrés, rendimiento, mutación y de tiempo real.

El objetivo es probar los requisitos definidos en la Especificación de requisitos y en el Modelo de casos de uso.

1.3. Definiciones, acrónimos, y abreviaturas

Establezca un glosario de los términos que puedan crear duda a los involucrados en el proceso de pruebas y defina adecuadamente cada uno para su mejor comprensión.

2. Requerimientos de las pruebas

La lista que se proporciona en esta sección identifica los elementos (casos de uso, requisitos funcionales y requisitos no funcionales) que son objetivos de las pruebas. Es decir, los elementos qué se van a probar.

3. Estrategia de prueba

Este plan estará guiado por una estrategia establecida para la ejecución de las técnicas de prueba, la cual es descrita a continuación.

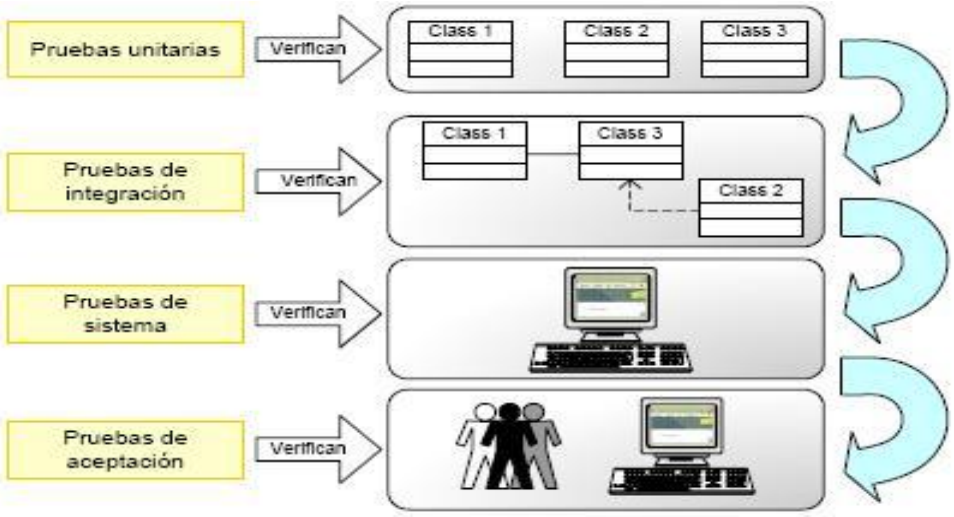


Figura 3.2: Estrategia de prueba a utilizar.

La estrategia de prueba del software puede definirse incrementalmente de “adentro” hacia “afuera”, o sea a partir de la prueba de unidad establecida para evaluar el correcto funcionamiento de los módulos, la prueba de integración establecida para verificar las interacciones entre los módulos, la prueba del sistema para comprobar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas y la prueba de aceptación para definir si el producto cumple con los requisitos establecidos y está listo para ser liberado y puesto en manos de los clientes.

3.1. Tipos de pruebas y técnicas

3.1.1 Prueba de unidad.

Es la escala más pequeña de la prueba, está basada en la funcionalidad de los módulos del programa, como funciones, procedimientos, módulos de clase.

Objetivos de la prueba	Analizar el resultado de ejecutar cada método bajo distintas condiciones o valores de los parámetros de entrada y de los atributos de la clase y analizar la evolución de los atributos de la clase bajo distintas combinaciones de llamadas a sus métodos para comprobar que funcionan correctamente.
Técnicas	Invocar cada clase o método con datos válidos e inválidos y con los valores de los límites para asegurar que los datos son los previstos y que fluyen adecuadamente.

Criterios de finalización	Establecer el funcionamiento de los módulos y los errores encontrados.
Consideraciones	Estas pruebas deben ser ejecutadas con la utilización de alguna herramienta de prueba para la automatización de las mismas (CxxTest).

3.1.2 Pruebas de integración.

Una vez probados las unidades lógicas del código sería de gran importancia establecer si las relaciones establecidas entre cada una de estas fluye correctamente y no proporciona salidas erróneas en el programa.

Objetivos de la prueba	El objetivo principal de las pruebas de integración es detectar fallos en la interacción entre las distintas clases que componen el sistema.
Técnicas	• Ir combinando los módulos de los niveles inferiores probando cada grupo moviéndose hacia arriba por la estructura del programa. • Utilizar técnicas de valores límites y clases de equivalencia, para determinar que los datos de los módulos que interaccionan fluyen correctamente.
Criterios de finalización	Establecer todas las combinaciones existentes entre módulos, comprobando su correcto funcionamiento.
Consideraciones	

3.1.3 Pruebas de interfaz de usuario.

Las pruebas de interfaz de usuario verifican la interacción del usuario con el sistema software. El objetivo de esta prueba es asegurar que la interfaz de usuario permite al usuario acceder y navegar a través de toda la funcionalidad de la aplicación. Además, la prueba de interfaz de usuario garantiza que las interfaces de usuario cumplen los estándares.

Objetivos de la prueba	Verificar los siguientes objetivos: • La navegación a través de la aplicación refleja adecuadamente las reglas de negocio y los requisitos incluyendo ventanas, campos y métodos de acceso (tabulador, movimientos del ratón y teclas de función). • Las ventanas y sus características, como menús, tamaño, posición y estado cumplen los estándares.
Técnicas	Crear o modificar pruebas para cada ventana con el objetivo de verificar la correcta navegación y su estado.
Criterios de finalización	Criterio de verificación de cada interfaz con los estándares utilizados.
Consideraciones	

3.1.4 Pruebas de sistema.

Las pruebas de sistema miden tiempos de respuesta, demanda de recursos y otros requisitos susceptibles al tiempo. El objetivo de estas pruebas es verificar y validar que los requisitos del sistema se han alcanzado.

Las pruebas de sistema normalmente se ejecutan varias veces usando cada vez una carga de trabajo diferente. La prueba inicial se debería realizar con una carga normal y la segunda prueba con una carga extrema.

Está constituida por una prueba de estrés basada en la funcionalidad del sistema bajo cargas pesadas, un gran número de repeticiones, manejo de grandes datos y demasiadas preguntas a bases de datos grandes; y además está constituida por una prueba de rendimiento diseñada para probar el rendimiento del software en tiempo de ejecución.

Objetivos de la prueba	Validar el tiempo de respuesta del sistema software para datos dentro de un rango válido que puedan producir inestabilidad bajo las condiciones siguientes: • Volumen de trabajo normal. • El peor volumen de trabajo.
Técnicas	**Estrés (Resistencia).** • Verificar cómo soporta el sistema interrupciones ejecutadas 4 ó 5 veces por segundo más de las normales. • Comprobar cómo responden las funciones de entrada al incrementar las frecuencias de datos en un orden de gran magnitud. • Comprobar cómo reacciona el sistema ante excesivas búsquedas de datos residentes en disco. **Rendimiento.** • Asegurar el rendimiento de los módulos individuales a medida que se llevan a cabo las pruebas de unidad. • Medir el rendimiento del producto en uso de memoria RAM, CPU y ancho de banda. **Mutación**. • Comprobar si el sistema es capaz de trabajar con datos incorrectos. • Asegurar que los casos de prueba diseñados reconocen los errores introducidos intencionalmente en el programa. **Pruebas de tiempo real.** • Asegurar el sincronismo entre las tareas a

	ejecutarse simultáneamente. • Asegurarse de que cada tarea se ejecuta en el tiempo previsto y comienza su ejecución en el momento preciso. • Asegurase de que no existen fallas de rendimiento cuando se ejecutan muchas interrupciones al mismo tiempo.
Criterios de finalización	Se han completado las pruebas sin ningún error y dentro de los tiempos de respuesta esperados.
Consideraciones	Ninguna.

3.1.5 Pruebas de aceptación.

Las pruebas de aceptación se ejecutan luego de culminar fase de pruebas de sistema, esta ejecución evaluará el sistema final con miras a su presentación frente al cliente. En esta prueba se verificará si el programa cumple con las especificaciones formales establecidas por el cliente.

Objetivos de la prueba	Establecer si se han satisfecho todos los requisitos establecidos por los usuarios y si el producto está listo para su explotación.
Técnicas	• Se comprueba el manual de instalación y el manual de usuario. • Se verifica si el software cumple con los requerimientos definidos por el usuario, mediante técnicas de caja negra. • Comprobar que lo que se encuentra en la aplicación se corresponde a lo descrito en el documento. Se ejecuta cada caso de uso y flujo del caso de uso con datos válidos e inválidos para verificar lo siguiente: • Cuando se utilizan datos correctos se obtienen los resultados esperados. • Cuando se utilizan datos incorrectos se obtienen los mensajes de error o advertencias adecuadas. • Cada requisito del negocio se ha implementado correctamente.
Criterios de finalización	Todas las pruebas planificadas se han ejecutado. Todos los defectos identificados se han considerado. El usuario se satisface con las especificaciones establecidas en el sistema.
Consideraciones	

3.2. Herramientas

Las siguientes herramientas se usarán para llevar a cabo el proceso de prueba:

Tipo de Prueba	Herramienta
Prueba de unidad	CxxTest
Registro de defectos	Bugzilla

4. Recursos

En esta sección se describen los recursos necesarios para realizar el proceso de prueba, sus principales responsabilidades y características.

4.1. Recursos hardware

Recurso	Cantidad	Nombre y Tipo
PC	2	Diseño de las pruebas
PC	3	Ejecución de las pruebas

4.2. Recursos software

Nombre del elemento software	Tipo y otras notas
Sistema operativo.	Plataforma en la que el software funcionará.
Herramientas de pruebas.	CxxTest y Bugzilla.

4.3. Configuración del entorno de prueba

Establecer bajo qué condiciones de hardware y características del software serán ejecutadas las pruebas. (Ejemplo computadoras Pentium 4, sistema operativo de la familia de Windows a partir de Windows 2000, etcétera).

4.4. Recursos humanos

RECURSOS HUMANOS		
Rol	**Mínimos recursos recomendados**	**Responsabilidades específicas o comentarios**
Gestor de prueba (Responsable de las pruebas)	1	Proporcionar una gestión adecuada. Responsabilidades: • Proporcionar una dirección técnica. • Exigir por el cumplimiento de acuerdos. • Adquirir los recursos apropiados. • Informar de la gestión.
Diseñador de prueba	2	Identificar, priorizar e implementar los casos de prueba. Responsabilidades: • Generar el Plan de pruebas. • Identificar técnicas apropiadas y

		herramientas para la implementación de las pruebas • Diseñar los Casos de prueba. • Definir listas de chequeo para las pruebas. • Evaluar el esfuerzo de prueba.
Probador (Tester)	3	Ejecutar las pruebas. Responsabilidades: • Ejecutar pruebas. • Recuperar los errores. • Documentar los defectos. • Registrar los resultados

5. Actividades de prueba

Las actividades del proceso de prueba para este sistema software son:

Actividad	Esfuerzo	Fecha de comienzo	Fecha de finalización
Planificación de la prueba			
Diseño de la prueba			
Ejecución de la prueba			
Evaluación de la prueba			

6. Resultados de las pruebas

Del proceso de prueba se obtienen los siguientes documentos de desarrollo de software:

Documento de desarrollo de software	Desarrollador	Revisión	Fecha
Plan de prueba			
Casos de prueba			
Informe de evaluación de pruebas			
Registro de defectos			

7. Tareas de la etapa de pruebas

Las tareas que se realizan en cada una de las actividades son:

- **Planificación de las pruebas:**
  - ✓ Identificar los requisitos para las pruebas.
  - ✓ Valorar los riesgos.
  - ✓ Desarrollar la estrategia de pruebas.
  - ✓ Identificar los recursos necesarios para realizar las pruebas.
  - ✓ Planificar la temporización.
  - ✓ Generar el Plan de pruebas.

- **Diseño de las pruebas:**
  - ✓ Análisis de la carga de trabajo.
  - ✓ Desarrollo de las pruebas.
  - ✓ Identificar y describir los casos de prueba.

- **Implementación de las pruebas:**
  - ✓ Establecer el entorno de prueba.
  - ✓ Desarrollar las clases de prueba y los datos de prueba.

- **Ejecución de las pruebas:**
  - ✓ Ejecutar los casos de prueba.
  - ✓ Evaluar la ejecución del proceso de prueba.
  - ✓ Verificar los resultados.
  - ✓ Investigar los resultados no esperados.
  - ✓ Registrar los defectos.

- **Evaluación de las pruebas:**
  - ✓ Evaluar la cobertura de los casos de prueba.
  - ✓ Evaluar la cobertura del código.
  - ✓ Analizar los defectos.
  - ✓ Determinar si se han alcanzado los criterios de las pruebas.
  - ✓ Crear los informes de evaluación de las pruebas.

Importancia de cálculo del esfuerzo de la prueba.

Estudios realizados han concordado en que la prueba exhaustiva es impracticable y en que nunca una persona podrá saber exactamente si ha podido eliminar todos los errores de un software, pero cómo saber entonces cuando terminar de probar. Para esto, un método muy efectivo es la estimación de la prueba, esto permitirá establecer cuáles son las áreas del programa más propensas a errores y podrá determinar el promedio de tiempo que se invertirá en cada prueba.

Muchas organizaciones estiman el esfuerzo de la prueba basadas solamente en el número de los casos de la prueba que se ejecutarán (esfuerzo de la prueba = número de casos de prueba * tiempo promedio en la ejecución de un caso de prueba).

El dilema de este método es que falta la causa principal de la prueba: los defectos. Una mejor forma de calcular el esfuerzo está basada en calcular un pronóstico de los defectos que se encontrarán en el producto durante la prueba más el tiempo de ejecución de todos los casos de prueba (esfuerzo de la prueba = (número previsto de defectos * tiempo promedio para hallar y depurar un error) + (número de casos de prueba * tiempo promedio para ejecutar un caso de prueba).

Para utilizar esta técnica se necesita un mecanismo para estimar el número previsto de defectos en el producto y para determinar el coste medio de encontrar y de depurar un defecto durante la prueba.

Hay solamente tres tipos de medidas requeridas para administrar los procesos de prueba: esfuerzo, tamaño y defectos. El "esfuerzo" es el esfuerzo realizado individualmente por cada uno de los participantes de la prueba para encontrar y para depurar los defectos encontrados en prueba. El "tamaño" es el tamaño del producto que es probado, medido frecuentemente en líneas del código (LOC, “Lines Of Code”). El "defecto" es la cantidad, el tipo, la fase en que se introduce, la fase en que se depura, y la descripción del defecto.

Usando estas tres medidas, se obtienen todos los datos que se necesitan para estimar el número de los defectos que se espera encontrar en la prueba, así como el coste medio para encontrarlos y para depurarlos.

Para determinar el tiempo promedio para encontrar y depurar un error se toma el esfuerzo total ocupado en un defecto dividido entre el número total de defectos que se encontraron en esa fase de prueba. Si se asume que todos los defectos encontrados en esa fase también fueron depurados, entonces el tiempo promedio para encontrar y depurar un defecto sería la suma del tiempo empleado por todos los probadores para encontrar un defecto y depurarlo dividido por el número total de defectos encontrados y depurados.

¿Qué sucede si han sido introducidos más defectos de los que se estiman y el esfuerzo estimado no es el adecuado para ejecutar la prueba?

Por ejemplo, si se espera encontrar 20 defectos en una fase de prueba (unidad, integración, sistema o aceptación), se puede orientar un indicio alrededor de este valor, dando por resultado una expectativa de 16 a 24 defectos encontrados por KLOC (por cada mil líneas de código). Si el número de defectos encontrados excede el límite superior, entonces se toman medidas para determinar si el equipo de la prueba hizo una revisión excepcionalmente buena, o si el producto era más defectuoso de lo esperado.

Similarmente, si es el número de defectos encontrados era menos que el esperado, entonces se determina si el equipo de prueba hizo una revisión pobre, o si el producto posee una calidad particularmente alta.

Este método sería, además una forma de evaluar el desempeño de ambos equipo, el de desarrollo y el de prueba.

Los seres humanos, tienden a ser muy repetitivos en las acciones. Como consecuencia, cuando se incurre en una equivocación una vez, es muy probable que se haya incurrido en esta misma equivocación en otra parte. Esta característica humana se puede utilizar como una ventaja, pues una vez encontrado un defecto en la prueba, esto se debe utilizar como base para buscar ocurrencias adicionales del mismo defecto en otras partes en el producto.

Los estudios realizados por diferentes especialistas detectan que los defectos tienden a aparecer en racimos. Un estudio realizado demuestra que de 1200 módulos analizados el 20 por ciento de estos contenían el 70 por ciento de los errores y el 50 por ciento de los defectos fueron encontrados en el 10 por ciento de lo módulos. Esto sugiere que si los probadores son capaces de identificar y depurar los módulos más propensos a errores, entonces, aproximadamente estarían eliminando el 70 por ciento de los errores que contiene el producto.

Si en la revisión de un módulo en particular se encuentran muchos defectos, entonces es posible que sea más beneficioso en costo codificarlo nuevamente que seguir probándolo.

Por ejemplo se asume un módulo X, cuyo tamaño es aproximadamente 1 KLOC (o mil líneas de código). Si según el plan y la estimación de la prueba, se espera un total de 4 a 5 defectos para un módulo de este tamaño en la fase en que se encuentra; se han ejecutado el 10 por ciento de los casos de prueba de esa fase, y se han encontrado 3 defectos hasta ese momento. Se pude asumir que puesto que el 10 por ciento de los casos de prueba se han ejecutado, estos 3 defectos encontrados representan el 10 por ciento de los defectos que serán encontrados en esa fase. Por tanto, se calcula que el módulo X tiene probablemente un total de 30 defectos, comparado con lo esperado en el plan (de 4 a 5). Como probador, ¿qué haría sobre este módulo, continuar probándolo o enviarlo al equipo de desarrollo para que se codifique nuevamente?

Acción 1: Continúe probándolo.

Si calculando el esfuerzo de la prueba se estimara que este módulo tiene un coste de 5 horas por defecto, entonces hasta el momento ha costado 15 horas. Si se continúa probando para encontrar los restantes 27 defectos costarían probablemente un total de 135 horas.

Acción 2: Codificarlo nuevamente.

Si se asume como promedio que un programador puede codificar 20 LOC por hora, entonces reescribir el módulo tendría un coste de 50 horas (1000 LOC * 20 LOC / hora = 50 horas) y dada la estimación de la cantidad de defectos a encontrar (de 4 a 5), el coste de probar el módulo sería aproximadamente 25 horas (5 defectos * 5 horas = 25 horas), por lo tanto el coste estimado de sobrescribir el módulo sería el coste total sería el coste de codificarlo nuevamente más el coste de probarlo (50 horas + 25 horas = 75 horas).

En este ejemplo, codificar el módulo y probarlo nuevamente rendirían un ahorro neto de 60 horas sobre la decisión de continuar probándolo. Si se calcula el costo total, el continuar probándolo se valoraría en un costo de las 50 horas de confección del módulo más las 135 horas invertidas en probarlo lo que redundaría en un costo de 185 horas.

Llevar a cabo de forma consecuente y organizada la puesta en práctica de un plan de prueba similar al expuesto y realizar métricas y estimaciones de las pruebas con conciencia de los beneficios que esto conlleva, servirá de guía a los proyectos de la Facultad 5 tanto en organización y como en coste, y llegará a ser de gran importancia para elevar la calidad y la productividad en estos proyectos.

3.4 Propuesta de técnicas de pruebas a utilizar.

Para cada procedimiento para la ejecución de los tipos de pruebas establecidos el equipo de desarrollo debe entregar al equipo de prueba un expediente que contiene:

- El producto software a probar.
- El documento de especificación de casos de uso.
- El manual de usuario y de instalación.
- Un glosario de términos.
- En caso de no estar incluidos en la especificación de casos de uso, un documento que contiene los requerimientos y a qué caso de uso corresponde cada uno de ellos.
- Los requerimientos mínimos de hardware y software para que la aplicación funcione, debieron ser entregados con antelación, para crear las condiciones requeridas.

- Una identificación de los riesgos asociados al proyecto y al historial de defectos.

3.4.1 Pruebas de unidad.

La prueba de unidad analiza el resultado de ejecutar cada método bajo distintas condiciones o distintos valores de los parámetros de entrada y de los atributos de la clase. Además analiza la evolución del estado de la clase (valor de sus atributos) bajo distintas combinaciones de llamadas a sus métodos.

1) Debe comenzar inmediatamente se vayan construyendo los módulos y serán desarrolladas por las mismas personas que desarrollaron roles de programadores.
2) Deben ser aplicadas a todo el código.
3) Se recomienda utilizar alguna herramienta de prueba para la automatización de las mismas, preferiblemente CxxTest, seleccionada en el proyecto SCADA mediante un estudio de las herramientas disponibles.
4) Deben aplicarse métodos de caja negra para probar la funcionalidad de los módulos (clases, procedimientos, funciones, métodos, objetos, o componentes).
5) Identificar las clases de equivalencia (de datos válidos y de datos no válidos o erróneos) (Ver Capítulo 2).
6) Identificar casos de prueba a partir de clases de equivalencia (Ver Capítulo 2).
7) Explorar las condiciones límites de un programa mediante la técnica de análisis de valores límites (Ver Capítulo 2).
8) Utilizar la técnica de conjetura de errores para determinar los errores que más se repiten y en función de esta elaborar los casos de prueba (Ver Capítulo 2).
9) Ejecutar los casos generados hasta el momento y analizar la cobertura obtenida para determinar el cumplimiento de los objetivos de prueba trazados.
10) Los módulos que no estén pasando las pruebas deben ser informados al equipo de desarrollo y una vez depurados estos errores deben ejecutarse nuevamente los casos de prueba para verificar que la reparación no ha causado otros errores.
11) Delimitar los errores y sus posibles causas, de este modo el probador podrá asegurarse de que no se repite el error en otras secciones del programa y podrá asegurarse también de que los cambios no introducen un comportamiento no deseado o errores adicionales.
12) Haciendo referencia a los principios de las pruebas enunciados por Myers (Ver capítulo I), se recomienda que si se detectan muchos fallos en un módulo, lo mejor sería desecharlo, diseñarlo y codificarlo nuevamente.

13) Los resultados obtenidos con la ejecución de las pruebas deben ser documentados debidamente en el registro de defectos.

Se exhorta a que, para la totalidad de los proyectos productivos de la Facultad 5, se exija la aplicación de pruebas de unidad a todos los productos realizados como requisito indispensable para la liberación de un entregable y/o la liberación del producto final.

3.4.2 Pruebas de integración.

1) Se debe crear un grupo independiente de 2 a 3 estudiantes con conocimientos básicos en pruebas, con un profesor responsable de la labor de los mismos. Este grupo se encargará de diseñar, guiar y dirigir las pruebas de integración.
2) El probador debe identificar los módulos críticos, los cuales serán probados lo antes posible. Un módulo crítico es aquel que tiene una o más de las siguientes características: (1) está dirigido a varios requisitos del software; (2) tiene un mayor nivel de control (está relativamente alto en la estructura del programa); (3) es complejo o propenso a errores o (4) tiene unos requisitos de rendimiento muy definidos. (PRESSMAN 2005)
3) Se combinan los módulos de bajo nivel en grupos que realicen una función o subfunción específica (o quizás si no es necesario, individualmente) de este modo se reduce el número de pasos de integración.
4) Se escribe para cada grupo un módulo impulsor o conductor, esto permite simular la llamada a los módulos, introducir datos de prueba y recoger resultados.
5) Se prueba cada grupo mediante su módulo impulsor.
6) Se eliminan los módulos impulsores y se sustituyen por módulos de nivel superior en la jerarquía.
7) Los errores encontrados son documentados e informados diariamente al equipo de desarrollo para su corrección.
8) El equipo desarrollo debe corregir los errores e informar al equipo de pruebas los cambios realizados.
9) Conjuntamente con el proceso de integración se deben aplicar las llamadas pruebas de regresión, para asegurar que los módulos causantes de fallas fueron efectivamente corregidos y que no se introdujeron nuevos errores al tratar de solucionar alguno. Estas pruebas de regresión consisten simplemente en aplicar el mismo proceso de pruebas planificado originalmente, sólo que se repiten cada vez que se identifica un resultado erróneo y se le da solución.
10) Las pruebas no terminarán hasta que no se hayan integrado todos los módulos del programa.

11) El resultado de las pruebas será documentado en el Documento de Evaluación de pruebas y los errores obtenidos deben ser recogidos en el registro de defectos.

3.4.3 Pruebas de sistema.

Una vez desarrolladas las pruebas básicas al sistema completo, se debe poner a trabajar el software bajo las condiciones reales.

Es en este punto donde se pasa a la etapa de pruebas del sistema. Esta etapa realmente está constituida por una serie de pruebas diferentes cuyo objetivo primordial es verificar profundamente el sistema global. Aunque cada prueba tiene un propósito distinto, todas convergen en la verificación de que se hayan integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas. Las pruebas del sistema que se deben considerar son:

3.4.3.1 Prueba de estrés.

Este tipo de prueba es diseñada para probar programas con situaciones anormales, como bajos recursos o competencias entre estos. Los objetivos de la prueba de estrés son encontrar los límites del sistema y asegurar que tras un fallo en el sistema, se recupera sin causar graves problemas. Sin este tipo de testeo es imposible saber qué ocurre cuando la aplicación funciona bajo cargas elevadas. ¿El tiempo de respuesta seria inferior? ¿En qué punto la aplicación fallará? Se buscan combinaciones de datos dentro de un rango válido que puedan producir inestabilidad o procesos incorrectos para así medir la capacidad que tiene la aplicación para continuar a pesar de la falla.

Este tipo de prueba es importante para los proyectos productivos de la Facultad 5, ya que software como el Simulador de Medicina Quirúrgica, el Simulador de Tiro y el Simulador de Auto, requieren que la computadora procese grandes cantidades de información y realice muchos cálculos matemáticos en tiempo real.

1) El equipo de pruebas debe familiarizarse con la aplicación a la que se le realizarán las pruebas, la arquitectura, las soluciones elegidas y sus requisitos de rendimiento.
2) Se debe desarrollar una estrategia de casos de prueba basados en un análisis de riesgos de aplicación.
3) El probador debe someter la aplicación a una carga elevada de valores numéricos complejos y analizar el comportamiento del sistema ante tales condiciones.

4) Se debe verificar además como el sistema responde a un elevado número de peticiones. El probador debe tener en cuenta si:
    - ✓ ¿Es capaz de trabajar con un disco al 90%?
    - ✓ ¿En qué porcentaje se modifica el tiempo de respuesta?
    - ✓ ¿En algún momento deja de operar el sistema?
    - ✓ ¿Cuántos accesos concurrentes soporta?
    - ✓ ¿Cuántos datos podrá procesar?
    - ✓ ¿Queda el sistema en un estado indeterminado o saturado completamente?
5) Verificar que todo sigue funcionando correctamente luego de un tiempo prudencial, ya que a veces sucede que si bien se supera la prueba con éxito, el sistema queda funcionando con mayor lentitud porque algunas variables no se restablecen a los valores normales. En esos casos, se deben repetir las pruebas monitoreando las variables más críticas.
6) El probador debe forzar cada componente de forma aislada más de lo que la aplicación podría experimentar en condiciones normales y observar si hay problemas evidentes.
7) Después de forzar cada componente individual, deberá someter a una situación de estrés a toda la aplicación con todos sus componentes y servicios y observar los resultados.
8) Es necesario que conozca los recorridos codificados y las situaciones a las que se enfrentan los usuarios, que comprenda lo que intentan hacer estos y que identifique todas las maneras en las que el usuario se mueve por la aplicación.
9) El probador debe analizar las interacciones con otras estructuras de datos, procesos y servicios tanto de los componentes internos como de otros servicios externos de la aplicación.
10) Se debe redactar un documento con los problemas detectados y los casos de prueba ejecutados.

3.4.3.2 Prueba de rendimiento.

Este tipo de prueba suele ir ligada a la prueba de resistencia o estrés y es diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

La prueba de rendimiento puede influir en el mejoramiento de la calidad de los productos de Entornos virtuales de la Facultad 5, ya que llevaría a la detección de defectos en la accesibilidad a gran cantidad de recursos que podría imposibilitar o retardar el desarrollo de los procesos en tiempo real.

1) El probador puede asegurar el rendimiento del sistema y a menudo pueden usarse instrumentos de software y/o hardware para medir la utilización de recursos.

2) El probador debe poner el sistema bajo elevada carga y medir el uso que hace el mismo de la RAM (Random-Access Memory) y CPU (Central Processing Unit).
3) Verificar cómo reacciona el sistema, bajo elevada carga, ante lecturas completas de tablas de la base de datos.
4) Ejecutar tareas durante varias horas seguidas para verificar si se afecta la memoria o el rendimiento del sistema al estar bajo las mismas condiciones por tiempo indefinido.
5) Redactar un documento con los problemas detectados y los casos de prueba ejecutados.
6) Recoger los errores detectados en el registro de defectos.

3.4.3.3 Prueba de mutación.

Es una técnica que consiste en alterar ligeramente el sistema bajo pruebas (introduciendo errores) para averiguar si la ejecución de pruebas es capaz de detectarlo. Si no, sería conveniente introducir nuevas pruebas.

Esta técnica es de gran importancia para los proyectos de la Facultad 5, ya que permitirá comprobar si los sistemas desarrollados pueden trabajar con datos incorrectos, lo que dará una medida de las dificultades que aún subsisten en el proceso. Además, permitirá comprobar el nivel de conocimientos y la calidad del trabajo desarrollado por los encargados de realizar las pruebas. En estudios analizados, se ha afirmado que esta técnica puede ahorrar hasta el 50% del coste de las pruebas sin degradar excesivamente su calidad.

1) Inicialmente se establece el conjunto de mutantes, el conjunto de casos de prueba y se define un umbral que representa el porcentaje mínimo de mutantes muertos que debe alcanzarse.
2) El probador deberá tener una copia del programa original e introducirle pequeños cambios, por ejemplo sintácticos, como un + por un *.
3) El probador ejecutará los casos de prueba correspondientes a la parte de código que ha sido modificada y calculará el porcentaje de mutantes muertos. Si el mutante está *vivo* (no ha sido reconocido el error en las pruebas) puede significar que sea equivalente al programa inicial o que no haya sido especificado un caso de prueba lo suficientemente bueno como para detectar el fallo, por lo que el probador deberá asegurase de que el ha realizado una prueba eficiente.
4) Si no se ha alcanzado el umbral previamente definido, se eliminan los casos de prueba que no han matado mutantes y se generan casos de prueba nuevos, especialmente dirigidos a matar los mutantes que han permanecido vivos.

5) Los pasos serán repetidos hasta que el por ciento de mutantes *muertos* (errores identificados por las pruebas) esté sobre el umbral previamente definido.
6) Redactar un documento con los problemas detectados y los casos de prueba ejecutados.

Los resultados obtenidos de cada una de estas pruebas deberán ser informados preferiblemente al equipo de desarrollo para que se tomen las medidas pertinentes, este a su vez deberá informar al equipo de pruebas las correcciones realizadas y para su análisis y puesta en prueba nuevamente si es necesario.

3.4.3.4 Pruebas para sistemas de tiempo real.

Las principales características que poseen algunas de las aplicaciones desarrolladas en la Facultad 5 es que deben ejecutarse en tiempo real. Esto ofrece una gran dificultad en el momento de realizar las pruebas a estos productos software, ya que las técnicas y casos de prueba deben ser muy específicamente enfocados a las características más críticas, como el tiempo, la ejecución de muchos procedimientos paralelamente y la relación que existe entre el software de tiempo real y su entorno de hardware, lo que también puede introducir problemas en la prueba.

1) Para realizar la prueba de tareas asíncronas es necesario confeccionar un diagrama de estado-transición donde se refleja una lista de todas las posibles interrupciones, este diagrama debe reflejar las interrupciones que deben ejecutarse al mismo tiempo y las que son consecuencia de la ejecución de otras.
2) Deben diseñarse casos de prueba que evalúen si se asignaron adecuadamente las prioridades para cada interrupción.
3) Deben diseñarse casos de prueba que evalúen si cada interrupción se realiza en el tiempo predeterminado para esta.
4) Deben diseñarse casos de prueba que evalúen si un gran volumen de interrupciones realizadas simultáneamente crean problemas de funcionamiento o rendimiento.
5) Cada dato generado de la ejecución de los casos de prueba debe ser debidamente analizado para establecer si existen algún rango diferente de tiempo o sincronismo del establecido para cada interrupción.
6) Cada defecto debe ser debidamente documentado en el registro de defectos e informado al equipo de desarrollo para su corrección.

3.4.4 Pruebas de aceptación.

Después de que el software haya pasado todas las pruebas de sistema y se hallan hecho las correcciones de errores, los usuarios adquieren un papel más activo en el proceso de prueba, pues requieren determinar por ellos mismos las condiciones del producto que pagarán. Para esto trabajan en conjunto con los probadores y desarrolladores determinando el nivel de aceptación del producto.

1) Se debe crear un grupo independiente de 2 a 3 estudiantes con conocimientos básicos de pruebas, con un profesor responsable de la labor de los mismos. Este grupo se encargará de diseñar, guiar y dirigir las pruebas de aceptación.
2) El proceso comienza cuando se le da el acabado a un producto en determinado proyecto de la facultad 5 y el equipo de desarrollo y el líder de proyecto pone el programa en manos del grupo de pruebas para realizar las pruebas de aceptación al mismo, una vez hayan sido desarrolladas como mínimo las pruebas de unidad.
3) El equipo de desarrollo y el grupo de pruebas se reúnen y planifican los cronogramas de entrega y liberación, los cuales serán debidamente firmados por ambas partes para dejar constancia de la toma de acuerdos.
4) Se realiza el plan por el cual se van a guiar las pruebas de aceptación, en el que queda recogido todo lo referente a las pruebas tanto por parte del equipo de pruebas (encargado de informar las deficiencias encontradas) como por parte de los desarrolladores (encargados de corregir las deficiencias e informar los resultados de la corrección).

Se propone que se utilice el siguiente procedimiento para las pruebas:

5) Antes de comenzar las pruebas se crea un expediente del producto el cual incluye todo lo entregado por los desarrolladores y al cual se le irá incluyendo todo lo que se genere durante el proceso de pruebas.
6) Se realizan las pruebas al manual de instalación (el grupo de pruebas instalará un producto que nunca ha visto guiándose solo por el manual de instalación, si no puede hacerlo dejará claro que el manual no cuenta con todo lo necesario para la comprensión del funcionamiento de la instalación del producto, lo cual deberá ser reportado al equipo desarrollo).
7) Una vez instalado el software, se comenzarán las pruebas al manual de usuario (el equipo de pruebas debe ser capaz de moverse por una aplicación que no conoce y comprenderla, así como saber qué opción ejecutar y qué hacer, guiándose por este manual, en caso de no comprender se deberá informar al equipo de desarrollo para que conforme nuevamente o rediseñe el manual de usuario).

8) Se revisarán además, otros detalles como ortografía y redacción. Aunque algo esté bien redactado puede ser perfeccionado para un mejor entendimiento del usuario final (esta revisión puede ser ejecutada por el equipo de prueba y/o por el usuario o cliente).
9) Se realizan las pruebas funcionales mediante la técnica de caja negra:
   - Se verifica si el software cumple con los requerimientos definidos por el usuario (para esto se diseñan los casos de pruebas, los mismos se realizan a partir de los casos de uso, de forma tal que al final del diseño se tendrá un caso de prueba por cada caso de uso existente).
   - Se prueba el documento de especificación de caso de uso
   - Se comprueba que lo que se encuentra en la aplicación se corresponde a lo descrito en el documento.

(Se propone que estas pruebas se pueden hacer de forma paralela, se puede designar un grupo de estudiantes para hacer las pruebas a los manuales, mientras que otro grupo diseña los casos de pruebas).

10) Si es un software complejo, el cual esta compuesto por varios módulos que se integraran al final para lograr el producto completo, se probarán primero los módulos por separados y luego se diseñará un caso de prueba de integración para probar el software como un todo, para esto el equipo de desarrollo deberá entregar con el expediente del producto un documento donde exponga como los datos manejados de un modulo afectan a otros. En caso de productos menos complejos esto no es necesario.
11) Una vez diseñados los casos de pruebas el equipo de calidad comienza a aplicar los mismos, en muchas ocasiones a medida que se diseñan los casos de pruebas el software es probado casi en su totalidad y se recogen las no conformidades que aparecen, esto ahorra tiempo, proporciona una mayor cantidad de pruebas y permite una mayor detección de errores y defectos.
12) Se diseña y aplica un caso de prueba del sistema, que no es más que un caso de prueba que sigue un flujo básico, ya que con las pruebas modulares se puede perder la visión general del producto, este caso probará todos los flujos alternativos. Con el mismo se puede simular el proceso en vivo como mismo debe ocurrir cuando el usuario final lo utilice.
13) Aplicadas las pruebas, se obtiene como resultado el documento de no conformidades, en el mismo están todos los defectos, errores y sugerencias realizadas al equipo de desarrollo, así como también queda documentado en el caso de prueba el resultado de todas las pruebas aplicadas.
14) El equipo de calidad entrega estos resultados al equipo de desarrollo, el cual trabajara en función de estos para perfeccionar al máximo el software.

15) Se crea el laboratorio de pruebas donde el usuario final probara el software. Estas pruebas corresponden a pruebas de tipo alfa, están guiadas por miembros del equipo de pruebas y contarán con la presencia de un miembro del equipo de desarrollo, ya que pueden surgir dudas, o el usuario puede querer saber algo específico, que solo el desarrollador puede responder.
16) 19- Para probar, el usuario final se guiará por los casos de pruebas ya refinados, necesitará el documento según las características del software con los datos que contiene la base de datos, aplicará el caso de prueba del sistema y las listas de chequeo.
17) Al final de cada día se debe realizar una reunión, donde se expondrán los resultados del día, así como cualquier dificultad o inconveniente que haya podido surgir.
18) Obtenidos todos los señalamientos hechos diariamente por el usuario se confecciona un documento con los mismos (Documento de evaluación de las pruebas). También se realizará el sumario de las listas de chequeo, donde se obtiene un resultado cuantitativo de las mismas.
19) Se realiza una reunión con los especialistas funcionales, el equipo de pruebas y el representante del equipo de desarrollo para discutir y firmar el "Documento de Evaluación de Pruebas" como constancia de que todas las partes están conformes con los señalamientos realizados y con las solicitudes de cambio, así como también se firmará el sumario de las listas de chequeo.
20) Los desarrolladores trabajarán sobre las solicitudes de cambio solicitadas hasta una segunda presentación del software al usuario final. Este proceso se repetirá las veces que sea necesario con la diferencia de que ya no se diseñaran nuevos casos de pruebas, el equipo de calidad refinará los ya realizados y los aplicará, así como verificará si estas solicitudes fueron llevadas a cabo y redactará un documento resumen de estas.
21) En la próxima presentación al usuario final, el mismo aplicará nuevamente los casos de pruebas, revisará las solicitudes de cambio para ver si fueron modificadas y volverá aplicar las listas de chequeo, el resultado de estas últimas será comparado con el de las aplicadas anteriormente.
22) Para cada solicitud de cambio los desarrolladores deben incluir qué solución fue dada y qué partes de la aplicación fueron afectadas con los cambios lo que incluye requerimientos y caso de uso; esto se hace en busca de nuevos errores incluidos.
23) Este proceso se repetirá cuantas veces sea necesario, y por lo que se establecerá una retroalimentación entre los dos equipos de trabajo en cuanto a los defectos encontrados. Cuando se entreguen versiones nuevas que solucionen errores señalados por el equipo de calidad, el equipo de desarrollo deberá entregar un documento adjunto a la nueva versión, que especifique exactamente qué parte se cambió en cada artefacto, para evitar volver a indicar errores ya

señalados, y aún no corregidos, este documento puede ser el mismo documento de no conformidades.

24) Al terminar el proceso se obtendrá el expediente actualizado con todo lo correspondiente al proceso de pruebas, así quedara archivado un historial sobre las pruebas realizadas, este incluirá un informe sobre las pruebas realizadas y recogerá todo lo sucedido en las diferentes etapas del proceso de pruebas.

Se recomienda se utilice una lista de chequeo (Anexo 12) para determinar que se cumpla correctamente el proceso de pruebas en el proyecto.

La idea propuesta por diversos investigadores y profesionales de cómo transformar la tarea de pruebas en un proceso de costes efectivos, ha crecido equitativamente respecto a la importancia de dicha tarea. La evolución de los modelos específicos de proceso de pruebas aumenta satisfactoriamente la calidad de dichos procesos y los coloca en una posición relevante dentro de la ingeniería de software.

Es por ello que el presente trabajo recomienda, que una vez establecidas las características fundamentales del proceso de pruebas en un proyecto se investigue y se ponga en práctica un proceso de mejora para la prueba, este trabajo propone se profundice más en el TMM o Modelo de Madurez de Prueba (Ver Capítulo II); esto perfeccionaría la calidad en los procesos de pruebas y a su vez los de desarrollo de software.

CONSIDERACIONES FINALES

A partir de la información recopilada en este capítulo sobre el desarrollo de pruebas en los proyectos productivos de la Facultad 5 de la UCI se ha llegado a las siguientes conclusiones:

- Existen deficiencias en la planificación y ejecución de las pruebas a los productos.
- No se tiene definido un proceso de desarrollo de pruebas consecuente.
- No se aplican herramientas para la automatización de pruebas.
- Algunos proyectos ejecutan solo las pruebas de unidad y otros ni siquiera estas.
- Es necesaria la aplicación de medidas en el marco de la calidad de software tales como:
    - La aplicación de técnicas de pruebas con el objetivo principal de satisfacer los requisitos establecidos por los clientes y de conseguir la prevención y/o detección de posibles errores con poca cantidad de tiempo y esfuerzo.
    - La aplicación de planes de pruebas para mejorar la organización del trabajo y no dejar todo para el último momento.
    - Es necesario realizar estimaciones y métricas para la prueba que permitan divisar las áreas del programa que deben tener mayor seguimiento para el desarrollo de las pruebas.

CONCLUSIONES GENERALES

- La documentación analizada sobre la línea de trabajo de pruebas de software facilitó la comprensión de este proceso.
- Las técnicas y/o estrategias de pruebas de software aplicadas a proyectos productivos desarrolladores de software deben ser adecuadas o modeladas para estos proyectos.
- En dependencia de las debilidades encontradas en los proyectos productivos de la Facultad 5, aplicar técnicas de pruebas que posibiliten la obtención de productos con la calidad y en los plazos de tiempo requerido por usuarios y clientes.

Con el estudio realizado y la propuesta de técnicas de prueba a utilizar se le da cumplimiento a objetivo planteado: Determinar las técnicas de pruebas de software existentes en el mundo para su aplicación en la Facultad 5 de la Universidad de las Ciencias Informáticas.

RECOMENDACIONES

Se recomienda:

- Poner a prueba la selección de técnicas de pruebas propuestas en los proyectos productivos “Entornos Virtuales” de la Facultad 5, para determinar el nivel de mejora en la calidad de los procesos desarrollados a partir de la aplicación de estas técnicas.
- Las técnicas de prueba de software seleccionadas deben ser ejecutadas con una correcta planificación del proceso de pruebas y llevadas a cabo de forma consecuente con métricas, estadísticas de resultados y documentación de la ejecución, con el objetivo de elevar, con cada experiencia, el proceso de mejoramiento continuo de pruebas.
- Realizar un análisis más profundo de técnicas de prueba que estén enfocadas al perfil de “Entornos Virtuales” para su aplicación en los proyectos de la facultad 5.
- Realizar un estudio de las herramientas automatizadas de prueba para su aplicación en los proyectos productivos de la facultad, ya que es un componente significativo del proceso de pruebas y conllevaría al ahorro de tiempo, a reducir el esfuerzo requerido, y a aumentar la calidad del producto.

Se exhorta:

- Conformar un grupo independiente de pruebas en la facultad que sea eficiente, y que, con la aplicación de un proceso de formación especializado en el desarrollo de pruebas, sea capaz de definir expectativas de defectos basadas en las experiencias que vayan obteniendo.
- Capacitar a todo el personal del proyecto respecto a la importancia de la Calidad de Software y realizar encuestas periódicamente a los jefes de proyectos para tener una idea clara de cómo se están llevando a cabo las pruebas de software y cuáles han sido los resultados obtenidos de su aplicación.
- Evaluar y controlar periódicamente los procesos de pruebas y sus productos o resultados asociados.

REFERENCIAS BIBLIOGRÁFICAS

1. BURNSTEIN, I. Practical software testing. New York, Springer-Verlag New York, Inc., 2003.
2. MYERS, G. J. The Art of Software Testing. Hoboken, New Jersey., John Wiley & Sons, Inc., 2004.
3. PRESSMAN, R. S. Capítulo 17: Técnicas de Prueba del Software. en: Ingeniería del Software. Quinta Edición. La Habana, Félix Varela, 2005. Parte 1: 601.p.
4. USAOLA, M. P. Mantenimiento Avanzado de Sistemas de Información, Departamento de Tecnologías y Sistemas de Información., 2006. [2007]. Disponible en: http://alarcos.inf-cr.uclm.es/doc/masi/doc/lec/parte5/polo-apuntesp5.pdf

BIBLIOGRAFÍA CONSULTADA

5. CARRILLO, L. V. L. Caracterización de la Prueba de Software: Clasificación y Técnicas, 2005. 1.

---. El Contexto de la Prueba de Software: Probar para Incrementar la Calidad
2005.

---. El Proceso de la Prueba de Software 1: Lenguajes de Definición de Procesos, 2005. 2.

---. El Proceso de la Prueba de Software 2: Lenguajes de Definición de Procesos, 2006. 2.

---. Fundamentos de la Prueba de Software: Conceptos, Justificación y Alcance, 2005. 2.

---. La Industria de Prueba de Software en México, 2006. 1.

5. HUTCHESON, M. L. Software Testing Fundamentals: Methods and Metrics. Indiana, Wiley Publishing Inc., Indianapolis, Indiana., 2003. p. 047143020X
6. IVAR JACOBSON, G. B., JAMES RUMBAUGH. Capítulo 11: PRUEBA. en: El proceso unificado de desarrollo de software. La Habana, Félix Varela, 2004. 1: 435.p.
7. MELO, M. L. La ingeniería de software automatiza su desarrollo. Aumenta la concienciación ante la idoneidad de apostar por el outsourcing de pruebas.: COMPUTEWORDL Tecnología, 2006. Nº:1083: 8.
8. MOLPECERES, A. Procesos de desarrollo: RUP, XP y FDD., 2002. [2006]. Disponible en: http://www.willydev.net/descargas/articulos/general/cualxpfddrup. PDF
9. ONLINE, C. D. Computer Dictionary Online, 2007. [Disponible en: http://www.computer-dictionary-online.org/
10. PRESSMAN, R. S. Capítulo 1: El Producto. en: Ingeniería del Software. Quinta Edición. La Habana, Félix Varela, 2005. Parte 1: 601.p.

11. QAGROUPS. QAGroups, Empresa de pruebas de software., 2006. [Disponible en: http://www.qagroup.com.mx/
12. WIKIPEDIA. Pruebas de Software, Wikimedia Foundation, Inc., 2007. [Disponible en: http://es.wikipedia.org/wiki/Prueba_de_software
13. GEORGE, E. Managing Your Way through the Integration and Test Black Hole, PS&J Software Six Sigma, 2004. [2007]. Disponible en: http://www.methodsandtools.com/archive/archive.php?id=13
14. LAPLANTE, P. A. Real-Time Systems, Design and Analysis, IEEE Press Editorial Board, A JOHN WILEY & SONS, INC., PUBLICATION, 2004.

ANEXOS

Anexo 1: Lista de errores encontrados.

Código del defecto	Localización	Descripción del error encontrado

Anexo 2: Plan de Pruebas.

<table>
<tr><td colspan="4">Fecha: <Fecha en que se hace el Documento></td></tr>
<tr><td colspan="4">Versión: <Número de Versión del Documento></td></tr>
<tr><td colspan="4">Responsables: <Nombre de las Personas que realizan este documento></td></tr>
<tr><td colspan="4">1. Requerimientos funcionales:</td></tr>
<tr><td colspan="4">CASOS DE USO</td></tr>
<tr><td colspan="2"><Caso de Uso1></td><td colspan="2"><Aquí se debe indicar brevemente en que consiste el CU, y en las secciones posteriores incluir la información necesaria para verificar su funcionamiento></td></tr>
<tr><td colspan="2">Escenarios</td><td colspan="2"><Aquí se plantea la matriz de escenarios correspondiente al Caso de Uso 1, esto es todas las posibles combinaciones del flujo normal de los eventos y los flujos alternativos></td></tr>
<tr><td>Nombre de escenario</td><td>Flujo inicial</td><td>Flujo alternativo</td><td>Flujo alternativo</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">Planilla de Condiciones</td><td colspan="2"><Aquí se plantea la planilla de Condiciones, esto es las condiciones que causan que se ejecute un escenario especifico dentro de los posibles escenarios identificados para el Caso de Uso1, indicando el resultado esperado de ejercitar ese Escenario-Condición y la referencia a los casos de prueba de la planilla CasosPruebaConDatos.xls para ese CU que se corresponden con ese Escenario-Condición></td></tr>
<tr><td colspan="4">ENTIDADES</td></tr>
<tr><td colspan="2">Nombre de la Entidad</td><td colspan="2">Estados</td></tr>
</table>

< Se identifica el nombre de cada entidad relevante existente en el sistema>	<Aquí se indican los estados que puede tomar la entidad1 a lo largo de su ciclo de vida>
CICLOS DE FUNCIONALIDAD	
<Ciclo1>	
Casos de Uso y Entidades	<Se indican los casos de uso que se encadenan en este Ciclo, las entidades involucradas y los estados de las mismas>
Caso de Uso	Entidades/Estados
Casos de Prueba	
<Aquí se plantean las pruebas para ese ciclo, dado que una prueba encadena un escenario de cada caso de uso, se debe dar la referencia al Escenario del Caso de Uso, la referencia a los datos que prueban ese escenario y las entidades involucradas en cada prueba y sus estados, para el caso de Prueba del ciclo debe darse el resultado esperado>	

Caso del Ciclo	Datos de la Prueba			Resultado Esperado
1	Escenario-Condición	Caso de Prueba	Entidades/Estados	
2	Escenario-Condición	Caso de Prueba	Entidades/Estados	

Anexo 3: Formato para realizar el plan de pruebas utilizando la prueba de caja negra.

Código de la Prueba	Fecha	Nombre de la interfaz	Código de clases de equivalencia	Responsable de realizar la prueba

Donde:

- ✓ Código de la prueba: Este es el código que el responsable de la prueba debe asignarle.
- ✓ Fecha: Fecha en la cual se debe realizar la prueba.
- ✓ Método de prueba: El responsable tiene que escribir que método de prueba va realizar.
- ✓ Nombre de la Interfaz: Indica el nombre de la interfaz a probar.
- ✓ Responsable de realizar la prueba: El nombre del responsable que tiene que realizar la prueba.

- ✓ Número del grupo de clases de equivalencia: Se especifica el grupo de clase de equivalencia al que pertenece.

Anexo 4: Formato de para presentar los resultados de las pruebas.

Fecha (dd/mm/aa)	Código de la prueba	Detalles del caso de prueba	Resultados esperados	Resultados Obtenidos	Fecha Final y Aprobación

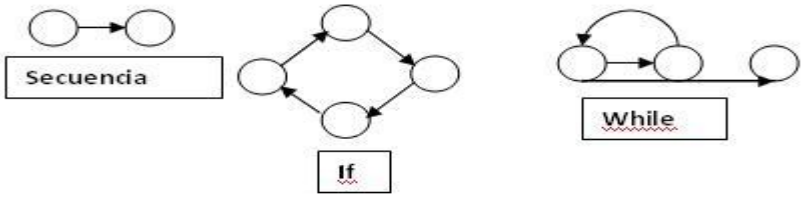
Donde:

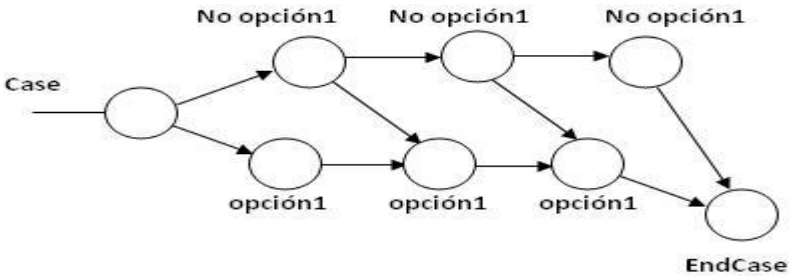
- ✓ Fecha indica la fecha en que se realizó la prueba
- ✓ Código de la prueba indica el código de las diferentes pruebas hechas a los métodos de la responsabilidad.
- ✓ Detalles del caso de prueba indica una breve descripción de la prueba a realizar.
- ✓ Resultados esperados indica los resultados que se esperan obtener antes de realizar la prueba.
- ✓ Resultados obtenidos son los resultados que se obtuvieron al realizar la prueba.
- ✓ Fecha Final y resultados indican la fecha en que se culminaron las pruebas y los resultados de esta, es decir, si el método está listo o queda pendiente.

Anexo 5: Notación de un grafo de flujo.

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo.

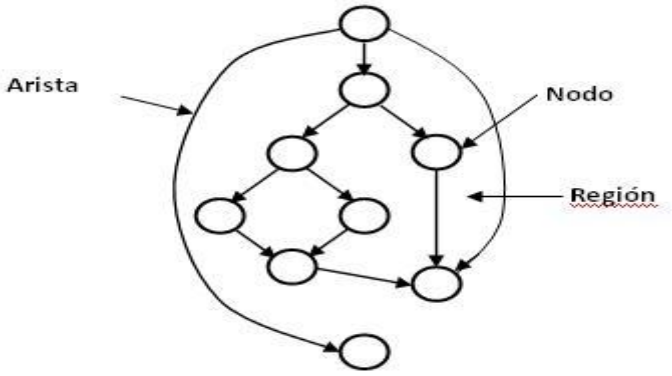
Cada nodo del grafo corresponde a una o más sentencias de código fuente. Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo.





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

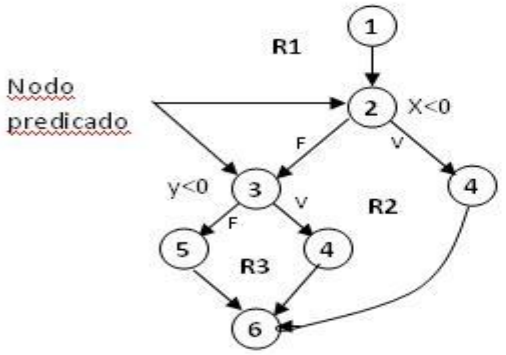
Anexo 6: Representación de Grafo de Flujo, en el cual aparecen sus componentes:



Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo. Cuando en un diseño se encuentran condiciones compuestas (uno o más operadores AND, NAND, NOR lógicos en una sentencia condicional), la generación del grafo de flujo se hace un poco más complicada. [Pressman, 2000]

Anexo 7: Elaboración de un Grafo de Flujo.

Si en un segmento de código se tiene una sentencia IF a OR b THEN entonces se crea un nodo aparte para cada una de las condiciones(a o b); cada nodo que contiene una condición se denomina nodo predicado y esta caracterizado porque dos o más aristas emergen de él.




```
PROCEDURE Imprime _media(VAR x, y : real;)
VAR resultado : real;
resultado:=0;
IF (x < 0 OR y < 0) THEN
        WRITELN("x e y deben ser positivos");
ELSE
```

Anexo 8: Cálculo de la complejidad ciclomática teniendo en cuenta la Figura 1.4:

1. El grafo de flujo tiene tres regiones.
2. V (G) = 8 aristas - 7 nodos + 2 = 3.
3. V (G) = 2 nodos predicados + 1 = 3.

Por tanto la complejidad ciclomática del grafo de flujo de la Figura 1.4 es 3.

Algunos consejos:

- ✓ Numerar los nodos del grafo secuencialmente.
- ✓ Describir el conjunto de caminos independientes (subrayar aristas que los hacen independientes de los anteriores).
- ✓ Algunos caminos no se pueden ejecutar solos y requieren la concatenación con otro.
- ✓ Algunos caminos pueden ser imposibles; seleccione otros.
- ✓ A partir de los caminos, analizar el código para ver qué datos de entrada son necesarios, y qué salida se espera.

Anexo 9: Encuestas aplicadas a proyectos de la facultad 5.

Proyectos	Auto Teórico-Práctico	Tiro	Medicina Quirúrgica	Laboratorios Virtuales	SCADA
Tipo de software que desarrolla	Simulador de auto para la obtención de la licencia de conducción	Simulador de Tiro	Simulador de Medicina	Software educativos	Software para la automatización
Metodologías base utilizadas	RUP	RUP	RUP	RUP	RUP
Productos realizados	✓ Simulador Práctico, Guatemala ✓ Multimedia para examen teórico de conducción,	Engine de Tiro	Demo de habilidades para cirujanos	Videos interactivos sobre enseñanza de la Programación	Solo prototipos y algunos entregables de todo el sistema

	República Dominicana				
Técnicas de pruebas utilizadas	Pruebas de unidad	No se utilizan	No se utilizan	No se utilizan	Pruebas de unidad basadas en el método de particiones equivalentes
Elaboración de planes de prueba	No se utilizan	No se utilizan	Se están elaborando	No se utilizan	No se utilizan
Planificación en paralelo con el proceso de desarrollo de software	No	No	No	No	No
Herramientas de pruebas usadas	No se utilizan	No se utilizan	No	No se utilizan	CxxTest, para la automatización de pruebas de unidad
Responsable por los procesos de pruebas	Sí	Sí	Sí	Sí	Sí
¿El probador reporta los defectos al equipo de desarrollo?					
Análisis de riesgos para la planificación de pruebas	No	No	No	No	No
Aplicación de soporte a usuarios insatisfechos	Sí	No	No	No	No
Utilización de pruebas de unidad	Sí	No se utilizan	No	No se utilizan	Sí
Utilización de pruebas de estrés	No	No se utilizan	No	No se utilizan	No
Utilización de pruebas de rendimiento	No	No se utilizan	No	No se utilizan	No
¿Un proceso de mejoramiento continuo para su proceso de pruebas?	No	No	No	No	No

Anexo 10: Resultados de la utilización de procesos pruebas en los proyectos productivos de la facultad 5.

	Se utilizan	No se utilizan	En proceso de elaboración	% utilización	% no utilización	% proceso de elaboración
Utilización de técnicas de pruebas	2	3	0	40%	60%	0%
Elaboración de planes de pruebas	0	4	1	0%	80%	20%
Realización de planes basados en riesgos	0	5	0	0%	100%	0%
Utilización de herramientas de pruebas	1	4	0	20%	80%	0%
Realización de técnicas de pruebas de unidad	2	3	0	40%	60%	0%
Utilización de estrategias de pruebas	0	5	0	0%	100%	0%

Anexo 11: Lista de chequeo para la elaboración del plan de pruebas.

Lista de chequeo para el cumplimiento del Plan de Pruebas. **Fecha:** **Responsable:**			
Actividades	Si	No	Información adicional
¿Refleja el plan de pruebas las siguientes características?			
Etapas y tipos de prueba que serán ejecutadas.			
Características o funciones que se probarán.			
Cualquier riesgo o contingencia que pueda afectar o afecta el esfuerzo de la prueba.			
¿Identifica claramente los artefactos (y sus versiones) usadas para generar el contenido del plan de pruebas?			

¿Cada requerimiento del proyecto tiene por lo menos un requisito asociado a la prueba o una declaración justificando el por qué no es un requisito para la prueba?			
¿Se identifican todos los requisitos para la prueba y se les da prioridad para cada uno de los tipos de pruebas que serán implementadas y ejecutadas?			
¿Para cada estrategia de prueba se identifica claramente la siguiente información?			
El nombre de la prueba y su objetivo.			
Una descripción de cómo será implementada y ejecutada la prueba.			
Una descripción de la métrica, los métodos, y los criterios que se utilizarán para evaluar la calidad y la terminación de la prueba			
¿Han sido identificados todos los recursos necesarios para implementar y ejecutar exitosamente la prueba, incluyendo hardware, software y personal?			
¿El plan de pruebas contiene una lista de iteraciones que identifican el proyecto y las tareas relacionadas a las pruebas (fechas de comienzo, de culminación y/o esfuerzo)?			
¿El plan de pruebas identifica los artefactos creados por las tareas de prueba, cuándo los artefactos se hacen disponibles, cómo serán distribuidos, su contenido, y cómo deben ser utilizados?			

Anexo 12: Lista de chequeo para comprobar la correcta aplicación del proceso de pruebas.

Lista de chequeo de aseguramiento de la calidad. **Fecha:** **Especialista:**			
Actividad	Si	No	Información adicional
¿Existe alguien en el proyecto responsable por los procesos de prueba?			
¿Se tiene y se usa un estándar para el plan de pruebas?			
¿Tiene y usa un estándar para el reporte de la ejecución de las pruebas?			
¿La planeación y ejecución de las pruebas se realiza en paralelo con el proceso de desarrollo de software?			

¿Se verifica que las especificaciones estén correctamente implementadas?			
¿Se verifica que las expectativas del cliente sean satisfechas?			
¿Se verifica la precisión y completitud de productos internos como el documento de requerimientos o los diseños?			
¿Los probadores reportan los errores al equipo de desarrollo para su corrección?			
¿Se identifican las prioridades de los riesgos del negocio para el desarrollo del plan de pruebas?			
¿Se han definido pronósticos de defectos basándose en datos y experiencias previas?			
¿Existe un proceso de mejoramiento continuo para su proceso de pruebas?			

GLOSARIO DE TÉRMINOS

A.

Administración del riesgo

Un acercamiento al análisis del problema identificando los riesgos, sus probabilidades de la ocurrencia, y su impacto. Da una comprensión más exacta de las pérdidas potenciales y cómo evitarlas. La administración de riesgos incluye la identificación, el análisis, la priorización, y el control del riesgo.

B.

Builds

Constituyen versiones o construcciones operacionales del sistema.

C.

CASE (*Computer Assisted Software Engineering, Ingeniería de Software Asistida por Ordenador)*

Las **Herramientas CASE** son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas proporcionan ayuda en todos los aspectos del ciclo de desarrollo del software, en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Ciclo de Vida del Software

El desarrollo de un buen sistema de software se realiza durante el *ciclo de vida* que es el período de tiempo que se extiende desde la concepción inicial del sistema hasta su eventual retiro de uso del mismo.

Cobertura

Es una medida del porcentaje de código que ha sido probado o “cubierto” con las pruebas. La cobertura requerida suele crecer con el grado de distribución del programa.

Clases de equivalencias

Son los conjuntos de estados (válidos o no) para las condiciones de entrada.

D.

Depuración

Es básicamente una técnica en la que una vez encontrados los errores del software se decide corregirlos.

Desarrollador

Un profesional técnico experto del software que está implicado en la definición, el diseño, la puesta en práctica, la evaluación, y el mantenimiento de un sistema de software.

E.

Entorno Virtual (EV)

Un EV puede definirse como una representación tridimensional por ordenador de un espacio en el que los usuarios pueden mover libremente su punto de vista en tiempo real.

Es un sistema interactivo que permite sintetizar un mundo tridimensional ficticio, creando una ilusión de realidad.

F.

Framework de prueba

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework de prueba puede incluir soporte de programas, bibliotecas entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de las pruebas.

H.

Herramienta de prueba

Sistemas de software y/o de hardware, u otros instrumentos, que se utilizan para medir y evaluar un artefacto del software.

I.

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.

Ingeniero de pruebas/Tester/ Probador

Un profesional técnico experto que está implicado en la prueba y la evaluación de un sistema de software, y además en la evaluación y mejora del proceso de prueba.

N.

Nivel de madurez

Una etapa evolutiva bien definida para un proceso apoyado por un sistema de metas y de prácticas que cuando son puestas en ejecución describen el estado del proceso.

O.

Orientado a Objetos (OO)

La arquitectura de software orientada a objetos se manifiesta en una serie de subsistemas organizados por capas, que encapsulan clases que colaboran entre sí. Cada uno de estos elementos del sistema realiza funciones que ayudan a alcanzar requerimientos del sistema. Es necesario probar un sistema OO, en una variedad de niveles diferentes, en un esfuerzo para descubrir errores, que deben ocurrir cuando las clases colaboran con otras entre sí, y los subsistemas se comunican por medio de las capas arquitectónicas.

P.

Particiones de equivalencia

Conjunto de datos de entrada que se espera que tengan un comportamiento equivalente.

Plan de contingencia

Es un plan para controlar las circunstancias del riesgo. Puede tratarse de incluir en la programación un número adicional de días para cumplir estas circunstancias, agregar personal y otros recursos, o cambiar la fecha de entrega.

R.

Recurso

Los medios físicos, humanos, y económicos necesitados apoyar un proceso, una política, un procedimiento, una meta, o un programa; por ejemplo, materiales, herramientas de hardware y de software, documentos de los estándares, miembros del personal, y fondos.

Refactorizar (refactoring)

Es la actividad de re-estructurar el software aplicando una serie de transformaciones sin cambiar su comportamiento externo. Refactorizar supone revisar de nuevo los códigos para procurar optimizar su funcionamiento.

Riesgo

El riesgo físico, humano, y económico. Posibilidad de sufrir pérdidas. Un riesgo es cualquier acontecimiento o condición probable que puedan tener un resultado potencialmente negativo en el proyecto en el futuro.

S.

SQA (Aseguramiento de la Calidad del Software)

Básicamente el equipo de SQA hace la planeación de las actividades de calidad en los proyectos, especificando las inspecciones, revisiones y pruebas requeridas durante el desarrollo.