

*Universidad de las Ciencias Informáticas*

*Facultad 6: Bioinformática*



*Título: “Propuesta de Pruebas para el Sistema de Manejo de  
Ensayos Clínicos”*

*Trabajo de Diploma para optar por el título de  
Ingeniero en Ciencias Informáticas.*

**Autor(es):** Lázara Y. Pérez Piñero.

Yurina López Borrero.

**Tutor(es):** Reynier García Vistorte.

**Co-tutor:** Yosdenis Urrutia Badillo.

Jorge Luis Vázquez González

**Julio de 2007**

*"El deber del hombre virtuoso no está sólo en el egoísmo de cultivar la virtud en sí, sino que falta a su deber el que descansa mientras la virtud no haya triunfado entre los hombres."*

*José Martí*

**DECLARACIÓN DE AUTORÍA**

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

Lazara Y. Pérez Piñero

Yurina López Borrero

Reynier García Vistorte

## AGRADECIMIENTOS

*Agradecemos a todos los profesores que a lo largo de estos 5 años han dado lo mejor para nuestra preparación.*

*Agradecemos a Yosdenis Urrutia Badillo y a Jorge Luis Vázquez González por su dedicación, su ayuda incondicional en los momentos en que más la necesitamos y por sus sugerencias siempre bien recibidas.*

*Agradecemos a todos nuestros compañeros del proyecto por habernos ayudado con todas nuestras dudas.*

*A nuestro tutor Reynier García Vistorte, que hizo su mayor esfuerzo por ayudarnos.*

*A nuestros padres, amigos y demás familiares que constituyeron el máximo apoyo para el desarrollo de nuestra labor como estudiantes y como parte del hermoso proyecto que es la UCI*

*Agradecemos infinitamente a nuestra Revolución, que nos haya permitido convertirnos en lo que hoy somos y que haya puesto en nuestras manos la honrosa tarea de llevar hacia delante este país, por eso y más este trabajo constituirá el primer paso de avance hacia ese camino.*

*A nuestro Comandante en jefe, Fidel Castro Ruz, por su incalculable capacidad de dirigir, su carácter intransigente, su inquebrantable confianza en la tropa del futuro, gracias.*

## **DEDICATORIA**

*Dedicamos este trabajo con mucho amor a:*

*Mi mamá y mi papá por su apoyo incalculable y se eterna Fé en mí.*

*A mis abuelas por el cariño y el afecto que me brindaron para darme fuerzas para terminar*

*A mis hermanas que me han agotado todos sus recursos por para que saliera adelante.*

*A mi novio Yuslier que me ha ayudado infinitamente, apoyándome en todas mis decisiones.*

*En fin a toda mi familia que de una forma u otra hicieron posible la realización de este trabajo.*

*Lázara*

*A mi familia, pero en especial:*

*A mi mamá Mabel porque es la principal responsable de que mi sueño se haya hecho realidad, porque es mi vida toda.*

*A mis abuelos Delvia, Pedro y Rosa que me inspiraron a seguir adelante.*

*A mi papá Carlos que me dio muchísimas fuerzas.*

*A Valencia por el apoyo y la paciencia de estos 5 años.*

*A mi novio Osmani por su confianza en mí y por permitirme entrar y permanecer en su vida.*

*A Albita, Osmani y Ariadna por brindarme su cariño y por todo lo que hicieron para que mi motivación se mantuviera siempre viva.*

*A todos ustedes muchas gracias de corazón porque sin ustedes no habría llegado hasta este momento.*

*Yurina*

# ÍNDICE

RESUMEN .....	IV
INTRODUCCIÓN .....	1
1.Fundamentación teórica .....	5
1.1 Calidad de software .....	5
1.2 Situación actual de la calidad de software .....	10
1.3 Factores que determinan la calidad de software .....	14
1.4 Pruebas de software y la calidad .....	17
1.5 Objetivos de las pruebas .....	20
1.6 Necesidad de realizar las pruebas.....	22
1.7 Tipos de pruebas .....	23
1.8 Conclusiones .....	35
2.Pruebas de software .....	36
2.1 SIMDEC (Sistema de Manejo de Datos de Ensayos Clínicos).....	37
2.2 La FDA y las pruebas del software .....	39
2.2.1 Objetivo fundamental de la FDA .....	39
2.2.2 Principios generales de la FDA.....	39
2.3 Propuestas de pruebas para posterior aplicación en el software .....	41
2.3.1 Pruebas de Unidad .....	41
2.3.2 Pruebas de Integración .....	58
2.2.3 Pruebas de sistema .....	58
2.4 Conclusiones .....	64
CONCLUSIONES .....	65
RECOMENDACIONES.....	66
REFERENCIAS BIBLIOGRÁFICAS .....	67
BIBLIOGRAFÍA .....	71
GLOSARIO DE TÉRMINOS .....	76

## **RESUMEN**

La evolución de la tecnología y la producción de software con diferentes propósitos en todo el mundo han generado diversas situaciones de competencia, donde las grandes empresas, debido a la necesidad de conservar una posición ventajosa han creado líneas de trabajo dedicadas a la calidad de productos en particular. Hace algunos años la calidad de software solo representaba una aspiración para los productores, sin embargo con el aumento de la competencia y las exigencias del mercado se ha hecho imprescindible para evitar grandes pérdidas de dinero y prestigio.

Cuba es uno de los países pioneros en este tema, en los momentos actuales se incursiona en las perspectivas económicas que brinda. Las empresas nacionales se adentran cada vez más en su desarrollo con el objetivo de incluirse en las grandes comunidades del software.

Las pruebas de software son esenciales para el resultado cualitativo del producto, tienen como objetivo principal detectar errores. Cuando se aplican las pruebas adecuadas, con personal calificado y previendo posibles ambientes de trabajo se detectan errores para corregir antes de la entrega al cliente, quien debe quedar satisfecho de acuerdo a sus expectativas.

# INTRODUCCIÓN

Durante el desarrollo de la informática a nivel mundial uno de los problemas en el escenario de las Tecnologías de la Información (TIC) es la calidad del software. Desde la década del 70, este tema ha sido motivo de preocupación para especialistas, ingenieros, investigadores y comercializadores de software, los cuales han realizado gran cantidad de investigaciones a partir del planteamiento de dos grandes interrogantes: ¿Cómo obtener un software con calidad? y ¿Cómo evaluar la calidad del software?, definitivamente estas interrogantes han hecho posible la creación de normas con tal de que organizaciones destinadas al desarrollo del software se apoyen en ellas y realicen un mejor trabajo. La calidad es subjetiva y cada una de las compañías productoras de software tiene diversas necesidades y criterios para valorar de diferentes formas los atributos de un producto o servicio. La realidad en nuestro país es que la mayor parte del software que se produce se realiza artesanalmente y no se aplican métodos industriales o procesos definidos de producción.

La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

La tendencia internacional actual está fundamentada en la aplicación y certificación sobre la base de las 9000 que suponen un lenguaje común, adoptado ya por un elevado número de países, pero además de las 9000, para ser efectivas en la calidad de determinados sectores es necesario compatibilizarlas con otras específicas adecuadas al tipo de actividad que desarrollan. Es por eso que en dependencia del software que sea, se hacen una serie de pruebas diferentes, con el objetivo de instaurar un software que reúna las características de funcionalidad necesaria y es la propuesta de este trabajo, una serie de pruebas factibles para el Sistema de Manejo de Datos de Ensayos Clínicos (SIMDEC).

Durante más de 100 años, los Institutos Nacionales de la Salud han descubierto nuevos tratamientos mediante ensayos clínicos. Gracias a este proceso, se descubren nuevas medicinas, se diseñan nuevos



tratamientos y se combaten enfermedades. Los ensayos clínicos han sido un medio importante para controlar enfermedades que un día fueron consideradas mortales.

Un ensayo clínico es un estudio que permite a los médicos determinar si un nuevo tratamiento, medicamento o dispositivo contribuirá a prevenir, detectar o tratar una enfermedad. Los ensayos clínicos también ayudan a descubrir si estos nuevos tratamientos son inocuos y si son mejores que los tratamientos actuales.<sup>1</sup>

En el ámbito internacional incluyendo a Cuba los Ensayos Clínicos son la base para el descubrimiento de valiosos medicamentos, debido a esto y a la gran responsabilidad que posee hacer un software encargado del manejo de datos de ensayos clínicos, la calidad comienza a jugar un papel muy importante. Muchas muestras de estos ensayos son las indicadas para comprobar la efectividad de una vacuna o medicamento de un tratamiento continuo, lo que hace inevitable que la calidad se convierta en uno de los paradigmas principales del desarrollo de este tipo de software. Para dar lugar a la afirmación anterior se visitaron una serie de sitios como la Revista Cubana de Medicina General Integral, además del sitio Web del Centro de Inmunología Molecular (CIM), se recopiló información acerca de los Ensayos Clínicos para así tener un mejor conocimiento de los tipos de pruebas que llevaran al software a la máxima eficiencia.

La Universidad de las Ciencias Informáticas (UCI) fue creada al calor de la batalla de ideas, y juega un rol importante en el desarrollo de la Industria Cubana del Software y en la materialización de los proyectos asociados al programa cubano de informatización. La Universidad está llamada a impulsar el desarrollo tecnológico de muchas organizaciones en el país a través de las TICs, para ello se realizan varios proyectos con el objetivo de mejorar las prestaciones y servicios de instituciones en las esferas de educación y salud fundamentalmente, de ahí la búsqueda de marcos de trabajo para aumentar la productividad, calidad, disminuir costos, mejorar los procesos de desarrollo y perfeccionar el diseño de pruebas.

Concebida la Universidad de las Ciencias Informáticas (UCI) como centro productor de software y tomando el perfil de Bioinformática como base de nuestras prioridades, a lo que se suma la novedad y

---

<sup>1</sup> CONDE, I. B. *Bioética en ensayos clínicos. Su aplicación actual*. Última actualización: 21 de abril de 1998. [Consultado el 17 de octubre de 2006]. Disponible en: [http://bvs.sld.cu/revistas/mgi/vol14\\_4\\_98/mgi07498.html](http://bvs.sld.cu/revistas/mgi/vol14_4_98/mgi07498.html)

urgencia de los proyectos actuales dentro de los cuales se incluye el SIMDEC y por la necesidad de poner este producto en marcha e incluirlo en el mercado para su uso extensivo en contraste con las perspectivas de salud, la **situación problemática** se enmarca en la propuestas de pruebas de calidad necesarias y no utilizadas en esta institución para validar este tipo de producto, además propiciar el acercamiento al tema de la calidad y el inicio de una serie de aportes que sirvan de fuente de consulta científica para garantizar la funcionalidad y aceptación en posibles mercados potenciales. El desconocimiento de pruebas para este producto permite no detectar los posibles errores del mismo, lo cual, imposibilita el buen manejo de los datos, por ejemplo el inserción de un dato fuera de rango puede traer como consecuencia un mal diagnóstico acerca de un paciente y puede afectar el estado de salud del mismo, por esto se hace necesario tener en cuenta las posibles pruebas de calidad aplicables a este software.

Luego de examinar cuidadosamente la necesidad de contar con producto de elevada calidad se identificó como **problema a resolver**: ¿Qué pruebas de calidad son las apropiadas para el SIMDEC?

Para dar solución al problema planteado se determino el siguiente **objetivo**: Proponer un conjunto de pruebas de calidad que permitan comprobar la funcionalidad de SIMDEC.

Partiendo del análisis del objetivo general se derivaron una serie de **objetivos específicos**:

- Identificar las pruebas adecuadas.
- Elaborar las propuestas de pruebas.

### **El objeto de estudio**

- Calidad de Software.

### **Campo de acción:**

- Pruebas de Calidad de Software para SIMDEC

Para cumplir los objetivos se realizarán varias **tareas** que harán posible dar solución a los problemas planteados así como la familiarización con los aspectos relacionados con el desarrollo del Sistema de Manejo de Datos de Ensayos Clínicos:

- Entrevistar a los desarrolladores del software, para obtener información acerca de como ha evolucionado el trabajo en la aplicación y cuales serán los requisitos que tendrá el software.
- Investigar sobre las pruebas de software más utilizadas mundialmente.
- Analizar las propuestas de pruebas para el caso específico del Sistema de manejo de datos de Ensayos Clínicos según la Agencia Regulatoria de Ensayos Clínicos.

Con este trabajo se pretende crear bases que apoyen la validación de productos de esta línea y similares del perfil de bioinformática, además de ser la primicia de una fuente de referencia técnica y conceptual para el desarrollo de la calidad de software de la rama de Ensayos Clínicos. Se hará posible la concepción de un producto a la altura de las exigencias actuales garantizando el prestigio de la Universidad de Ciencias Informáticas como centro productor de software. Por otra parte se espera que los involucrados en este proceso cuenten con una herramienta eficaz y de total confiabilidad para la obtención de resultados categóricos de las pruebas de Ensayos Clínicos.

.El presente trabajo consta de Introducción, 2 capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, y glosario de términos.

Capítulo 1: **Fundamentación teórica:** Ofrece una descripción de los principales aspectos relacionados con la calidad de software, así como de la actualidad nacional e internacional y su repercusión en el desarrollo económico de una empresa, institución, etcétera. Conocer los elementos necesarios para crear un sistema de manejo de datos de ensayos clínicos con la calidad requerida, haciendo énfasis en los posibles diseños de casos de prueba que se puedan aplicar a software de esta categoría.

Capítulo 2: **Pruebas de software:** En este capítulo se propondrán las pruebas para SIMDEC sobre la base de la agencia regulatoria de Ensayos Clínicos.

# Capitulo

## FUNDAMENTACIÓN TEÓRICA

### Introducción

En este capítulo se aborda de forma general los aspectos relacionados con la calidad de Software, situación actual, factores que determinan la calidad de un producto, vinculación de la pruebas a este tema como elemento crítico, y los diferentes tipos de pruebas que se pueden aplicar, haciendo énfasis en las aplicables al Sistema de manejo de Datos de Ensayos Clínicos.

### 1.1 Calidad de software

A lo largo de toda la historia, la búsqueda y el afán de perfección por parte del hombre ha sido constante, de tal forma, que el interés por el trabajo bien hecho y la necesidad de asumir responsabilidades sobre la labor efectuada poco a poco derivó en el concepto de calidad <sup>2</sup> que con el tiempo ha adquirido un carácter multidimensional. debido a que los diferentes autores, conocidos como los gurús del tema, lo han enfocado desde puntos de vistas diferentes: Edward Deming, lo enfoca como **“el grado predecible de uniformidad y conformidad a un bajo costo que se ajuste a las necesidades del mercado”**; . Philip

---

<sup>2</sup> PAZ, A. C. *El modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial* [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

B. Crosby como **“cumplir con los requisitos”**; y Joseph M. Juran como **“la idoneidad o aptitud para el uso”**.

Evidentemente todas esas definiciones son propias de autores que han investigado sobre el tema al igual que la planteada por Roger S. Pressman el cual expreso que la calidad de Software era **“la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de explícitamente documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente”**<sup>3</sup> ; esta definición es un poco mas abarcadora, porque la calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. Calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad, aunque la más utilizada es la brindada por la IEEE **“La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”**.<sup>4</sup>

EL control de calidad moderno o control de calidad estadístico comenzó en los años 30 del siglo XX con la aplicación industrial del cuadro de control ideado por el Dr. W. A. Shewhart, de Bell Laboratories, que fue el inventor de los conocidos gráficos de control.<sup>5</sup>

En este proceso cronológico se destaca el hecho de que años posteriores al desarrollo de la Segunda guerra mundial los japoneses comienzan a hacer verdadero énfasis en la calidad. En 1950 la Unión de Científicos e Ingenieros Japoneses realizó un cuyo conferencista el Dr. W.E., desarrolló los siguientes temas: Cómo mejorar la calidad mediante el ciclo de planear, hacer, verificar y actuar; La importancia de captar la dispersión en las estadísticas; Control de procesos mediante el empleo de cuadros de control y su aplicación.<sup>6</sup>

---

<sup>3</sup> PRESSMAN, R. S. Ingeniería del Software. Un enfoque practico. Quinta edición. ed. McGraw Hill, 2005. vol. I, 500 p

<sup>4</sup> SALANOVA, P. E. (2006). Modelos de Calidad Web. Clasificación de Métricas., UNIVERSIDAD NACIONAL DE EDUCACION A DISTANCIA: 308

<sup>5</sup> PAZ, A. C. *El* modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

<sup>6</sup> PAZ, A. C. *El modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial* [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

Cuatro años más tarde el Dr. J.M. Juran introdujo en Japón la idea de que la calidad de un producto o servicio residía en el grado de mentalización de todo el personal de la organización. La visita de Juran marcó una transición en las actividades de control de calidad y creó un ambiente en que se reconoció el Control de Calidad como un instrumento de gerencia abriéndose las puertas para el establecimiento del control total de calidad tal como se concibe hoy.<sup>7</sup>

La calidad es un tema de reciente desarrollo, ya no se puede hablar de hacer las cosas bien sino mantener un nivel de calidad adecuado durante la realización de un producto o servicio. Existen diferentes definiciones de calidad, el uso de cada una depende del área en que se está trabajando. Anteriormente se creía que la calidad era demasiado costosa y por eso influía en las ganancias producidas por la empresa. Ahora se sabe que el buscar la calidad resulta una baja en los costos de las empresas y una mayor ganancia.

El control de calidad de cualquier producto debe comprobar que la calidad del producto final se ajusta a una especificación dada. En el caso concreto del software, el control de calidad debe comprobar que el producto final funcione correctamente de acuerdo con sus especificaciones y en colaboración con otros sistemas software y bases de datos.<sup>8</sup>

Todos los autores coinciden en que el software posee determinados índices medibles que son las bases para la calidad, el control y el perfeccionamiento de la productividad.

A la hora de definir la calidad del software se debe diferenciar entre la calidad del producto software y la calidad del proceso de desarrollo de éste (calidad de diseño y fabricación). No obstante, las metas que se establezcan para la calidad del producto van a determinar los objetivos a establecer para la calidad del

---

<sup>7</sup> PAZ, A. C. El modelo de mcall como aplicación de la calidad a la revisión del software de gestión empresarial [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

<sup>8</sup> GUTIÉRREZ, J. J.; ESCALONA, M. J., et al. *Estudio comparativo de propuestas para la generación de casos de prueba a partir de requisitos funcionales*. [Consultado el: 25 de enero de 2007 de 2007]. Disponible en: <http://www.lsi.us.es/docs/informes/LSI-2005-01.pdf>.

proceso de desarrollo, ya que la calidad del primero va a depender, entre otros aspectos, de ésta. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto.<sup>9</sup>

Pero la calidad del producto software se diferencia de la calidad de otros productos de fabricación industrial, ya que el software tiene sus propias características específicas:

- El software es un producto mental, no restringido por las leyes de la Física o por los límites de los procesos de fabricación. Es algo abstracto, un intangible.
- Se desarrolla, no se fabrica. El coste está fundamentalmente en el proceso de diseño, no en la posterior producción en serie, y los errores se introducen también en el diseño, no en la producción.
- Los costes del desarrollo de software se concentran en las tareas de Ingeniería, mientras que en la fabricación clásica los costes se acentúan más en las tareas de producción.
- El software no se deteriora con el tiempo. No es susceptible de los efectos del entorno y su curva de fallos es muy diferente de la del hardware. Todos los problemas que surjan durante el mantenimiento estaban allí desde el principio y afectan a todas las copias del mismo; no se generan nuevos errores.
- Es artesanal en gran medida. El software, en su mayoría, se construye a medida, en vez de ser construido ensamblando componentes existentes y ya probados, lo que dificulta aún más el control de su calidad.
- El mantenimiento del software es mucho más complejo que el mantenimiento del hardware. Cuando un componente del hardware se deteriora se sustituye por una pieza de repuesto, pero cada fallo en el software implica un error en el diseño o en el proceso mediante el cual se tradujo el diseño en código máquina ejecutable.
- Es engañosamente fácil realizar cambios sobre un producto software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.

---

<sup>9</sup> PAZ, A. C. *El modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial* [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

- Como disciplina, el desarrollo de software es aún muy joven, por lo que las técnicas de las que dispone aún no están perfeccionadas.
- El software con errores no se rechaza. Se asume que es inevitable que el software presente algunos errores de poca importancia.<sup>10</sup>

La problemática general a la que se enfrenta el software es el:

- Aumento constante del tamaño y complejidad de los programas.
- Carácter dinámico e iterativo a lo largo de su ciclo de vida, es decir que los programas de software a lo largo de su vida cambian o evolucionan de una versión a otra para mejorar las prestaciones con respecto a las anteriores.
- Dificultad de conseguir productos totalmente depurados, ya que en ningún caso un programa será perfecto.
- Se dedican elevados recursos monetarios a su mantenimiento, debido a la dificultad que los proyectos de software entrañan y a la no normalización a la hora de realizar los proyectos.
- No suelen estar terminados en los plazos previstos, ni con los costes estipulados, ni cumpliendo los niveles deseables de los requisitos especificados por el usuario.
- Incrementos constantes de los costes de desarrollo debido entre otros, a los bajos niveles de productividad.
- Los clientes tienen una alta dependencia de sus proveedores por ser en muchos casos aplicaciones a "medida".
- Procesos artesanales de producción con escasez de herramientas.
- Insuficientes procedimientos normalizados para estipular y evaluar la calidad, costes y productividad.<sup>11</sup>

---

<sup>10</sup> PAZ, A. C. *El modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial* [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

<sup>11</sup> SCALONE, L. F. *ESTUDIO COMPARATIVO DE LOS MODELOS Y ESTANDARES DE CALIDAD DEL SOFTWARE*. Tutor: Martínez, D. R. G. Tesis de maestría, UNIVERSIDAD TECNOLÓGICA NACIONAL, FACULTAD REGIONAL BUENOS AIRES, 2006



También es importante destacar que la calidad de un producto software debe ser considerada en todos sus estados de evolución (especificaciones, diseño, códigos,...). No basta con verificar la calidad del producto una vez finalizado cuando los problemas de mala calidad ya no tienen solución o su reparación es muy costosa.<sup>12</sup>

### 1.2 Situación actual de la calidad de software

En un artículo publicado por Ana Ascasso, responsable de soluciones de calidad de COMPUWARE, empresa americana dedicada al estudio de la calidad, planteó que en el mundo, la calidad de software es una pionera en las empresas, por ejemplo en los países europeos, las empresas asumen ya la calidad de software pero no saben aplicar metodologías consistentes.

El 23,9% de las organizaciones admiten que sus equipos de calidad carecen de formación y de experiencia. En un 30,5% de los casos, los equipos sí tienen experiencia pero no están dedicados en la totalidad de su tiempo a asegurar la calidad.

Las empresas europeas están descuidando la calidad en el desarrollo de sus aplicaciones. Concretamente, el 78,3% de las empresas fallan a la hora de aplicar, de manera consistente, una metodología formal para asegurar la calidad (QA). De hecho, cerca de la mitad de las empresas (49%), no han aplicado un programa de calidad como CMM o Six Sigma para sus procedimientos de desarrollo, y con el objetivo de asegurar una calidad continua en sus aplicaciones.

El impacto negativo de la no implantación de un programa de calidad conlleva inconsistencia a la hora de desarrollar. Así, el 44,5% de las empresas declaran que la calidad la gestionan a nivel departamental y que implantan diferentes metodologías en la misma organización. Se estima que el 50% de los proyectos de tecnología informática no consiguen sus objetivos, y que en la mayoría de los casos ello se debe a una pobre metodología de calidad desde el principio.

---

<sup>12</sup> PAZ, A. C. *El modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial* [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en: <http://www.monografias.com/trabajos5/call/call.shtml#algu>.

Mejorar la calidad es vital para eliminar fallos y consecuentemente pérdidas, pero desafortunadamente en la actualidad todavía la calidad no forma parte ni siquiera de las agendas de las empresas.<sup>13</sup> En la siguiente grafica se pone de manifiesto el comportamiento de las estadísticas anteriormente expuestas.

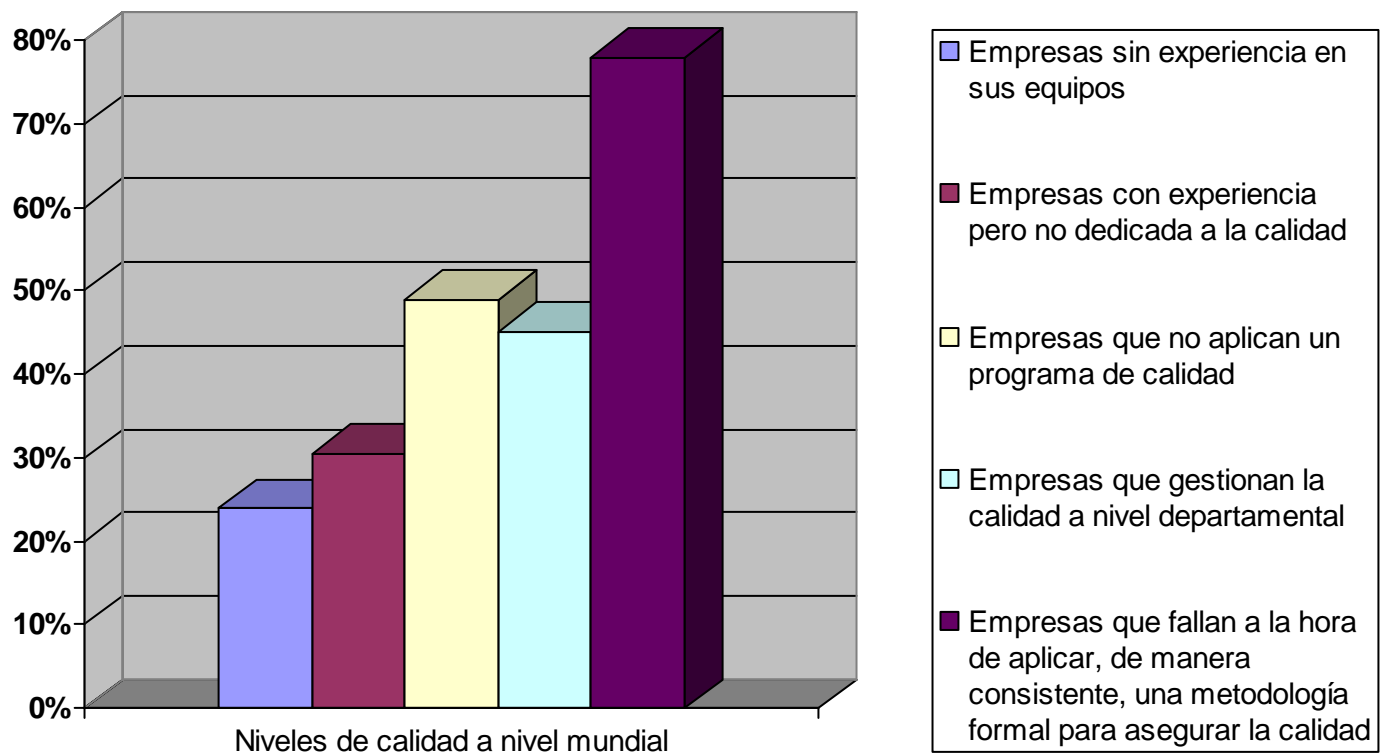


Fig. 1 Niveles de calidad a nivel mundial

El estudio de Compuware, dirigido por Ana Ascasso que se realizó en 24 países europeos y con un total de 184 entrevistas con responsables de los departamentos de Tecnología de la Información, sitúa a España, entre los cinco países europeos más adelantados en cuanto a asegurar la calidad del software.

<sup>13</sup> COMPUWARE. *LAS EMPRESAS EUROPEAS ASUMEN YA LA CALIDAD DEL SOFTWARE PERO NO SABEN APLICAR METODOLOGIAS CONSISTENTES* [Sitio Web]. [Consultado el: 15 de marzo de 2007 de 2007]. Disponible en: <http://www.noticias.com/notaprensa/28-04-2006/bdi-comunicacion/empresas-europeas-asumen-ya-calidad-software-pero-no-saben-aplicar-metodologias-consistentes-c29.html>.

Así, y entre los países europeos más avanzados en este aspecto según el estudio de Compuware, se destacan Austria, Reino Unido, Francia, Alemania y España.<sup>14</sup>

El desarrollo de una Industria Nacional de Software es una tarea de gran prioridad para Cuba debido a la alta perspectiva económica que posee, así como para el aseguramiento de un grupo importante de actividades del país. A pesar de ello, los resultados alcanzados no cubren las expectativas, ya que la productividad es baja, la cantidad real de recursos a consumir - en tiempo principalmente- es casi impredecible y el trabajo realizado casi nunca tiene la calidad y profesionalidad requerida. Los proyectos están excesivamente tarde y los beneficios que pudieran obtenerse al utilizar los mejores métodos e instrumentos en las distintas etapas no se detectan en este medio indisciplinado y caótico de desarrollo.<sup>15</sup>

Evidentemente nuestro país es una ínfima parte de lo que pueda ser el desarrollo de la calidad de software pero de lo que si se esta seguro es que no basta para comenzar a resolver los problemas existentes con respecto a la calidad, la aplicación de nuevas metodologías y tecnologías para desarrollar y mantener software, es necesario también aprender a administrar procesos de software para que se realicen con calidad. Todo esto requiere el establecimiento de sistemas de calidad y por tanto definir rigurosamente los procesos de software que se llevan a cabo en la empresa.

En estos momentos algunas empresas se están adentrando en este mundo, esta encuesta refleja los conocimientos de calidad que existen en estos momentos en nuestro país.

Resultados de la encuesta:

De un total de **10 personas** encuestadas:

1. **2** poseen un conocimiento regular sobre la calidad de software, y **8** admiten tener poco conocimiento del tema; lo que representa un 100% de personas con conocimiento entre medio y bajo.

---

<sup>14</sup> COMPUWARE. *LAS EMPRESAS EUROPEAS ASUMEN YA LA CALIDAD DEL SOFTWARE PERO NO SABEN APLICAR METODOLOGIAS CONSISTENTES* [Sitio Web]. [Consultado el: 15 de marzo de 2007 de 2007]. Disponible en: <http://www.noticias.com/notaprensa/28-04-2006/bdi-comunicacion/empresas-europeas-asumen-ya-calidad-software-pero-no-saben-aplicar-metodologias-consistentes-c29.html>.

<sup>15</sup> ESTRADA, A. F., S. A. Cárdenas, et al. (2006). "Calidad de Software y la empresa, enseñanza de un tema imprescindible para el ingeniero." Consultado: 1 de diciembre de 2006, 2006. Disponible en: <http://www.somece.org.mx/memorias/2000/docs/123.DOC>

2. **4** afirman conocer el tema pero nunca han aplicado pruebas, **4** admiten no conocer del tema de las pruebas y **2** afirman conocer las pruebas y haberlas aplicado. Esto representa un 20 % aproximadamente de personas que realmente conocen las pruebas de software.
3. **2** afirman haber realizado pruebas de software y conocer de lo que se esta haciendo en nuestro país con respecto a la calidad, y **6** plantean que no; lo que representa un 20% de personas que han aplicado pruebas y que conocen del tema de calidad.
4. **2** consideran como medio el nivel de aplicación de pruebas en las empresas cubanas de software, **5** insuficiente, y **3** no poseen criterio acerca del tema; lo que representa aproximadamente un 20% de personas que cree que existe algún nivel de avance en este sentido, 50% no lo creen así, y un 30% carece de opinión al respecto.
5. **7** le conceden mucha importancia a la aplicación de las pruebas en las empresas cubanas y solo **3**, regular; esto representa 70% y 30% respectivamente.
6. **6** personas dicen no tener conocimiento acerca de la calidad de software, y **4** personas dicen que saben algo de calidad. Esto representa un 60% de personas que no conocen nada del tema.

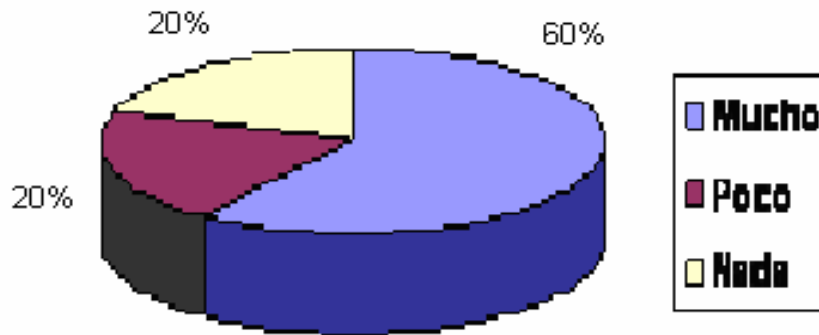


Fig. 2 Estado del conocimiento acerca del tema de Calidad de Software.

### 1.3 Factores que determinan la calidad de software

La calidad se puede determinar según factores importantes que nos dan en su medida hasta donde puede ser medida la calidad en un software determinado. Estos factores se clasifican en tres grupos:

- **Operaciones del producto** (características operativas):

– *Corrección* (¿Hace lo que se le pide?): El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente.

– *Fiabilidad* (¿Lo hace de forma fiable todo el tiempo?): El grado que se puede esperar de una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida.

– *Eficiencia* (¿Qué recursos hardware y software necesito?): La cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados.

– *Integridad* (¿Puedo controlar su uso?): El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado.

– *Facilidad de uso* (¿Es fácil y cómodo de manejar?): El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados.

- **Revisión del producto** (capacidad para soportar cambios)

- *Facilidad de mantenimiento* (¿Puedo localizar los fallos?): El esfuerzo requerido para localizar y reparar errores.
- *Flexibilidad* (¿Puedo añadir nuevas opciones?): El esfuerzo requerido para modificar una aplicación en funcionamiento.
- *Facilidad de prueba* (¿Puedo probar todas las opciones?): El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos.

- **Transición del producto** (adaptabilidad a nuevos entornos)

- *Portabilidad* (¿Podré usarlo en otra máquina?): El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
- *Reusabilidad* (¿Podré utilizar alguna parte del software en otra aplicación?): Grado en que partes de una aplicación pueden utilizarse en otras aplicaciones.
- *Interoperabilidad* (¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?): El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas.<sup>16</sup>

Un tema a destacar es que la calidad no es un problema de las personas de SQA (aseguramiento de calidad) solamente, sino un tema de todos los que participan en el proyecto. Todos, o casi todos, deben tener claro que los problemas de calidad de los componentes generados en las tareas iniciales del proyecto son los más costosos de corregir si no se detectan en tiempo.

En general, nunca se tiene tiempo para hacer las cosas bien de entrada. Pero sí para rehacer algo varias veces, siendo la sumatoria de los tiempos involucrados muy superiores al de hacerlo una vez bien. Todo esto sin contar los famosos costos de la “no calidad”, que en el caso de sistemas, en general, afectan los procesos críticos de la empresa y la percepción del servicio por parte de los clientes: insatisfacción,

---

<sup>16</sup> LOVELLE, J. M. C. (1999). “Calidad del software.” Consultado: 19 de abril de 2007, 2007, Disponible en: [http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad\\_software.PDF](http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF)

quejas, migración a otras empresas, pérdidas monetarias, deterioro de la imagen de la empresa, entre otros.<sup>17</sup>

La calidad del *software* puede definirse de muchas maneras. Una de las más limitadas, conocida como “calidad pequeña” define la calidad como la ausencia de defectos. Para evaluarla de esta forma se emplean procedimientos estadísticos a partir de las tendencias de aparición de fallas durante la prueba de *software*. Existen estándares industriales que marcan aceptabilidad cuando se estima el número de defectos residuales en 0.02 defectos por millar de líneas de código y aún menos.<sup>18</sup>

Obteniendo la calidad requerida en el *software*, se logra reducir su número de errores, o eliminarlos completamente, se alcanza una mayor fiabilidad para las funciones que debe realizar el mismo, mayor eficiencia e integridad de los datos así como mayor flexibilidad y reusabilidad.

El estudio de la calidad y fiabilidad tiene una importancia cada vez mayor en el mundo de la Ingeniería del Software. No sólo se trata de obtener sistemas desarrollados correctamente, de acuerdo a los requerimientos y a los estándares establecidos, sino que se pretende conseguir programas fáciles de mantener y, lo que es más importante, sistemas fiables en tareas críticas. A pesar de los avances en las técnicas de generación de código, no se pueden producir programas totalmente libres de errores. De esta forma, entre las distintas fases del ciclo de desarrollo se van filtrando una serie de errores que obligan a emplear mucho esfuerzo en su detección y corrección. Los errores pueden producirse en cualquier fase del ciclo de vida, pero sus efectos son mayores cuanto más temprana sea su aparición.

Podemos afirmar entonces que: **“La garantía de calidad del software es la guía de los preceptos de gestión y de las disciplinas de diseño de la garantía de calidad para el espacio tecnológico y la aplicación de la ingeniería del software”**.<sup>19</sup> La capacidad para garantizar la calidad es la medida de la madurez de la disciplina de ingeniería.

---

<sup>17</sup> BILELLO, M. A. *El valor de cada uno*. [Sitio Web]. Consultado: 11 de diciembre de 2006, 2006, Disponible en: [http://bloggers.com.ar/elsevier.com/system/noticia\\_detalle.php?id\\_prod=607](http://bloggers.com.ar/elsevier.com/system/noticia_detalle.php?id_prod=607).

<sup>18</sup> REYES., I. B. y TORRES., I. N. Las pruebas de software, su aplicación al Config. CASE. .Tutor. Estrada., M. A. F. TRABAJO DE DIPLOMA, INSTITUTO SUPERIOR POLITÉCNICO JOSÉ ANTONIO ECHEVARRÍA, 2003.

<sup>19</sup> PRESSMAN, R, S. Ingeniería del Software. Un enfoque practico. Quinta edición. ed. McGraw Hill, 2005. vol. I, 500 p

### 1.4 Pruebas de software y la calidad

Hoy en día, debido al aumento del tamaño y la complejidad del software, el proceso de prueba se ha convertido en una tarea vital en el de desarrollo de cualquier sistema informático.

La fase de pruebas es una de las más costosas del ciclo de vida software. En sentido estricto, deben realizarse pruebas de todos los artefactos generados durante la construcción de un producto, lo que incluye especificaciones de requisitos, casos de uso, diagramas de diversos tipos y, por supuesto, el código fuente y el resto de productos que forman parte de la aplicación (p.ej., la base de datos). Obviamente, se aplican diferentes técnicas de prueba a cada tipo de producto software.

Las pruebas del software son un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. La creciente percepción del software como un elemento del sistema y la importancia de los costes asociados a un fallo del propio sistema, están motivando la creación de pruebas minuciosas y bien planificadas.

De manera general, se puede decir que la prueba de *software* permite al desarrollador determinar si el producto generado satisface las especificaciones establecidas. Así mismo, una prueba de *software* permite detectar la presencia de errores que pudieran generar salidas o comportamientos inapropiados durante su ejecución.

De acuerdo a la IEEE <sup>20</sup> el concepto de prueba se define como:

***“Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente”***. Para Myers, probar (o la prueba) es el ***“proceso de ejecutar un programa con el fin de encontrar errores”*** <sup>21</sup>.

Otro concepto importante a tomar en consideración es el emitido por Pressman en su edición de 1998, que plantea que lo siguiente: ***“La prueba del software es un elemento crítico para la garantía de***

---

<sup>20</sup> IEEE, IEEE Std1995, “Metrics”, IEEE, 1991.

<sup>21</sup> REYES., I. B. y TORRES., I. N. Las pruebas de software, su aplicación al Config. CASE. .Tutor. Estrada., M. A. F. TRABAJO DE DIPLOMA, INSTITUTOSUPERIOR POLITÉCNICO JOSÉ ANTONIO ECHEVARRÍA, 2003



***calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación”.***<sup>22</sup>

La importancia de los costos asociados a los errores promueven la definición y aplicación de un proceso de pruebas minuciosas y bien planificadas. Las pruebas permiten validar y verificar el software, entendiendo como validación del software el proceso que determina si el software satisface los requisitos, y verificación como el proceso que determina si los productos de una fase satisfacen las condiciones de dicha fase.

El desarrollo de software tiene sólo algunas décadas; la industria del software está aún definiendo sus caminos, buscando la manera adecuada de desarrollarse. Considerar las propuestas basadas en la experiencia de los demás nos permitirá desarrollarnos con mayor velocidad; por ello hablaremos de las últimas tendencias de las pruebas de software.

Anteriormente, las pruebas de software se consideraban sólo una actividad que realizaba el programador para encontrar fallas 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 son un proceso cuyo propósito principal es evaluar la funcionalidad del software respecto de los requerimientos establecidos al inicio.

¿Cuál es la nueva tendencia en las pruebas? 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 de software; la propuesta es iniciar el análisis del testware junto con el análisis del software. Ello habla de sondeos preventivos que permitan ejecutar las pruebas tan pronto como el software esté listo y con ello no sólo descubrir errores, sino evitarlos.

---

<sup>22</sup> PRESSMAN, R, S. Ingeniería del Software. Un enfoque practico. Quinta edición. ed. McGraw Hill, 2005. vol. I, 500 p.

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 en conjunto, y en esta revisión deberá participar un especialista en pruebas.

Por otro lado, se debe reconocer que las pruebas son una especie de administrador de riesgos: al igual que en los problemas de combinatorias complejas, se puede definir cuál debe considerarse buen resultado, aunque no necesariamente sea el mejor; con ello se quiere decir que las pruebas sólo deben obtener un producto práctico con la calidad y funcionalidad requeridas.

Cuando aparecieron los primeros grandes sistemas informáticos se incluyó a nivel metódico e imprescindible un nuevo proceso en la confección de los mismos: el proceso de prueba.

Las pruebas de software presentan una interesante anomalía para el Ingeniero del Software. Durante las fases de definición y de desarrollo, el Ingeniero intenta construir el software partiendo de un concepto abstracto y llegando a una implementación tangible. Luego, llegan las pruebas. El Ingeniero crea una serie de casos de prueba que intentan “demoler” el software construido. De hecho, las pruebas son uno de los pasos de la Ingeniería del Software que se puede ver como destructivo en lugar de constructivo.<sup>23</sup>

---

<sup>23</sup> SCALONE, L. F. *ESTUDIO COMPARATIVO DE LOS MODELOS Y ESTANDARES DE CALIDAD DEL SOFTWARE*. Tutor: Martínez, D. R. G. Tesis de maestría, UNIVERSIDAD TECNOLÓGICA NACIONAL, FACULTAD REGIONAL BUENOS AIRES, 2006

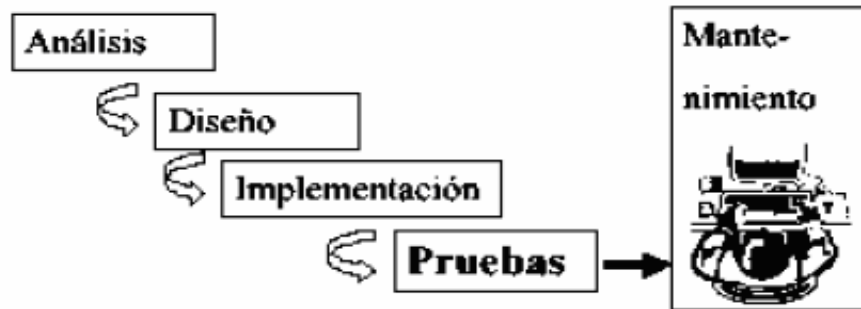


Fig. 3 Etapas de desarrollo de software

Los errores de codificación, con ser los más conocidos, no son los más importantes, a pesar de ello, son los más tratados pues es más simple automatizar su detección. Los errores en las pruebas son muy importantes pues dan por válido un código que no lo es, y permiten que se entregue un sistema defectuoso al cliente. Los errores de mantenimiento pueden deberse a la ignorancia de fallos antiguos o a la introducción de otros nuevos.

Los principios básicos de las pruebas de software son:

- A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.
- Las pruebas deberían planificarse mucho antes que empiecen.
- Las pruebas deberían empezar por “lo pequeño” y progresar hacia “lo grande”;
- Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.<sup>24</sup>

### 1.5 Objetivos de las pruebas

Dentro de los objetivos que se persiguen a la hora de aplicar las pruebas al software podemos mencionar:

- Brindar un correcto funcionamiento en los productos que se van generando
- Detectar fallas o errores.
- Aumentar la calidad del producto final.

<sup>24</sup> SCALONE, L. F. *ESTUDIO COMPARATIVO DE LOS MODELOS Y ESTANDARES DE CALIDAD DEL SOFTWARE*. Tutor: Martínez, D. R. G. Tesis de maestría, UNIVERSIDAD TECNOLOGICA NACIONAL, FACULTAD REGIONAL BUENOS AIRES, 2006

Los objetivos anteriores no descartan la idea de que una buena prueba es aquella en donde no se detectan errores, evidentemente nuestro objetivo con las pruebas es sacar a la luz diferentes clases de errores con la menos cantidad de tiempo y el menor esfuerzo posible.<sup>25</sup>

La prueba demuestra hasta que punto las funciones del software parecen funcionar de acuerdo con las especificaciones. Además los datos que se van recogiendo a medida que se lleva a cabo la prueba proporcionan una buena indicación de la fabricación del software y de laguna manera, indican la calidad del software como un todo.

Vale destacar la frase de Pressman que dice ***“La prueba no puede asegurar la ausencia de defectos; solo puede demostrar que existen defectos en el software”***.<sup>26</sup>

La afirmación anterior da la verdadera dimensión del proceso de prueba: sólo se puede demostrar que el software es erróneo, y no que sea completamente correcto. Cuando una prueba falla (esto es, cuando no encuentra errores) no se está en condiciones de afirmar que el programa probado está libre de defectos, sino que también puede ocurrir que la prueba sea incompleta o esté mal construida (en una palabra, que contenga errores). Esto implica que las medidas referidas al número de errores dependen enormemente del proceso de prueba que se haya seguido.

Las pruebas de software son una parte del proceso de aseguramiento de calidad; realizar pruebas a un sistema de información no significa necesariamente que el proceso de desarrollo esté asegurado y tampoco que de manera directa esté mejorando. Pero implementar un proceso de pruebas de software y más aún sostenerlo en el tiempo, es un buen inicio para más adelante aumentar el alcance y con base en las reflexiones al interior del equipo y de los hallazgos registrados en su producto, realizar un mejoramiento del proceso de desarrollo basado en los lineamientos del aseguramiento de calidad de software.

---

<sup>25</sup> PRESSMAN, R, S. Ingeniería del Software. Un enfoque practico. Quinta edición. ed. McGraw Hill, 2005. vol. I, 500 p

<sup>26</sup> PRESSMAN, R, S. Ingeniería del Software. Un enfoque practico. Quinta edición. ed. McGraw Hill, 2005. vol. I, 500 p

### 1.6 Necesidad de realizar las pruebas

Durante el desarrollo de software existe una alta posibilidad de que la fiabilidad humana aparezca de forma muy activa, lo que hace posible la existencia de fallos durante el ciclo de desarrollo del software. Hoy en día los responsables expertos de compañías de todo el mundo industrializado reconocen que la alta calidad del producto se traduce en ahorro de coste y en una mejora general. En realidad el hecho de que el software terminado posea la calidad requerida, depende mucho de la aplicación de las pruebas, ya que estas son un elemento crítico e imprescindible para lograr la calidad, de ahí la necesidad de probar diferentes software.

Algunas de las causas que sustentan las pruebas son:

**Propensión a equivocarse:** El ser humano es propenso a cometer equivocaciones; éstas se manifiestan en diversos problemas contenidos en los modelos (defectos o faltas) y pueden manifestarse como fallas en tiempo de ejecución.

**Fallas de hardware:** La infraestructura empleada para el desarrollo de *software* (*hardware*, sistemas operativos, compiladores, etc.) no está exenta de fallas, lo que introduce defectos adicionales o permite que subsistan inadvertidos los que introdujo el desarrollador.

**Creatividad del desarrollado:** El desarrollo de *software* es una labor creativa y por ello es común que el producto desarrollado no coincida con el modelo contenido en las especificaciones.<sup>27</sup>

Hoy en día se calcula que la fase o proceso de pruebas representa más de la mitad del coste de un programa, ya que requiere un tiempo similar al de la programación lo que obviamente acarrea un alto costo económico cuando estos no involucran vidas humanas, puesto que en este último caso el costo suele superar el 80% siendo esta etapa mas cara que el propio desarrollo y diseño de los distintos programas que conforman el sistema.

---

<sup>27</sup> REYES., I. B. y TORRES., I. N. Las pruebas de software, su aplicación al Config. CASE. .Tutor. Estrada., M. A. F. TRABAJO DE DIPLOMA, INSTITUTOSUPERIOR POLITÉCNICO JOSÉ ANTONIO ECHEVARRÍA, 2003

Un proceso de pruebas requiere mucho más que tiempo y dinero, necesita una verdadera metodología la cual exige herramientas y conocimientos destinados a dicha tarea, este texto trata de ser una pequeña guía para los programadores que aún no se han entrenado en este plano.

### 1.7 Tipos de pruebas

Durante el ciclo de vida de desarrollo de un software se pueden aplicar infinidad de pruebas, estas pruebas se deben aplicar de acuerdo al tipo de software que se esté construyendo.

Tom McCabe, planteó dos tipos fundamentales de pruebas sobre las que generalmente los programadores se basan, estas fueron publicadas en 1976 las cuales son:

- Prueba estructural o de caja blanca. Se centra en la estructura interna del programa (analiza los caminos de ejecución).
- Prueba funcional o de caja negra. Se centra en las funciones, entradas y salidas.<sup>28</sup>

Aunque la mayoría de las pruebas se centran en las anteriores no quiere decir que con solo aplicar caja blanca y caja negra, que precisamente son pruebas específicas y tiene un diseño propio, ya tenemos un software exento de errores. En este trabajo se mencionaran las pruebas que son aplicables a cualquier tipo de software y además se diseñaran las pruebas que necesita el software SIMDEC las cuales se basan en los principios que plantea la FDA (Food and Drug Administration) como agencia regulatoria de este tipo de software.

Existen muchos tipos de prueba que se pueden realizar y que son capaces de comprobar la funcionalidad del software, entre ellas están:

- **Pruebas de Integración:** Comprueban la correcta unión de los componentes del sistema entre sí a través de sus interfaces, y si cumplen con la funcionalidad establecida. Tienen por objetivo verificar el conjunto funcionamiento de dos o mas módulos, si bien se deben poner en práctica desde la creación de

---

<sup>28</sup> LETELIER, P. (2006). "Pruebas de software." Consultado: 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

dos módulos que interactúen entre si, en el supuesto caso que se necesiten mas de dos módulos para efectuar las pruebas, deberán generarse simples emuladores de módulos que entreguen datos esperados para la prueba individual de cada uno.<sup>29</sup>

Las pruebas 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 tengan unos pocos módulos, aunque no terminarán hasta disponer de la totalidad.<sup>30</sup>

- **Pruebas de Sistema:** Prueban a fondo el sistema, comprobando su funcionalidad e integridad globalmente, en un entorno lo más parecido posible al entorno final de producción. Uno de los objetivos de la fase de pruebas del sistema es verificar que el comportamiento externo del sistema software satisface los requisitos establecidos por los clientes y futuros usuarios del mismo. A medida que aumenta la complejidad de los sistemas software y aumenta la demanda de calidad, se hacen necesarios procesos y métodos que permitan obtener buenos conjuntos de pruebas del sistema. Este trabajo describe los modelos necesarios para generar de manera sistemática un conjunto de pruebas que permitan verificar la implementación de los requisitos funcionales de un sistema software.

- **Pruebas de requerimientos:** Los requerimientos de software deben tener una explicación clara, precisa y completa del problema que facilite el análisis de errores y la generación de casos de prueba. Un asunto de gran importancia es asegurar la corrección, coherencia y exactitud de los requisitos. Durante el proceso de elicitación de requerimientos, una persona, revisará el documento de especificación de requerimientos, con la lista de chequeo general del documento y la lista de chequeo de requerimientos.

---

<sup>29</sup> D'ONOFRIO, D. L. (2003). "Probando software y números de versión." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.masterdiseny.com/master-net/librecom/index.php3>

<sup>30</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

La corrección del contenido del documento será responsabilidad del analista y el usuario líder, quienes son los encargados de aprobar los requerimientos definidos en el documento.<sup>31</sup>

- **Pruebas de Implantación:** Las pruebas de implantación cubren un rango muy amplio, que va desde la comprobación de cualquier detalle de diseño interno hasta aspectos tales como las comunicaciones. Se debe comprobar que el sistema puede gestionar los volúmenes de información requeridos, se ajusta a los tiempos de respuesta deseados y que los procedimientos de respaldo, seguridad e interfaces con otros sistemas funcionan correctamente. Se debe verificar también el comportamiento del sistema bajo las condiciones más extremas.<sup>32</sup>

- **Pruebas de Aceptación:** Verifican que el sistema cumple con todos los requisitos indicados y permite que los usuarios del sistema den el visto bueno definitivo. Con este tipo de prueba se comprobará en la práctica que el sistema posee todas las funcionalidades exigidas en el cuestionario de especificaciones técnicas tanto en los que respecta a las características generales como en lo relativo a cada uno de los grupos de funciones y utilidades. Para ello se realizarán pruebas reales de funcionamiento sobre todas y cada una de las características funcionales exigidas<sup>33</sup>

- **Pruebas de Regresión:** El objetivo es comprobar que los cambios sobre un componente del sistema, no generan errores adicionales en otros componentes no modificados. Todos los sistemas sufren una evolución a lo largo de su vida activa. En cada nueva versión se supone que o bien se corrigen defectos, o se añaden nuevas funciones, o ambas cosas. En cualquier caso, una nueva versión exige una nueva pasada por las pruebas. Si éstas se han sistematizado en una fase anterior, ahora pueden volver a

---

<sup>31</sup> CORTÉS, O. H. G. (2004). “Aplicación práctica del diseño de pruebas de software a nivel de programación.” Consultado: 26 de enero de 2007, 2007, Disponible en: [http://www.willydev.net/descargas/oguzman-diseno\\_pruebas.pdf](http://www.willydev.net/descargas/oguzman-diseno_pruebas.pdf)

<sup>32</sup> PÚBLICAS, M. d. A. “Implantación y Aceptación del Sistema.” Consultado: 25 de enero de 2007, 2007, Disponible en: <http://www.csi.map.es/csi/metrica3/iasproc.pdf>

<sup>33</sup> GUTIÉRREZ, J. J., M. J. ESCALONA, et al. (2004). “Aplicando Técnicas de Testing en Sistemas para la Difusión Patrimonial.” Consultado: 26 de enero de 2007, 2007, Disponible en: [http://www.turismo.uma.es/turitec/turitec2004/docs/actas\\_turitec\\_pdf/15.pdf](http://www.turismo.uma.es/turitec/turitec2004/docs/actas_turitec_pdf/15.pdf)



pasarse automáticamente, simplemente para comprobar que las modificaciones no provocan errores donde antes no los había.<sup>34</sup>

Las pruebas de regresión son particularmente espectaculares cuando se trata de probar la interacción con un agente externo. Existen empresas que viven de comercializar productos que "graban" la ejecución de una prueba con operadores humanos para luego repetirla cuantas veces haga falta "reproduciendo la grabación". Y, obviamente, deben monitorizar la respuesta del sistema en ambos casos, compararla, y avisar de cualquier discrepancia significativa.<sup>35</sup>

- **Prueba de seguridad.** Se efectúa para garantizar que sólo los usuarios con la autoridad apropiada puedan utilizar las características aplicables del sistema. El ingeniero de sistemas establece diferentes configuraciones de seguridad para cada usuario en el entorno de pruebas.

El desarrollo seguro de aplicaciones es un campo muy reciente y que está aún bajo estudio; de hecho, las primeras referencias y los primeros libros que aparecen sobre el tema son del año 2001, lo que demuestra que el interés de todos los implicados en el campo del desarrollo sobre como realizar éste de forma segura es todavía muy nuevo. Esto explica fácilmente el hecho de que a pesar de la importancia del desarrollo seguro y de los grandes riesgos y amenazas que conlleva el no ajustarse a los principios del mismo a la hora de desarrollar las aplicaciones, las prácticas de desarrollo seguro no estén, ni mucho menos, ampliamente extendidas y, además, gran parte de los profesionales que trabajan en este campo no tienen aún clara su necesidad.<sup>36</sup>

La pregunta que surge es ¿cuánta seguridad es realmente necesaria? Para dar una adecuada respuesta a esto hay que tener en cuenta varias cosas:

---

<sup>34</sup> GUTIÉRREZ, J. J., M. J. ESCALONA, et al. (2004). "Aplicando Técnicas de Testing en Sistemas para la Difusión Patrimonial." Consultado: 26 de enero de 2007, 2007, Disponible en:

[http://www.turismo.uma.es/turitec/turitec2004/docs/actas\\_turitec\\_pdf/15.pdf](http://www.turismo.uma.es/turitec/turitec2004/docs/actas_turitec_pdf/15.pdf)

<sup>35</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en:

<http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>36</sup> Pérez, P. G. (2005). Principios básicos del desarrollo seguro. Consultado: 6 de junio de 2007. 2007, Disponible en:

[http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

- ¿Cuál es la prioridad principal para el desarrollo que se esté llevando a cabo? ¿La funcionalidad o la seguridad? Sin olvidar que, aunque se quiera tener el desarrollo lo antes posible, las consecuencias de implementar software sin ningún tipo de salvaguardas de seguridad pueden ser catastróficas.
- ¿En qué entorno se va a manejar el programa desarrollado? En un entorno hostil (Internet) o en un entorno controlado (corporación).
- ¿Cuál es el coste de introducir todas las medidas de seguridad? Es necesario valorar el coste que supone proteger completamente el desarrollo, frente a las ventajas que aporta hacerlo y los riesgos que se corren al no hacerlo. No tiene sentido invertir mucho en seguridad para algo que quizá no lo necesite realmente.
- ¿Qué métodos y principios del desarrollo seguro se ajustan más a la organización? Es necesario analizar y valorar las distintas maneras de realizar un desarrollo seguro y cómo pueden encajar dentro de la organización.<sup>37</sup>

En definitiva, en la práctica del desarrollo seguro es fundamental tener en cuenta la seguridad desde el principio del desarrollo, a la hora de encarar un proyecto.<sup>38</sup>

Una prueba de seguridad adecuada debería tener mecanismos para probar a las personas que realizan el desarrollo (en cuanto a su educación en desarrollo seguro y su preocupación por el mismo), los procesos que se siguen, la tecnología utilizada y por supuesto la implementación que se está llevando a cabo.<sup>39</sup>

- **Recorridos (walkthroughs).** Quizás es una técnica más aplicada en control de calidad que en pruebas. Consiste en sentar alrededor de una mesa a los desarrolladores y a una serie de críticos, bajo las órdenes de un moderador que impida un recalentamiento de los ánimos. El método consiste en que los revisores se leen el programa línea a línea y piden explicaciones de todo lo que no está meridianamente claro. Puede que simplemente falte un comentario explicativo, o que detecten un error auténtico o que

---

<sup>37</sup> Pérez, P. G. (2005). Principios básicos del desarrollo seguro. Consultado: 6 de junio de 2007. 2007, disponible en: [http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

<sup>38</sup> Pérez, P. G. (2005). Principios básicos del desarrollo seguro. Consultado: 6 de junio de 2007. 2007, disponible en: [http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

<sup>39</sup> Pérez, P. G. (2005). Principios básicos del desarrollo seguro. Consultado: 6 de junio de 2007. 2007, disponible en: [http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

simplemente el código sea tan complejo de entender/explicar que más vale que se rehaga de forma más simple. Para un sistema complejo pueden hacer falta muchas sesiones.

Esta técnica es muy eficaz localizando errores de naturaleza local; pero falla estrepitosamente cuando el error deriva de la interacción entre dos partes alejadas del programa. Nótese que no se está ejecutando el programa, sólo mirándolo con lupa, y de esta forma sólo se ve en cada instante un trocito del listado.<sup>40</sup>

- **Aleatorias (random testing).** Ciertos autores consideran injustificada una aproximación sistemática a las pruebas. Alegan que la probabilidad de descubrir un error es prácticamente la misma si se hacen una serie de pruebas aleatoriamente elegidas, que si se hacen siguiendo las instrucciones dictadas por criterios de cobertura (caja negra o blanca).

Como esto es muy cierto, probablemente sea muy razonable comenzar la fase de pruebas con una serie de casos elegidos al azar. Esto pondrá de manifiesto los errores más patentes. No obstante, pueden permanecer ocultos errores más sibilinos que sólo se muestran ante entradas muy precisas.

Si el programa es poco crítico (una aplicación personal, un juego...) puede que esto sea suficiente. Pero si se trata de una aplicación militar o con riesgo para vidas humanas, es de todo punto insuficiente.<sup>41</sup>

- **Solidez (robustness testing).** Son las encargadas de verificar la capacidad del programa para soportar entradas incorrectas, por ejemplo en un sistema de facturación donde el usuario debe ingresar códigos de productos y luego cantidades es mas que factible que en algún momento ingrese un código en el campo de cantidad, si el programa fue sometido a pruebas de robustez este valor sería rechazado o grabado como una cantidad inmensa pero que no daría error por desbordamiento de datos. Se prueba la capacidad del sistema para salir de situaciones embarazosas provocadas por errores en el suministro de datos. Estas pruebas son importantes en sistemas con una interfaz al exterior, en particular cuando la interfaz es humana.

---

<sup>40</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>41</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

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

- órdenes correctas, todas y cada una
- órdenes con defectos de sintaxis, tanto pequeñas desviaciones como errores de bulto
- órdenes correctas, pero en orden incorrecto, o fuera de lugar
- la orden nula (línea vacía, una o más)
- órdenes correctas, pero con datos de más
- provocar una interrupción (BREAK, ^C, o lo que corresponda al sistema soporte) justo después de introducir una orden.
- órdenes con delimitadores inapropiados (comas, puntos, ...)
- órdenes con delimitadores incongruentes consigo mismos (por ejemplo, esto] <sup>42</sup>

• **Aguante (stress testing).** Se utilizan para saber hasta donde puede soportar el programa condiciones extremas, por ejemplo los tiempos de respuesta con el procesador a un 95% de su utilidad o con muy poco espacio en disco.

En ciertos sistemas es conveniente saber hasta dónde aguantan, bien por razones internas (¿hasta cuantos datos podrá procesar?), o bien por razones externas (¿es capaz de trabajar con un disco al 90%?, ¿aguenta una carga de la CPU del 90?, etc.).<sup>43</sup>

• **Prestaciones (performance testing).** A veces es importante el tiempo de respuesta, u otros parámetros de gasto. Típicamente nos puede preocupar cuánto tiempo le lleva al sistema procesar tantos datos, o cuánta memoria consume, o cuánto espacio en disco utiliza, o cuántos datos transfiere por un canal de comunicaciones. Para todos estos parámetros suele ser importante conocer cómo evolucionan al variar la dimensión del problema (por ejemplo, al duplicarse el volumen de datos de entrada).<sup>44</sup>

---

<sup>42</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>43</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>44</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

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

Las pruebas, de caja negra, que se le pasan a un producto para detectar discrepancias respecto a una norma de las descritas en el párrafo anterior se denominan de conformidad u homologación. Suelen realizarse en un centro especialmente acreditado al efecto y, si se pasan satisfactoriamente, el producto recibe un sello oficial que dice: "homologado".<sup>45</sup>

- **Interoperabilidad (interoperability testing).** En el mismo escenario del punto anterior, programas de comunicaciones que deben permitir que dos ordenadores se entiendan, aparte de las pruebas de conformidad se suelen correr una serie de pruebas, también de caja negra, que involucran 2 o más productos, y buscan problemas de comunicación entre ellos.<sup>46</sup>

- **Mutación (mutation testing).** Es una técnica curiosa consistente en alterar ligeramente el sistema bajo pruebas (introduciendo errores) para averiguar si nuestra batería de pruebas es capaz de detectarlo. Si no, más vale introducir nuevas pruebas. Todo esto es muy laborioso y francamente artesano.<sup>47</sup>

- **Pruebas Unitarias:** Prueban el diseño y el comportamiento de cada uno de los componentes del sistema una vez construidos, es decir se prueba cada uno de los módulos. Para Pruebas Unitarias existen fundamentalmente dos enfoques:

- Pruebas de Caja Blanca (o Pruebas Estructurales)
- Pruebas de Caja Negra (o Pruebas Funcionales)<sup>48</sup>

---

<sup>45</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>46</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>47</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

<sup>48</sup> Letelier, P. (2006). "Pruebas de software." Consultado: 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

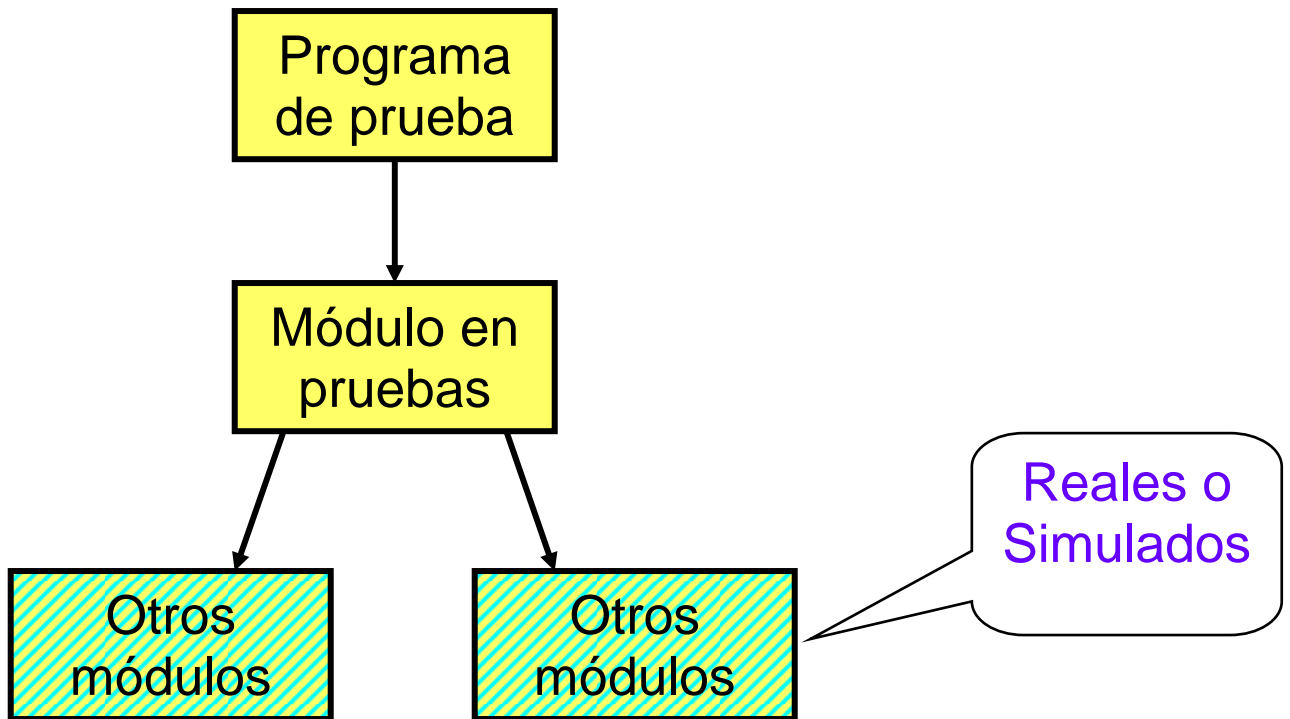


Fig.4 Esquema de representación de Prueba unitaria<sup>49</sup>

- **Prueba de caja blanca.** Durante la aplicación de este tipo de prueba se trabaja con el código del proyecto, este es separado por segmentos y se le asigna un número a cada instrucción con el objetivo de crear un grafo con estos números. Cada sentencia tiene una representación, con esta prueba se realiza la revisión del código detectando errores lógicos dentro de este. Una vez hecho esto se calcula la complejidad ciclomatica del grafo formado con el objetivo de definir la cantidad de casos de pruebas que se van a hacer.

Con este método se determina cuáles son los casos de prueba a partir del código fuente del software y se utilizan las especificaciones para determinar el resultado esperado del caso. 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. La prueba de caja blanca del software se basa en el

<sup>49</sup> Collado, M. (2003). "Pruebas de software. Técnicas de prueba del software. Estrategia de prueba del software." Consultado: 17 de febrero de 2007, 2007. Disponible en: <http://lml.ls.fi.upm.es/ftp/ed2/0203/Apuntes/pruebas.ppt>

minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y ciclos. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.<sup>50</sup>

• **Prueba de Caja negra.** Se centran en las funciones, entradas y salidas. Es impracticable probar el software para todas las posibilidades. De nuevo hay que tener criterios para elegir buenos casos de prueba. Suelen ser llamadas también :

- pruebas de caja opaca
- pruebas funcionales
- pruebas de entrada/salida
- pruebas inducidas por los datos

Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro.

Las pruebas de caja negra están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario (en sentido general: teclado, pantalla, ficheros, canales de comunicaciones, etc.) Este comentario no basta para que sean útiles en cualquier módulo del sistema. Estas pruebas se apoyan en la especificación de requisitos del módulo.

El problema con las pruebas de caja negra no suele estar en el número de funciones proporcionadas por el módulo (que siempre es un número muy limitado en diseños razonables); sino en los datos que se le pasan a estas funciones. El conjunto de datos posibles suele ser muy amplio (por ejemplo, un entero).

A la vista de los requisitos de un módulo, se sigue una técnica algebraica conocida como "clases de equivalencia". Esta técnica trata cada parámetro como un modelo algebraico donde unos datos son

---

<sup>50</sup> JOSÉ, F. M. and G. J. PABLO. (1999). "Sistemas de Programas." Consultado: 12 de abril de 2007, 2007, Disponible en: <http://www ldc.usb.ve/~teruel/ci3711/test1/index.html#3>.

equivalentes a otros. Si logramos partir un rango excesivamente amplio de posibles valores reales a un conjunto reducido de clases de equivalencia, entonces es suficiente probar un caso de cada clase, pues los demás datos de la misma clase son equivalentes.

El problema está en identificar clases de equivalencia, tarea para la que no existe una regla de aplicación universal.<sup>51</sup>

Independientemente de todas las pruebas anteriormente presentadas, y obviando los innumerables tipos de pruebas que se pueden aplicar, podemos mencionar que en nuestra universidad solo se aplican pruebas funcionales, es decir pruebas de caja negra, en estos momentos se está incursionando en el tema de caja blanca y otros tipos de pruebas, pero por el momento solo las funcionales están presentes.

Lo que caracteriza un escrito formal de caso de prueba es que hay una entrada conocida y una salida esperada, los cuales son formulados antes de que se ejecute la prueba. La entrada conocida debe probar una precondición y la salida esperada debe probar una poscondición.

Bajo circunstancias especiales, podría haber la necesidad de ejecutar la prueba, producir resultados, y luego un equipo de expertos evaluaría si los resultados se pueden considerar como "Correctos". Esto sucede a menudo en la determinación del número del rendimiento de productos nuevos. La primera prueba se toma como línea base para los subsecuentes ciclos de pruebas/lanzamiento del producto.

Los casos de prueba escritos, incluyen una descripción de la funcionalidad que se probará, la cuál es tomada ya sea de los requisitos o de los casos de uso, y la preparación requerida para asegurarse de que la prueba pueda ser dirigida.

El objetivo de las pruebas de un software es predecir si dicho software cumplirá los objetivos de diseño relativos a una alta disponibilidad, seguridad, escalabilidad y facilidad de administración en condiciones de producción.

---

<sup>51</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>



El diseño de casos de prueba está totalmente mediatizado por la imposibilidad de probar exhaustivamente el software. Pensemos en un programa muy sencillo que sólo suma dos números enteros de dos cifras (del 0 al 99). Si quisiéramos probar, simplemente, todos los valores válidos que se pueden sumar, deberíamos probar 10.000 combinaciones distintas de cien posibles números del primer sumando y cien del segundo. Pensemos que aún quedarían por probar todas las posibilidades de error al introducir los datos (por ejemplo, teclear una letra en vez de un número).

La conclusión es que, si para un programa tan elemental debemos realizar tantas pruebas, pensemos en lo que supondría un programa medio.

En consecuencia, las técnicas de diseño de casos de prueba tienen como objetivo conseguir una confianza aceptable en que se detectarán los defectos existentes (ya que la seguridad total sólo puede obtenerse de la prueba exhaustiva, que no es practicable) 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.

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, podemos tener un cierto nivel de confianza (que depende de la elección de los casos) en que el programa no tiene defectos.

Los casos de prueba deben estar "vivos", deben modificarse si se modifica el comportamiento del sistema, crecer y sobre todo mejorarse. Al principio no suele salir lo que se espera y surgen muchas dudas, pero en este caso, el tiempo y el uso de los casos de pruebas despejan las dudas, la mente, y realmente brindan la seguridad de que todo funciona como debe funcionar.

Evidentemente las pruebas son la herramienta necesaria para llevar a cabo un software con calidad, ya que nos apoyamos primero en los juegos de datos que el propio ingeniero de prueba puede hacer, para

luego pasar a las pruebas de aceptación, que serian con el cliente como tal. Con base en lo antes explicado podemos afirmar que la confección del software de SIMDEC, con la calidad requerida, posibilita los buenos resultados de un diagnostico que luego será aplicado a pacientes, por lo cual recae sobre el software una gran responsabilidad, es por eso que las pruebas de calidad efectuadas al software deben ser las adecuadas para que este quede con la calidad requerida.

### **1.8 Conclusiones**

Después de hacer un análisis de la información recopilada sobre la calidad de software y las pruebas de calidad que se realizan se puede concluir que dadas las especificidades de los proyectos de Bioinformática y según la agencia regulatoria, para el software SIMDEC, es necesario el diseño de las pruebas de unidad, de sistema y de integración. Es muy importante realizar estas pruebas teniendo en cuenta todos los aspectos fundamentales que se requieren para este tipo de producto debido a su importancia para la salud y exclusivamente para combatir el cáncer.

Las pruebas a diseñar propician beneficios ya que aumentan el grado de consistencia del software que se desea implementar, con su aplicación se garantiza la funcionalidad de la aplicación y evitan resultados erróneos de los ensayos realizados. Además debido a la situación económica actual y el costo de estas pruebas se hace necesario llevar a cabo todo el procedimiento para fortalecer el desarrollo de la bioinformática en nuestro país que ya muestra resultados meritorios.

# Capítulo **2**

## PRUEBAS DE SOFTWARE

### Introducción

En la actualidad existen diversos problemas para obtener la calidad del software. La calidad se puede expresar como eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.

La calidad del software se puede medir y varía de un programa a otro según las funciones para las que fue elaborado, por ejemplo el software que se desarrolla para el control de aparatos médicos debe de ser confiable "cero fallas" un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad; mientras que un producto de software que es utilizado durante un periodo de 5 años necesita ser confiable, mantenible y flexible para de esta manera poder disminuir los costos de mantenimiento que pueda haber durante el tiempo de su explotación.

En Cuba, la industria del software se esta convirtiendo en fuerza impulsora de la economía, la realización de software con calidad es un elemento crítico e imprescindible para lograr insertar nuestros productos en el mercado mundial. El diseño de pruebas esta muy ligado a la calidad lo que implica que si se quiere lograr un software con la calidad requerida es necesario aplicar una serie de pruebas que apoyen los resultados finales de los productos. En este capítulo se diseñarán las pruebas necesarias para que el SIMDEC muestre la calidad requerida.

### 2.1 SIMDEC (Sistema de Manejo de Datos de Ensayos Clínicos).

El sistema de manejo de datos de ensayos clínicos es un software que requiere de alta confiabilidad, se manejarán los datos de aquellas personas que padecen de cierta y determinada enfermedad, quiere decir que debe ser extremadamente eficaz y confiable para los usuarios. Para este proceso del manejo de datos se usarán cuadernos de recogida de datos (CRD), los cuales serán una especie de documento automatizado que se usarán para recoger y transmitir toda la información requerida en el protocolo de cada sujeto del ensayo clínico. Se manejarán los datos de pacientes a los cuales se les realizarán ensayos para algún tipo de tratamiento.

El sistema debe cumplir con ciertos requisitos:

- **Navegación amigable, adecuada y cronológica.** El sistema debe contemplar un modo de navegación similar al utilizado en el CRD en papel que sigue el esquema (Paciente->Visita->Modelo->Página) pudiendo mostrarse el árbol de navegación en la parte izquierda de la pantalla. No deben ser visible a un usuario paginas que no le son accesibles, ya sea por sus permisos o porque los datos en la misma corresponden a un estadio de tiempo posterior al actual en el curso del ensayo.
- **El SC (sistema de computo) debe usar/tener diferentes fuentes, formatos y colores en la tipografía de una misma pantalla según el mensaje que se escriba.**

Ejemplos:

- La descripción de un campo puede tener un color, el valor que registra otro y un texto que funciona como recordatorio o aclaración otro.
- Las páginas de entrada de datos, Monitoreo, Auditoria y demás deben indicar con íconos, tipos de letra o colores diferentes el estado de cada dato (ej: si tiene notas, si esta pendiente de monitoreo o revisión de los datos, si la revisión es completa o parcial, etc.)
- **Control de Acceso.** Cada usuario del sistema solo puede acceder a la parte del sistema que le corresponde y durante el tiempo que se defina. Por ejemplo, el cierre de la base de datos implica que se deja de tener acceso a modificaciones el sistema.

- **Autenticación y firma electrónica.** Cada usuario deberá tener un login, password y firma electrónica que garanticen la atributabilidad de los datos entrados o modificados en el estudio. Para ello el SC debe permitir:
  - ✓ Especificar, por el administrador, el intervalo de tiempo en el cual se obligará a los diferentes usuarios a cambiar su password. Utilizar refrescador de pantalla y login automático de un usuario después de un tiempo de inactividad en el sistema. Impidiendo así que un usuario trabaje en la sesión de otro.
  - ✓ En cada sesión de trabajo debe estar visible en pantalla la identificación del usuario que trabaja en esa sesión y si es posible sus derechos según su Rol.
  - ✓ Las diferentes opciones, paginas y herramientas del SC deben ser personalizadas según el Rol del usuario activo en el sistema
- **Instrucciones de validación de los datos recogidos.** El sistema debe permitir diseñar instrucciones de validación propias para cada variable, las cuales pueden depender del ensayo y del campo en cuestión. Las instrucciones están relacionadas con:
  - ✓ Establecimiento de rangos para las variables (limites inferiores y/superiores, permitiendo que en cualquier caso alguno de estos limites pueda quedar sin especificar; ejemplo Edad mayor de 18 años).
  - ✓ Comprobación de respuestas entre variables relacionadas (la respuesta de una variable depende del valor de otra; ejemplos: si el paciente no viene a la visita, entonces no puede tener datos registrados de esa visita, el campo fecha de inclusión debe ser mayor o igual al de la fecha del consentimiento informado).

Como resultado de esta validación el software debe permitir generar diferentes mensajes y acciones como las siguientes:

- ✓ Mensajes dirigidos al personal que está manipulando el dato, ya sea entrando o modificando su valor. Estos pueden ser de tipo "Error" (no se permite asentar el dato, hasta una consulta y

aprobación por autoridad predefinida en el sistema), de tipo “Advertencia” (puede ser un error pero se permite continuar y registrar el valor), de tipo “obligatoriedad” (no se permite continuar hasta que se completa el dato o este se corrige).

- ✓ Mensajes dirigidos a una o varias personas que no son las que están manipulando el dato (monitor, promotor o investigador principal del sitio).

- **Evitar campos abiertos.** El SC debe evitar el uso de campos abiertos, pues estos introducen ambigüedad en los datos recogidos y dificultan los análisis estadísticos. Para esto el mismo debe permitir al gestor central de datos introducir información para definir plecas con los posibles valores a tomar por un campo de datos dado (aunque incluyendo siempre otros para introducir texto libre).

## 2.2 La FDA y las pruebas del software

**La FDA o Food and Drug Administration** (Administración de Drogas y Alimentos, por sus siglas en inglés) es un organismo normativo científico responsable de la seguridad de los alimentos, cosméticos, medicamentos, productos biológicos, dispositivos médicos y productos radiológicos tanto producidos en el país como importados.<sup>52</sup>

### 2.2.1 Objetivo fundamental de la FDA

El supremo mandato de la FDA es regular la multitud de productos medicinales de una manera tal que asegure la seguridad de los consumidores norteamericanos y la efectividad de las drogas comercializadas.

### 2.2.2 Principios generales de la FDA

La FDA publicó un documento llamado “Principios generales para la validación de software”, en el cual se brinda la información necesaria correspondiente a las pruebas para este tipo de software, es decir para

---

<sup>52</sup> UNIDOS, A. P. L. A. Y. M. L. E. *Centro para la Seguridad Alimentaria y la Nutrición Aplicada*. Última actualización: 24 de enero de 2005. [Consultado el 25 de marzo de 2007]. Disponible en: <http://www.cfsan.fda.gov/~mow/scfsan4.html>.

software médicos. A continuación están algunos de los principios por los que se debe regir el personal calificado para efectuar las pruebas.

- Un buen caso de la prueba tiene una alta probabilidad de exponer un error.
- Una prueba acertada es una que encuentra un error.
- Hay independencia de la codificación.
- Tanto la aplicación usuario y la programación del software deben ser bien revisados.
- Examinar solamente un caso de prueba generalmente no es suficiente.
- La documentación de prueba debe permitir su reutilización y una confirmación independiente del estado del resultado de la prueba durante la revisión subsecuente.

Para la realización de este trabajo se tuvo en cuenta los requerimientos del software y el documento planteado por la FDA: General Principles of Software Validation; Final Guidance for Industry and FDA Staff<sup>53</sup> (Principios Generales de Aprobación del Software; Última Guía para la Industria y Personal de la FDA) acerca de los principios generales para la validación de software, planteados en la sección 5.2.5. Testing by the Software Developer (Probando por el Diseñador del Software) es por eso que se hace una selección de las diferentes pruebas que se pueden aplicar de acuerdo a lo anterior, dichas pruebas son:

- Pruebas de unidad
  - Prueba de caja blanca (“White box”)
  - Prueba de caja negra (“Black box”)
- Pruebas de Integración
- Pruebas de sistema
  - Prueba de usabilidad
  - Prueba de Rendimiento
  - Pruebas de Seguridad

---

<sup>53</sup> U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES. U.S. Food and Drug Administration [Web Site]. Rockville: [Consultado el: 14 febrero de 2007]. Disponible en: <http://www.fda.gov/oc/spanish/>

## **2.3 Propuestas de pruebas para posterior aplicación en el software**

### **2.3.1 Pruebas de Unidad**

#### **2.3.1.1 Prueba de Caja blanca**

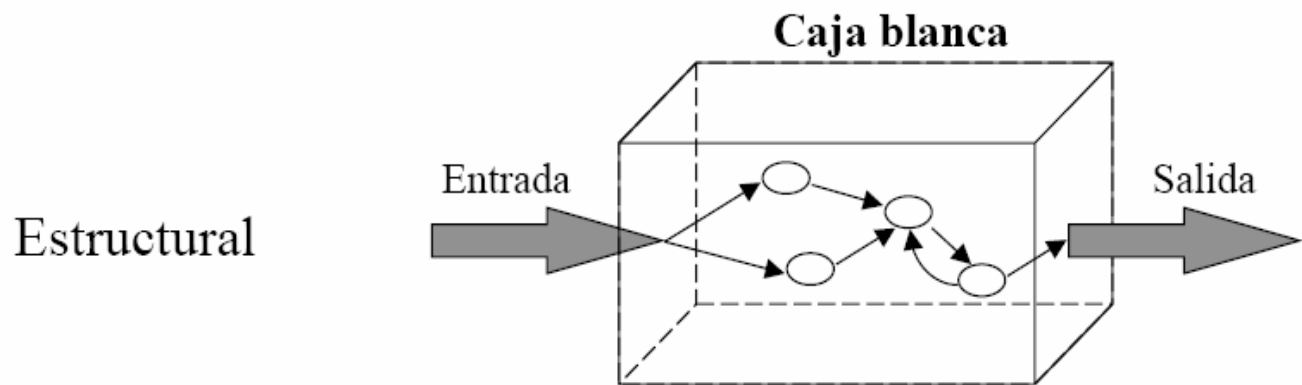


Fig.5. Esquema estructural de prueba de caja blanca

La prueba de caja blanca, denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, se pueden obtener casos de prueba que:

- ✓ Garanticen que se ejerciten por lo menos una vez, todos los caminos independientes de cada módulo
- ✓ Que se ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa
- ✓ Que se ejecuten todos los ciclos en sus límites y con sus límites operacionales
- ✓ Que se ejerciten las estructuras internas de datos para asegurar su validez.

El objetivo de estas pruebas es comprobar la parte estructural del software, para detectar errores en la implementación. Para el diseño de los casos de pruebas hay que basarse en la elección de caminos



importantes que ofrezcan una seguridad aceptable en descubrir defecto, y para ello se utilizan los llamados criterios de cobertura lógica.

En estas pruebas estamos siempre observando el código, que las pruebas se dedican a ejecutar con ánimo de "probarlo todo". Esta noción de prueba total se formaliza en lo que se llama "cobertura" y no es sino una medida porcentual de ¿cuánto código hemos cubierto?

Hay diferentes posibilidades de definir la cobertura. Todas ellas intentan sobrevivir al hecho de que el número posible de ejecuciones de cualquier programa no trivial es (a todos los efectos prácticos) infinito. Pero si el 100% de cobertura es infinito, ningún conjunto real de pruebas pasaría de un infinitésimo de cobertura

### **Criterios de cobertura lógica**

El análisis de cobertura del código es el proceso de encontrar fragmentos del programa que no son ejecutados por los casos de prueba, además permite crear casos de prueba adicionales que incrementen la cobertura. Posibilita también determinar un valor cuantitativo de la cobertura (que es, de manera indirecta, una medida de la calidad del programa). Adicionalmente, el análisis de cobertura también permite la identificación de casos de prueba redundantes, que no incrementan la cobertura.<sup>54</sup>

#### **• Cobertura de segmentos**

A veces también denominada "cobertura de sentencias". Por segmento se entiende una secuencia de sentencias sin puntos de decisión. Como el ordenador está obligado a ejecutarlas una tras otra, es lo mismo decir que se han ejecutado todas las sentencias o todos los segmentos.

El número de sentencias de un programa es finito. Basta coger el código fuente e ir contando. Se puede diseñar un plan de pruebas que vaya ejercitando más y más sentencias, hasta que hayamos pasado por todas, o por una inmensa mayoría.

---

<sup>54</sup> USAOLA, D. M. P. *Mantenimiento Avanzado de Sistemas de Información. Pruebas del Software* [Sitio Web]. Disponible en: <http://alarcos.inf-cr.uclm.es/doc/masi/doc/lec/parte5/polo-apuntesp5.pdf>.

En la práctica, el proceso de pruebas termina antes de llegar al 100%, pues puede ser excesivamente laborioso y costoso provocar el paso por todas y cada una de las sentencias.

A la hora de decidir el punto de corte antes de llegar al 100% de cobertura hay que ser precavido y tomar en consideración algo más que el índice conseguido. En efecto, ocurre con harta frecuencia que los programas contienen código muerto o inalcanzable. Puede ser que este trozo del programa, simplemente "sobre" y se pueda prescindir de él; pero a veces significa que una cierta funcionalidad, necesaria, es inalcanzable: esto es un error y hay que corregirlo.<sup>55</sup>

### • Cobertura de ramas

La cobertura de segmentos es engañosa en presencia de segmentos opcionales. Por ejemplo:

```
IF Condición THEN EjecutaEsto; END;
```

Desde el punto de vista de cobertura de segmentos, basta ejecutar una vez, con éxito en la condición, para cubrir todas las sentencias posibles. Sin embargo, desde el punto de vista de la lógica del programa, también debe ser importante el caso de que la condición falle (si no lo fuera, sobra el IF). Sin embargo, como en la rama ELSE no hay sentencias, con 0 ejecuciones tenemos el 100%.

Para afrontar estos casos, se plantea un refinamiento de la cobertura de segmentos consistente en recorrer todas las posibles salidas de los puntos de decisión. Para el ejemplo de arriba, para conseguir una cobertura de ramas del 100% hay que ejecutar (al menos) 2 veces, una satisfaciendo la condición, y otra no.

Estos criterios se extienden a las construcciones que suponen elegir 1 de entre varias ramas. Por ejemplo, el CASE.

Nótese que si se lograra una cobertura de ramas del 100%, esto llevaría implícita una cobertura del 100% de los segmentos, pues todo segmento está en alguna rama. Esto es cierto salvo en programas triviales

---

<sup>55</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

que carecen de condiciones (a cambio, basta 1 sola prueba para cubrirlo desde todos los puntos de vista). El criterio también debe refinarse en lenguajes que admiten excepciones (por ejemplo, Ada). En estos casos, hay que añadir pruebas para provocar la ejecución de todas y cada una de las excepciones que pueden dispararse.<sup>56</sup>

### • Cobertura de condición/decisión

La cobertura de ramas resulta a su vez engañosa cuando las expresiones booleanas que usamos para decidir por qué rama tirar son complejas. Por ejemplo:

```
IF Condicion1 OR Condicion2 THEN HazEsto; END;
```

Las condiciones 1 y 2 pueden tomar 2 valores cada una, dando lugar a 4 posibles combinaciones. No obstante sólo hay dos posibles ramas y bastan 2 pruebas para cubrir las. Pero con este criterio podemos estar cerrando los ojos a otras combinaciones de las condiciones.

Consideremos sobre el caso anterior las siguientes pruebas:

Prueba 1: Condicion1 = TRUE y Condicion2 = FALSE

Prueba 2: Condicion1 = FALSE y Condicion2 = TRUE

Prueba 3: Condicion1 = FALSE y Condicion2 = FALSE

Prueba 4: Condicion1 = TRUE y Condicion2 = TRUE

Bastan las pruebas 2 y 3 para tener cubiertas todas las ramas. Pero con ellos sólo hemos probado una posibilidad para la Condición1.

Para afrontar esta problemática se define un criterio de cobertura de condición/decisión que trocea las expresiones booleanas complejas en sus componentes e intenta cubrir todos los posibles valores de cada uno de ellos.

---

<sup>56</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

Nótese que no basta con cubrir cada una de las condiciones componentes, si no que además hay que cuidar de sus posibles combinaciones de forma que se logre siempre probar todas y cada una de las ramas. Así, en el ejemplo anterior no basta con ejecutar las pruebas 1 y 2, pues aun cuando cubrimos perfectamente cada posibilidad de cada condición por separado, lo que no hemos logrado es recorrer las dos posibles ramas de la decisión combinada. Para ello es necesario añadir la prueba 3.

El conjunto mínimo de pruebas para cubrir todos los aspectos es el formado por las pruebas 3 y 4. Aún así, nótese que no hemos probado todo lo posible. Por ejemplo, si en el programa nos colamos y ponemos AND donde queríamos poner OR (o viceversa), este conjunto de pruebas no lo detecta. Sólo queremos decir que la cobertura es un criterio útil y práctico; pero no es prueba exhaustiva.<sup>57</sup>

### • Cobertura de bucles

Los bucles no son más que segmentos controlados por decisiones. Así, la cobertura de ramas cubre plenamente la esencia de los bucles. Pero eso es simplemente la teoría, pues la práctica descubre que los bucles son una fuente inagotable de errores, todos triviales, algunos mortales. Un bucle se ejecuta un cierto número de veces; pero ese número de veces debe ser muy preciso, y lo más normal es que ejecutarlo una vez de menos o una vez de más tenga consecuencias indeseables. Y, sin embargo, es extremadamente fácil equivocarse y redactar un bucle que se ejecuta 1 vez de más o de menos.

Para un bucle de tipo WHILE hay que pasar 3 pruebas

1. 0 ejecuciones
2. 1 ejecución
3. más de 1 ejecución

Para un bucle de tipo REPEAT hay que pasar 2 pruebas

1. 1 ejecución
2. más de 1 ejecución

---

<sup>57</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

Los bucles FOR, en cambio, son muy seguros, pues en su cabecera está definido el número de veces que se va a ejecutar. Ni una más, ni una menos, y el compilador se encarga de garantizarlo. Basta pues con ejecutarlos 1 vez.

No obstante, conviene no engañarse con los bucles FOR y examinar su contenido. Si dentro del bucle se altera la variable de control, o el valor de alguna variable que se utilice en el cálculo del incremento o del límite de iteración, entonces eso es un bucle FOR con trampa.

También tiene "trampa" si contiene sentencias del tipo EXIT (que algunos lenguajes denominan BREAK) o del tipo RETURN. Todas ellas provocan terminaciones anticipadas del bucle.

Si el programa contiene bucles LOOP, o simplemente bucles con trampa, la única cobertura aplicable es la de ramas. El riesgo de error es muy alto; pero no se conocen técnicas sistemáticas de abordarlo, salvo reescribir el código.<sup>58</sup>

### • La cobertura de caminos

Es el criterio más elevado, cada uno de los posibles caminos del programa se debe ejecutar al menos una vez. Se define un camino como la secuencia de sentencias encadenadas desde la sentencia inicial del programa hasta su sentencia final. Para reducir el número de caminos a probar, se habla del concepto de **camino de prueba (test path)** que no es más que un camino del programa que atraviesa, como máximo, una vez el interior de cada bucle que encuentra. La idea en la que se basa consiste en que ejecutar un bucle más de una vez no supone una mayor seguridad de detectar defectos en él. Sin embargo, otros especialistas recomiendan que se pruebe cada bucle tres veces: una sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo dos veces. Esto último es interesante para comprobar cómo se comporta a partir de los valores de datos procedentes, no del exterior del bucle (como en la primera iteración), sino de las operaciones de su interior.

---

<sup>58</sup> MAÑAS, J. A. (1994). "Pruebas de Programas." Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

## El método de cobertura ciclométrica de caminos o método del camino básico.

La prueba del camino básico es una técnica de prueba de caja blanca propuesta inicialmente por Tom McCabe. El método del camino básico 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. Es el único método caja blanca que cubriremos. Se basa en construir un caso de prueba por camino básico que se encuentre en el grafo de programa asociado al método de la clase que se desea someter a pruebas. Los pasos del método son:

- ✓ **.Dibujar el grafo G del programa del método de la clase.**

Para dibujar este grafico es necesario tener en cuenta, las siguientes sintaxis:

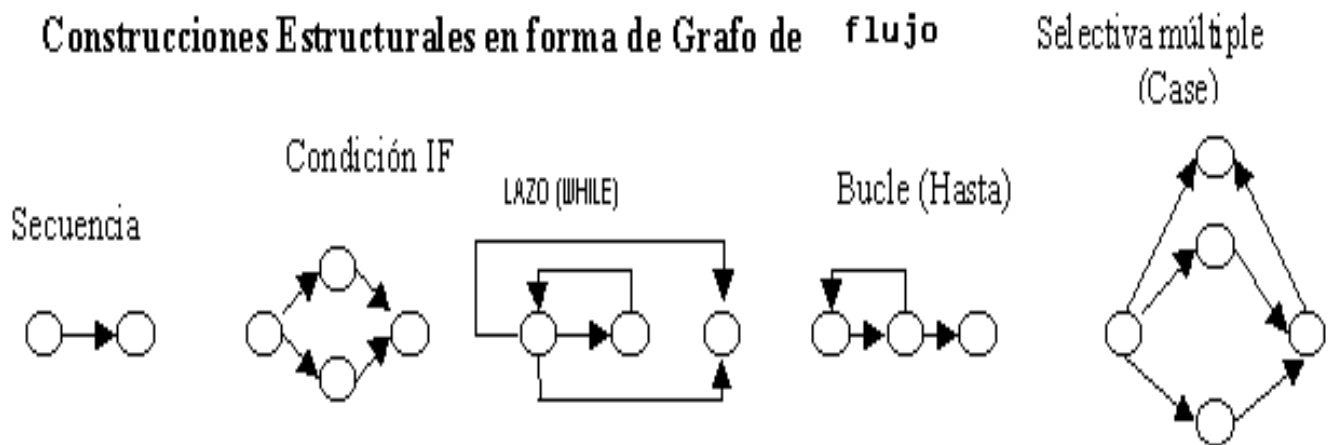
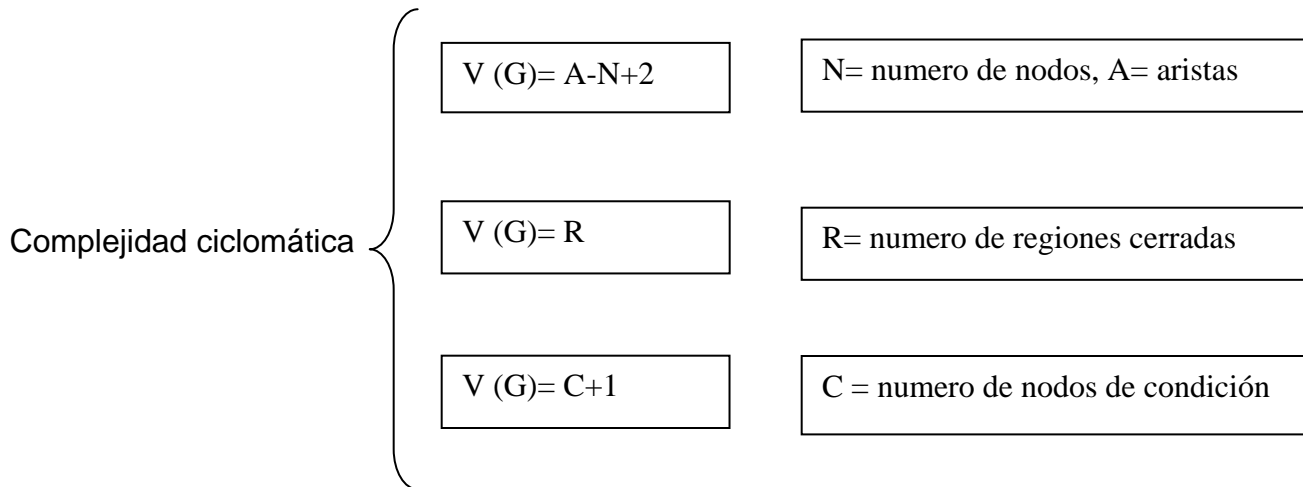


Fig. 6 Notación de grafos de flujo para las instrucciones SECUENCIALES, IF, WHILE, REPETIR, CASE

- ✓ **Determinar el número ciclomático del grafo,  $V(G)$ . Este numero se puede calcular de varias formas:**



- ✓ **Construir una secuencia de  $V(G)$  caminos linealmente independientes en  $G$ .**
  - (a) El primer camino es cualquiera de los caminos mínimos del nodo origen del grafo a uno de los nodos de terminación del grafo. Se inicializa  $A(G)$ , el conjunto de aristas en la secuencia linealmente independiente de  $G$  con todas aquellas aristas que forman este primer camino.
  - (b) El próximo camino se forma agregando un nuevo camino entre el origen y una terminación de  $G$  Este nuevo camino debe agregar el mínimo número posible de aristas nuevas  $A$ , (pero que agregue por lo menos una arista nueva).
- ✓ **Preparar un caso de prueba por camino hallado en el paso anterior.**
  - (a) Determinar los datos a proporcionar como entrada para ejecutar el camino hallado.

(b) Usando la especificación funcional del método, indicar cuál es el resultado esperado.<sup>59</sup>

Un *grafo* de flujo  $G$ , es un par ordenado de  $V$  y  $A$ , donde  $V$  es el conjunto de vértices o nodos del grafo y  $A$  es un conjunto de aristas, a estas también se les llama arcos o ejes del grafo. Un vértice puede tener 0 o más aristas, pero toda arista debe unir exactamente a dos vértices.

Los grafos representan conjuntos de objetos que no tienen restricción de relación entre ellos. Un grafo puede representar varias cosas de la realidad cotidiana, tales como mapas de carreteras, vías férreas, circuitos eléctricos, etc.

La notación  $G = A(V, A)$  se utiliza comúnmente para identificar un grafo.

Los grafos se constituyen principalmente de tres partes: aristas, *nodos* y *regiones* que pueda contener el mismo grafo, formadas por las aristas.

### **Aristas**

Son las líneas con las que se unen los nodos de un grafo y con la que se construyen también caminos.

### **Nodos**

Cada círculo representado se denomina nodo del Grafo de Flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder a una secuencia de procesos o a una sentencia de decisión. Puede ser también que hallan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

### **Regiones**

Las regiones son zonas delimitadas por las aristas del grafo incluyendo la zona exterior del mismo.

---

<sup>59</sup> JOSÉ, F. M. and G. J. PABLO. (1999). "Sistemas de Programas." Consultado: 12 de abril de 2007, 2007, Disponible en: <http://www ldc.usb.ve/~teruel/ci3711/test1/index.html#3>.



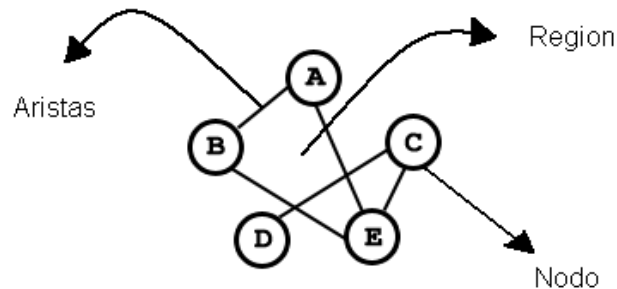


Fig. 7 Representación de grafos con sus componentes.

## Como obtener un grafo de flujo a través del código

Para identificar los caminos que tiene un programa conviene utilizar un grafo que represente el mismo. Un nodo del grafo será una condición (de una sentencia condicional o de un bucle) o un conjunto de sentencias tal que no pueda ejecutarse una sin que se ejecuten las demás. Los arcos del grafo serán líneas que reflejen el camino que puede seguir la ejecución del programa.

### Ejemplo:<sup>60</sup>

```

PROCEDURE imprime _ media (VAR x, y: real;)
VAR resultado: real;
resultado:=0;                               (1)
IF (x < 0 OR y < 0)                          (2 y 3)
THEN
WRITELN ("x e y deben ser positivos");      (4)
ELSE
resultado:= (x+y)/2                          (5)
WRITELN ("La media es: ", resultado);      (5)
ENDIF                                       (6)
END imprime _ media
    
```

<sup>60</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20de%20SW.ppt>

Los nodos a los que pertenece cada instrucción han sido numerados del 1 al 6. Se enumeran las asignaciones y luego las condiciones en el orden en que aparezcan.<sup>61</sup>

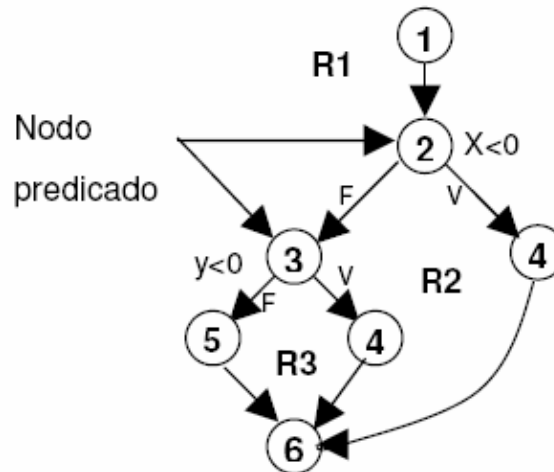


Fig. 8 Representación del grafo de flujo del ejemplo anterior

Calculando la complejidad ciclomática del grafo de cualquiera de las tres formas antes mostradas se tiene que:  $V(G)=3$ , por lo tanto hay que determinar tres caminos independientes.

**Camino independiente** es aquel que introduce por lo menos una sentencia de procesamiento (o valor de condición) que no estaba considerada.<sup>62</sup>

En este caso se tienen estos tres caminos:

**Camino 1: 1-2-3-5-6**

**Camino 2: 1-2-4-6**

**Camino 3: 1-2-3-4-6**<sup>63</sup>

<sup>61</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

<sup>62</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

<sup>63</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

Una vez obtenidos los caminos ya se pueden efectuar los casos de pruebas para cada camino encontrado:

### Casos de pruebas

**Camino 1:** Escoger algún **X** y **Y** tales que se cumpla  $X \geq 0$  AND  $Y \geq 0$

Respuesta:  $x=25$   $y=1$

**Camino 2:** Escoger algún **X** tal que se cumpla  $x < 0$

Respuesta:  $X = -2$

**Camino 3:** Escoger algún **X** y **Y** tales que se cumpla  $X \geq 0$  AND  $Y < 0$

Respuesta:  $X = 10$   $Y = -20$ <sup>64</sup>

Luego de confeccionar los casos de pruebas se ejecutan cada uno de ellos y se obtienen los resultados que son los encargados de decir si el código del proyecto está funcionando en buen estado.

Aunque la prueba de caja blanca no se vaya a aplicar, por que el software todavía está en construcción se muestra como es que se debe realizar esta prueba pues es una de las pruebas más importantes que se debe efectuar a la hora de medir la confiabilidad del software.

### 2.3.1.2 Prueba de caja negra

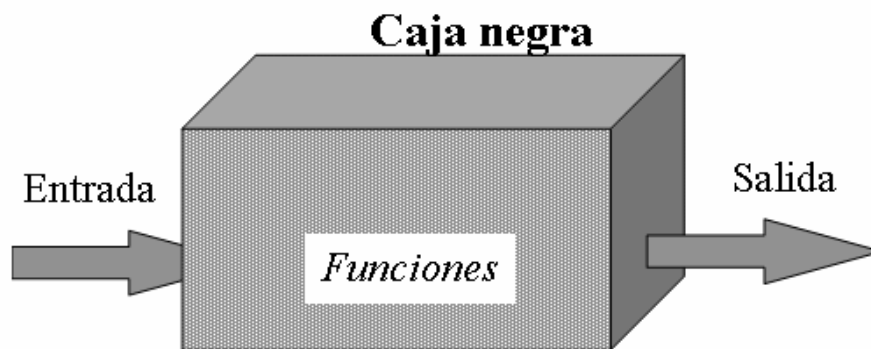


Fig. 9 Esquema de prueba de caja negra.

<sup>64</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

Las técnicas de pruebas de caja negra consisten en seleccionar los casos de prueba basándose en la funcionalidad esperada del mismo, es decir, en la especificación de lo que debe hacer el programa. El objetivo de estas pruebas es comprobar si existen errores en la funcionalidad del software, es decir se comprueban las interfaces del mismo para detectar defectos de función en ellas. Antes de realizar un programa conviene tener por escrito qué es lo que debe hacer el mismo (documento de especificación *software*), las pruebas de caja negra deben emplear esta documentación para seleccionar los casos de prueba. La prueba de todos los casos posibles es inviable, por lo que, al igual que para las pruebas de caja blanca, se debe realizar una selección de los casos que se van a utilizar, para ello se emplean dos técnicas:

✚ **Selección de clases de equivalencia.** Esta técnica intenta seleccionar casos de prueba que sean representativos de un conjunto de datos de entrada.

El primer paso consiste en seleccionar las clases de equivalencia tales que todos los casos de prueba de una misma clase se comporten, por lo menos *a priori*, de la misma forma. Una vez identificadas las clases hay que elegir un caso de prueba de cada una de ellas.

La partición equivalente es un método que divide el campo de entrada de un programa en clases de datos. **Una condición de entrada** es un valor numérico específico, un rango de valores, un miembro de un conjunto de valores o lógica. **Una clase de equivalencia** representa un conjunto de estados válidos y no válidos para una condición de entrada. La prueba de partición equivalente se basa en evaluar las clases de equivalencia para una condición de entrada

### Pasos para aplicar esta técnica

#### 1. Identificar Clases de Equivalencia

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

- Si un parámetro de entrada debe estar comprendido en un cierto rango (Rango), aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.

- Si una entrada requiere un valor concreto (Valor específico), aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.
- Si una entrada requiere un valor de entre los de un conjunto (Miembro de conjunto), aparecen 2 clases de equivalencia: en el conjunto o fuera de él.
- Si una entrada es booleana (lógica), hay 2 clases: si o no.

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

- ✓ Se examina cada condición de entrada y se divide en dos o más grupos.
- ✓ Se identifican dos tipos de clases:
  - Clases de equivalencia válidas
  - Clases de equivalencia no válidas<sup>65</sup>

Condición de entrada	Clases de Equivalencia Válidas	Clases de Equivalencia No Válidas
<i>[Se escribe la condición de entrada ]</i>	<i>[Se determinan las clases que el sistema debe aceptar ]</i>	<i>[Se determinan las clases que el sistema no debe aceptar ]</i>

Tabla 1. Formato de prueba de caja negra <sup>66</sup>

2. Asignar un número único a cada clase de equivalencia
3. Escribir casos de prueba que cubran tantas clases válidas no incorporadas como sea posible hasta que se cubran todas las clases de equivalencia válidas.
4. Escribir casos de prueba que cubran una sola clase no válida no incorporada hasta que se cubran todas las clases de equivalencia no válidas.<sup>67</sup>

<sup>65</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

<sup>66</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

### Ejemplo: <sup>68</sup>

Un usuario puede conectarse al banco por Internet y realizar una serie de operaciones bancarias. Una vez accedido al banco con las consiguientes medidas de seguridad (clave de acceso y demás), se requiere la siguiente entrada:

- *Código del banco.* En blanco o número de tres dígitos. En este último caso, el primero de ellos tiene que ser mayor que 1
- *Código de sucursal.* Un número de cuatro dígitos. El primero de ellos mayor de 0
- *Número de cuenta.* Número de cinco dígitos
- *Clave personal.* Valor alfanumérico de cinco posiciones. Este valor se introducirá según la orden que se desee realizar
- *Orden.* Puede estar en blanco o ser una de las dos cadenas siguientes:
  - “Talonario”
  - “Movimientos”

En el primer caso el usuario recibirá un talonario de cheques, mientras que en el segundo recibirá los movimientos del mes en curso. Si este código está en blanco, el usuario recibirá los dos documentos.


---

<sup>67</sup> Letelier, P. (2006). “Pruebas de software.” Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

<sup>68</sup> Letelier, P. (2006). “Pruebas de software.” Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

<b>Condición de Entrada</b>	<b>Tipo</b>	<b>Clase Equivalencia Válida</b>	<b>Clases Equivalencia No Válida</b>
Código banco	Lógica (puede estar o no) Si está es Rango	1: En blanco 2: 100<= Código banco <= 999	3: Un valor no numérico 4: Código banco < 100 5: Código banco > 999
Código sucursal	Rango	6: 1000 <= Código sucursal <= 9999	7: Código sucursal < 1000 8: Código sucursal >= 9999
Nº Cuenta	Valor	9: Cualquier número de cinco dígitos	10: Número de más de cinco dígitos 11: Número de menos de cinco dígitos
Clave	Valor	12: Cualquier cadena de caracteres alfanuméricos de 5 posiciones	13: Cadena de menos de cinco posiciones 14: Cadena de más de cinco posiciones
Orden	Conjunto, con comportamiento distinto	15: "" 16: "Talónario" 17: "Movimientos"	18: Cadena distinta de blanco y de las válidas

Tabla 2. Ejemplo de prueba de caja negra<sup>69</sup>

 **Análisis de los valores límite.** Esta técnica intenta seleccionar los casos de prueba que examinan las restricciones límites del programa tanto en la entrada como en la salida del mismo. La regla básica para la selección de los casos es elegir cada condición de los datos de entrada y de los datos de salida y seleccionar los valores que están justo a ambos lados de la condición, los que la cumplen y los que no la cumplen.

<sup>69</sup> Letelier, P. (2006). "Pruebas de software." Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

- Si los datos de entrada deben estar dentro de un rango de valores, como puede ser entre 1 y 10, definir un caso para el 1, otro para el 0, otro para el 10 y otro para el 11. La misma regla será aplicable a los datos de salida, aunque para éstos lo que se debe intentar es escoger unos datos de entrada que hagan que la salida dé como resultado los valores mencionados.
- Si se especifica un número de valores, como, por ejemplo, no se pueden dar más de 10 pares de números, seleccionar un caso de prueba que permita dar 10 pares de números y otro que permita dar 11. Lo mismo será aplicable para la salida. La técnica de los valores límite complementa la de la selección de clases de equivalencia, pues lo que intenta es, en lugar de elegir cualquier valor como representante de una clase de equivalencia, seleccionar aquellos que están en los límites de la misma.

La técnica de Análisis de Valores Límites selecciona casos de prueba que ejerciten los valores límite, además complementa la prueba de partición equivalente. En lugar de realizar la prueba con cualquier elemento de la partición equivalente, se escogen los valores en los bordes de la clase y se derivan tanto casos de prueba a partir de las condiciones de entrada como con las de salida.

Las pruebas de caja negra se centran en los requisitos funcionales del software. La prueba de la caja negra intenta encontrar errores de los siguientes tipos:

- Funciones incorrectas o inexistentes
- Errores relativos a las interfaces
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores debidos al rendimiento
- Errores de inicialización o terminación

De las dos técnicas anteriormente explicadas la más efectiva es la de la partición equivalente, con esta técnica se pueden examinar los valores válidos e inválidos de las entradas existentes en el *software*, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de clases de prueba que hay que desarrollar.



### 2.3.2 Pruebas de Integración

Estas pruebas demuestran la integración entre los módulos de un sistema, se puede decir que es una representación de caja negra, pero solo se prueba que exista vínculo entre las interfaces de cada módulo y con la base de datos. El software SIMDEC constará con cuatro módulos que son: Administración, Diseño, Publicador y Monitoreo, cada uno de ellos depende del otro, es decir el trabajo que se realiza en el modulo de diseño depende del realizado en el de Administración, como también para que en el módulo de Publicador se pueda ejecutar el trabajo previsto se necesita la información del modulo de diseño, y a su vez en Monitoreo se necesitan los datos del módulo Publicador. Por estas particularidades del software no se puede utilizar la plantilla propuesta anteriormente para la prueba de caja negra debido a que se hace difícil determinar las clases validas e invalidas para la relación entre los módulos ya que la comunicación entre ellos se realiza a través de los datos introducidos o modificados en cada uno de ellos. Para el diseño de esta prueba es necesario navegar por la aplicación con el objetivo de poder comprobar los errores y defectos que puedan existir en la misma. El objetivo de estas pruebas es comprobar que los módulos que deben relacionarse entre sí, estén integrados unos con otros.

### 2.2.3 Pruebas de sistema

#### 2.3.3.1 Prueba de Usabilidad

Se trata de pruebas efectuadas con usuarios, con el objetivo de determinar si la organización de los contenidos y las funcionalidades que se ofrecen desde la aplicación son entendidas y utilizadas por los usuarios de manera simple y directa.

Las pruebas tradicionales son:

##### **Prueba Inicial:**

Para ver cómo funciona la organización de contenidos y elementos iniciales de diseño (botones, interfaces). El material con que se prueba es una imagen dibujada del Sitio Web.

### Prueba de Boceto Web:

Para ver si se entiende la navegación, si se pueden cumplir tareas y si el usuario entiende todos los elementos que se le ofrecen. El material con que se prueba es una maqueta web semi funcional.

En ambos casos la prueba consiste en mostrar a un grupo de personas el Sitio Web y hacerles preguntas sobre lo que ellos imaginan existe allí. Hay que recordar que en esta etapa del desarrollo las funcionalidades no existen como tales, aunque están definidas. Por lo mismo, todo el trabajo tiene que ver con los aspectos visuales y de organización de los contenidos.

Los resultados de cada una de esas etapas permitirán adecuar los elementos con los que se esté trabajando en esos momentos, con el fin de atender a los usuarios y ofrecerles una experiencia a la altura de sus expectativas.

Es importante enfatizar en estas pruebas, ya que generan insumos que serán muy útiles y permitirán darse cuenta a tiempo de errores conceptuales en la entrega de la información, que puedan ser remediados de manera temprana y sin afectar el desarrollo total del proyecto.

Como en este trabajo el Sistema de Manejo de Datos de Ensayos Clínicos se encuentra en la fase de análisis y diseño, se perfilara la prueba inicial ya que para la prueba de boceto web es necesario presentar una maqueta web semifuncional y dada la etapa de elaboración en que se encuentra será imposible disponer de este elemento. Para la delineación de la prueba inicial se utilizaran los prototipos no funcionales definidos hasta el momento.

Para la elaboración de esta prueba se redactaron una serie de preguntas las cuales sirven de apoyo al usuario en cuanto a su orientación para el posterior manejo de la aplicación y a su vez sirven como fuente para corregir las irregularidades que se detecten con el objetivo de que el sistema de computo sea entendido y utilizado por los usuarios de manera simple y directa.


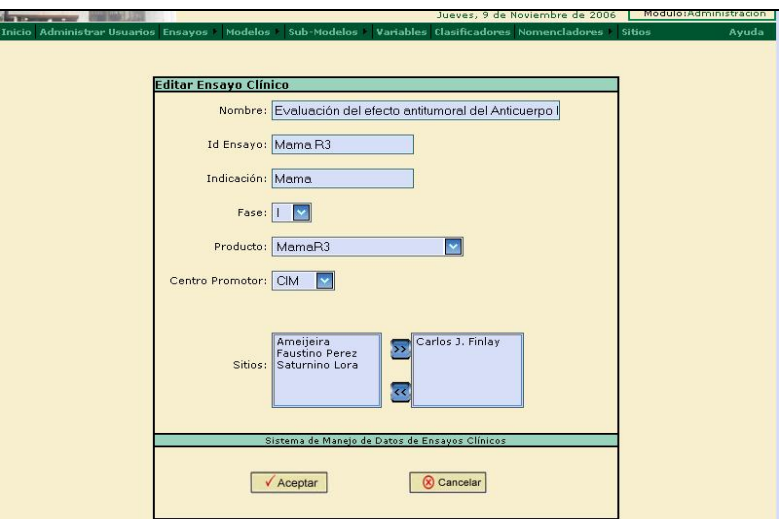
Preguntas Estándares	Interfaz
<p>1. ¿De la interfaz mostrada se puede inferir su funcionalidad? ¿Por qué?</p> <p>2. ¿Faltaría algún campo que permita enmarcar más la información mostrada?</p> <p>3. ¿Tienen un orden lógico los campos visualizados?</p> <p>4. ¿Qué funcionalidad se puede deducir de la imagen expuesta?</p> <p>5. ¿Las interfaces reveladas conjuntamente con cada campo incluyendo las líneas de texto publicadas en la imagen indican lo que se quiere hacer con la imagen plasmada?</p>	
	

Tabla 3. Formato de prueba de usabilidad

### 2.3.3.2 Pruebas de rendimiento

Hay pruebas de Carga y Estrés concernientes a la faceta de pruebas de "Sistemas o Producción", que analizan el rendimiento de un sistema, proyecto o aplicación. El objetivo de estas pruebas es comprobar si el software es capaz de soportar la ejecución simultánea de un mismo camino por cierta cantidad de usuarios para verificar en que punto colapsa el sistema.

**Carga:** se determina el número de usuarios que se desea intervengan a la vez en un camino crítico (o serie de funcionalidades más importantes que necesitan mayor atención). Se van simulando incorporaciones graduales de usuarios (carga de 10 en 10 o 20 en 20,...) que ejecutan el mismo camino. La prueba indica la degradación que va sufriendo el sistema (en toda su estructura, servidores físicos, comunicaciones, bases de datos, sistemas operativos, servidores aplicaciones,...) hasta detectar el punto óptimo, comportamiento en el nº de usuarios necesario, a qué nivel se cae el sistema, componentes críticos, etc.

Para la aplicación de esta prueba es necesario que los usuarios se vayan incorporando al sistema gradualmente y para tener un control del momento en el que se van incorporando es importante tener en cuenta la hora de entrada, con el objetivo de marcar la diferencia y dar a conocer que los usuarios no entraron simultáneamente.

<b>Cantidad de usuarios que entran al sistema</b>	<b>Resultados del software</b>	<b>Resultados esperados</b>	<b>Hora de entrada al sistema</b>
<i>[Se escribe la cantidad de usuarios que van entrando al sistema, representados por un número.]</i>	<i>[Durante la aplicación de la prueba se describe detalladamente que ha ocurrido realmente, cómo es que ha reaccionado el sistema ante cada entrada, referenciando detalles.]</i>	<i>[Se describe detalladamente como se espera reaccione el sistema ante cada entrada específica]</i>	<i>[Se anota la hora exacta en que cada usuario entra al sistema]</i>

Tabla 4. Formato propuesto para prueba de carga.

Estrés: es parecido a la prueba anterior pero en vez de que los usuarios entren al sistema gradualmente, entran a la vez (un grupo de 20 en el mismo instante).

Número de usuarios conectados al sistema	Resultados del software	Resultados esperados
<i>[Se escribe la cantidad de usuarios que están conectados simultáneamente al sistema, representados por un número.]</i>	<i>[Durante la aplicación de la prueba se describe detalladamente que ha ocurrido realmente, cómo es que ha reaccionado el sistema, referenciando detalles.]</i>	<i>[Se describe detalladamente como se espera reaccione el sistema ante el trabajo simultaneo de todos los usuarios en el sistema.]</i>

Tabla 5. Formato propuesto para prueba de Estrés.

### 2.3.3.3 Prueba de seguridad

Las pruebas de seguridad son una forma de garantizar la integridad de los datos que se maneja en SIMDEC. El objetivo de las pruebas de seguridad, para el caso específico del SIMDEC, es comprobar la seguridad dentro de la aplicación, es decir, evidenciar que solamente puedan acceder a la aplicación usuarios registrados, y que cada uno de ellos tenga acceso a la información que le corresponde de acuerdo al rol que desempeña dentro del sistema. De modo general se puede decir que a través de estas pruebas se garantiza que la aplicación siga funcionando correctamente frente a cualquier ataque, que custodie los datos que maneja y los proteja frente a manipulación consciente o inconsciente de los usuarios y que su disponibilidad esté garantizada.<sup>70</sup>

<sup>70</sup> Pérez, P. G. (2005). Principios básicos del desarrollo seguro. Retrieved 6 de junio de 2007. 2007, from [http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

### **2.3.3.3 Etapas de aplicación de las pruebas**

Las pruebas a los productos software se le pueden aplicar a lo largo del ciclo de desarrollo de los mismos, las pruebas anteriormente mostradas, aplicables al Sistema de Manejo de Datos de Ensayos Clínicos, también se ponen en práctica en diferentes fases de desarrollo.

La prueba de caja blanca, se aplica en la fase de Construcción dentro del flujo de trabajo de Implementación, porque esta prueba consiste en medir el código del sistema y es en esa fase donde se puede poner en práctica dicha prueba, la prueba de caja negra se aplica dentro de la fase de transición, cuando el software esta terminado ya, porque esta prueba es para medir la funcionalidad del producto, apoyándonos en las diferentes interfases del mismo, esta prueba es muy eficiente, porque la interfaz es lo primero que el usuario ve y prueba del sistema, y a través de ella se pueden corregir los diferentes errores en la funcionalidad del producto.

La prueba de Integración se hace cuando algunos módulos estén terminados es decir, se puede aplicar en cualquier fase de desarrollo, esta prueba solo depende de la terminación de 2 módulos o más para poder ponerse en práctica, con ella se infiere la integración de los diferentes módulos que pueda tener la aplicación en específico, lo cual es muy importante porque el vínculo entre módulos es lo que le da, en conjunto con la prueba de caja negra, la total funcionalidad al sistema.

La prueba de usabilidad se puede llevar a cabo en la fase de elaboración dentro del flujo de trabajo de análisis y diseño, para esta prueba hace falta la opinión del cliente, se trabaja con los prototipos no funcionales del sistema lo cual hace posible que el cliente de el visto bueno antes de hacer la aplicación como tal, esta prueba también es importante por que le da la posibilidad al probador de informarles a los desarrolladores los posibles errores que el cliente detectó y así llevar a cabo el desarrollo del producto con un mínimo de errores y la prueba de rendimiento se puede llevar a cabo en la fase de construcción o en la de transición. Esta prueba también es muy importante porque con ella se puede medir hasta donde puede aguantar la aplicación, es decir si sostiene una cantidad de usuarios determinada, esto da la posibilidad que el sistema no colapse en medio de una ejecución de alguna interfaz.

Las pruebas de seguridad se pueden tener en cuenta durante todo el ciclo de desarrollo del producto, aunque en este caso, para el software SIMDEC se propone que se hagan en la fase de transición, es decir, ya cuando el producto está listo para ser entregado.

Las pruebas anteriormente explicadas tienen una particularidad, que son pruebas destinadas a comprobar la usabilidad del software.

### **2.4 Conclusiones**

En este capítulo 2 se propusieron las pruebas necesarias para evaluar la usabilidad del Sistema de manejo de datos de Ensayos Clínicos. Para proponer estas pruebas se tuvieron en cuenta requisitos y especificaciones del propio software así como los principios para la validación de software planteados por la FDA (Food and Drug Administration). Con este capítulo se puede concluir que si en años posteriores estas pruebas se llegan a aplicar en varias iteraciones quedará probada la funcionalidad y usabilidad del software brindándole al cliente una correcta ejecución del mismo. Con estas pruebas se demuestra, casi en su totalidad, el funcionamiento como un todo del sistema, teniendo presente los aspectos con más probabilidad de errores pueden poseer.

## CONCLUSIONES

Es imprescindible adelantarse al defecto cuando se habla de software, más aun cuando implica calidad de vida por encima de calidad tecnológica y funcional, defectos que luego arrastran presupuesto, personas, tiempo, oportunidades, riesgos. Un ambiente agresivo de comercialización como el imperante en la actualidad requiere mostrar la mejor carta de triunfo, que en este caso es el software probado, corregido, no solo una vez sino retroalimentado tantas veces como sea necesario que a su vez conlleve a ser funcional, seguro, preciso. Hay un compromiso humano llamado utilidad y es más que un negocio aunque implique recursos monetarios, en nuestro país es importante una salud sin fallos, donde los avances no se maltraten con causas evitables. Este producto encierra gran sensibilidad en cualquier tipo de resultado erróneo o vulnerable por inconsistencias, son los sueños de mucha gente luchando con la incertidumbre de vivir. La búsqueda de altos niveles de calidad es buscar excelencia y solo se logra cuando la estrategia es óptima, en el caso de las pruebas de software es esencial aplicarlas adecuadamente porque los productos requieren comprobaciones específicas en cada caso y fase de desarrollo, en la investigación se concluye que las pruebas necesarias son en base a la funcionalidad y usabilidad, la selección y estudio de las técnicas de prueba realizado al SIMDEC es correcto y de gran importancia para la etapa de despliegue del mismo. Como punto de partida en calidad para perfil Bioinformática funcionará como guía al equipo de pruebas. Específicamente de las pruebas de software se pueden decir a modo de conclusión final:

- Que el estudio de los posible tipos de pruebas para este software garantiza la obtención de pruebas adecuadas para este sistema según lo planteado por la agencia regulatoria
- Que pruebas de software existen muchas pero cada software tiene sus particularidades y en dependencia de ellas son las pruebas que se diseñan y aplican.
- Las pruebas propuestas para el Sistema de Manejo de Datos de Ensayos Clínicos son las adecuadas para garantizar la usabilidad y funcionalidad del producto en específico.



### RECOMENDACIONES

El diseño de las pruebas de software, es un elemento fundamental para su posterior aplicación, dada la importancia de la detección de errores en las aplicaciones software, a todas aquellas personas o empresas productoras de software, se recomienda:

- ✓ Tratar de establecer correctos diseños de pruebas con el fin de corregir todos los posibles defectos del sistema.
- ✓ Aplicar las pruebas propuestas a este producto de gran importancia para el sector de la salud en nuestro país.
- ✓ Adaptar estas pruebas a nuevos productos de la misma línea para lograr un estándar de pruebas que demuestren la usabilidad de los sistemas.
- ✓ Diseñar otras pruebas que garanticen la seguridad y confiabilidad de software en su totalidad.
- ✓ Que el documento obtenido como resultado de esta investigación quede como fuente de consulta y bibliografía en este tema de pruebas de software.

## REFERENCIAS BIBLIOGRÁFICAS

CONDE, I. B. *Bioética en ensayos clínicos. Su aplicación actual*. Última actualización: 21 de abril de 1998. [Consultado el 17 de octubre de 2006]. Disponible en:

[http://bvs.sld.cu/revistas/mgi/vol14\\_4\\_98/mgi07498.htm](http://bvs.sld.cu/revistas/mgi/vol14_4_98/mgi07498.htm)

PAZ, A. C. *El modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial* [Sitio Web]. [Consultado el 21 de enero de 2007]. Disponible en:

<http://www.monografias.com/trabajos5/call/call.shtml#algu>.

PRESSMAN, R. S. *Ingeniería del Software. Un enfoque practico*. Quinta edición. ed. McGraw Hill, 2005. vol. I, 500 p

SALANOVA, P. E. (2006). *Modelos de Calidad Web. Clasificación de Métricas.*, UNIVERIDAD NACIONAL DE EDUCACIÓN A DISTANCIA: 308

GUTIÉRREZ, J. J.; ESCALONA, M. J., et al. *Estudio comparativo de propuestas para la generación de casos de prueba a partir de requisitos funcionales*. [Consultado el: 25 de enero de 2007 de 2007]. Disponible en: <http://www.lsi.us.es/docs/informes/LSI-2005-01.pdf>.

SCALONE, L. F. *ESTUDIO COMPARATIVO DE LOS MODELOS Y ESTANDARES DE CALIDAD DEL SOFTWARE*. Tutor: Martínez, D. R. G. Tesis de maestría, UNIVERSIDAD TECNOLÓGICA NACIONAL, FACULTAD REGIONAL BUENOS AIRES, 2006

COMPUWARE. *LAS EMPRESAS EUROPEAS ASUMEN YA LA CALIDAD DEL SOFTWARE PERO NO SABEN APLICAR METODOLOGIAS CONSISTENTES* [Sitio Web]. [Consultado el: 15 de marzo de 2007 de 2007]. Disponible en: <http://www.noticias.com/notaprensa/28-04-2006/bdi->

[comunicacion/empresas-europeas-asumen-ya-calidad-software-pero-no-saben-aplicar-metodologias-consistentes-c29.html](http://comunicacion/empresas-europeas-asumen-ya-calidad-software-pero-no-saben-aplicar-metodologias-consistentes-c29.html).

ESTRADA, A. F., S. A. CÁRDENAS, et al. (2006). *“Calidad de Software y la empresa, enseñanza de un tema imprescindible para el ingeniero Informático.”* Consultado en: 1 de diciembre de 2006, 2006, Disponible en: <http://www.somece.org.mx/memorias/2000/docs/123.DOC>

LOVELLE, J. M. C. (1999). *“Calidad del software.”* Consultado: 19 de abril de 2007, 2007, Disponible en: [http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad\\_software.PDF](http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF)

BILELLO, M. A. *El valor de cada uno.* [Sito Web]. Consultado: 11 de diciembre de 2006, 2006, Disponible en: [http://bloggers.com.ar.elsevier.com/system/noticia\\_detalle.php?id\\_prod=607](http://bloggers.com.ar.elsevier.com/system/noticia_detalle.php?id_prod=607).

REYES., I. B. y TORRES., I. N. *Las pruebas de software, su aplicación al Config. CASE.* .Tutor. Estrada., M. A. F. TRABAJO DE DIPLOMA, INSTITUTOSUPERIOR POLITÉCNICO JOSÉ ANTONIO ECHEVARRÍA, 2003.

IEEE, IEEE Std1995, Metrics, IEEE, 1991.

LETELIER, P. (2006). *“Pruebas de software.”* Consultado 7 de diciembre de 2006, 2006, disponible en: <https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

D’ONOFRIO, D. L. (2003). *“Probando software y números de versión.”* Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.masterdisseny.com/master-net/librecom/index.php3>

MAÑAS, J. A. (1994). *“Pruebas de Programas.”* Consultado: 24 de enero de 2007, 2007, Disponible en: <http://www.it.uc3m.es/tsps/testing.htm#s1>

CORTÉS, O. H. G. (2004). “*Aplicación práctica del diseño de pruebas de software a nivel de programación.*” Consultado: 26 de enero de 2007, 2007, Disponible en:

[http://www.willydev.net/descargas/oguzman-diseno\\_pruebas.pdf](http://www.willydev.net/descargas/oguzman-diseno_pruebas.pdf)

PÚBLICAS, M. d. A. “*Implantación y Aceptación del Sistema.*” Consultado: 25 de enero de 2007, 2007, Disponible en: <http://www.csi.map.es/csi/metrica3/iasproc.pdf>

GUTIÉRREZ, J. J., M. J. ESCALONA, et al. (2004). “*Aplicando Técnicas de Testing en Sistemas para la Difusión Patrimonial.*” Consultado: 26 de enero de 2007, 2007, Disponible en:

[http://www.turismo.uma.es/turitec/turitec2004/docs/actas\\_turitec\\_pdf/15.pdf](http://www.turismo.uma.es/turitec/turitec2004/docs/actas_turitec_pdf/15.pdf)

PÉREZ P. G. (2005). *Principios básicos del desarrollo seguro.* Consultado: 6 de junio de 2007. 2007, Disponible en

[http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

COLLADO, M. (2003). “*Pruebas de software. Técnicas de prueba del software. Estrategia de prueba del software.*” Consultado: 17 de febrero de 2007, 2007. Disponible en:

<http://lml.ls.fi.upm.es/ftp/ed2/0203/Apuntes/pruebas.ppt>

JOSÉ, F. M. and G. J. PABLO. (1999). “*Sistemas de Programas.*” Consultado: 12 de abril de 2007, 2007, Disponible en: <http://www.ldc.usb.ve/~teruel/ci3711/test1/index.html#3>.

UNIDOS, A. P. L. A. Y. M. L. E. *Centro para la Seguridad Alimentaria y la Nutrición Aplicada.* Última actualización: 24 de enero de 2005. [Consultado el 25 de marzo de 2007]. Disponible en: <http://www.cfsan.fda.gov/~mow/scfsan4.html>.

U.S. DEPARTAMENT OF HEALTH AND HUMAN SERVICES. *U.S. Food and Drug Administration* [Web Site]. Consultado el: 14 febrero de 2007. Disponible en: <http://www.fda.gov/oc/spanish/>

USAOLA, D. M. P. *Mantenimiento Avanzado de Sistemas de Información. Pruebas del Software* [Sitio Web]. Disponible en: <http://alarcos.inf-cr.uclm.es/doc/masi/doc/lec/parte5/polo-apuntesp5.pdf>.

## BIBLIOGRAFÍA

1. CONDE, I. B. *Bioética en ensayos clínicos. Su aplicación actual.*  
[http://bvs.sld.cu/revistas/mgi/vol14\\_4\\_98/mgi07498.htm](http://bvs.sld.cu/revistas/mgi/vol14_4_98/mgi07498.htm)
2. PAZ, A. C. *Modelo de mccall como aplicación de la calidad a la revisión del software de gestión empresarial.*  
<http://www.monografias.com/trabajos5/call/call.shtml#algu>.
3. PRESSMAN, R, S. *Ingeniería del Software. Un enfoque practico.*
4. SALANOVA, P. E. *Modelos de Calidad Web. Clasificación de Métricas.* TESIS DE MAESTRIA, UNIVERIDAD NACIONAL DE EDUCACIÓN A DISTANCIA: 308
5. GUTIÉRREZ, J. J.; ESCALONA, M. J. *Estudio comparativo de propuestas para la generación de casos de prueba a partir de requisitos funcionales.*  
<http://www.lsi.us.es/docs/informes/LSI-2005-01.pdf>.
6. SCALONE, L. F. *Estudio comparativo de los modelos y estándares de calidad del software.*
7. COMPUWARE. *Las empresas europeas asumen ya la calidad del software pero no saben aplicar metodologías consistentes.*  
<http://www.noticias.com/notaprensa/28-04-2006/bdi-comunicacion/empresas-europeas-asumen-ya-calidad-software-pero-no-saben-aplicar-metodologias-consistentes-c29.html>.
8. ESTRADA, A. F., S. A. CÁRDENAS, et al “*Calidad de Software y la empresa, enseñanza de un tema imprescindible para el ingeniero Informático.*”  
<http://www.somece.org.mx/memorias/2000/docs/123.DOC>

9. LOVELLE, J. M. C. "Calidad del software."

[http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad\\_software.PDF](http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF)

10. BILELLO, M. A. *El valor de cada uno.*

[http://bloggers.com.ar.elsevier.com/system/noticia\\_detalle.php?id\\_prod=607.](http://bloggers.com.ar.elsevier.com/system/noticia_detalle.php?id_prod=607)

11. REYES., I. B. y TORRES., I. N. *Las pruebas de software, su aplicación al Config. CASE.*

12. IEEE. *Metrics.*

13. LETELIER, P. "Pruebas de software."

<https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Pruebas%20del%20SW.ppt>

14. PÉREZ P. G. *Principios básicos del desarrollo seguro.*

[http://germinus.com/sala\\_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf](http://germinus.com/sala_prensa/articulos/ppos%20basicos%20desarrollo%20seguro.pdf)

15. COLLADO, M. "Pruebas de software. Técnicas de prueba del software. Estrategia de prueba del software." <http://lml.ls.fi.upm.es/ftp/ed2/0203/Apuntes/pruebas.ppt>

16. UNIDOS, A. P. L. A. Y. M. L. E. *Centro para la Seguridad Alimentaria y la Nutrición Aplicada.*

Última actualización: 24 de enero de 2005. <http://www.cfsan.fda.gov/~mow/scfsan4.html>.

17. U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES. *U.S. Food and Drug Administration* <http://www.fda.gov/oc/spanish/>

18. HERNÁNDEZ, L.A y COELO, G.S. *El paradigma cuantitativo de la investigación científica.*

19. BLANCO .R, Díaz .E, Tuya J. *Generación automática de casos de prueba mediante búsqueda dispersa.*
20. GÓMEZ, L y HOYOS, P. *Eficiencia de la caracterización de técnicas de pruebas, un contexto real de aplicación: PARQUESOFT.*
21. ROCA, M. J. *Pruebas de Integración de Productos: Un enfoque práctico.*
22. GUZMÁN, O. H. *Aplicación práctica del diseño de pruebas de software a nivel de programación.*
23. MAÑAS, J. A. *Pruebas de Programas.*
24. FERNÁNDEZ, L.S y Alarcón, M.I. *Necesidades de medición en la gestión y el aseguramiento de calidad del software.*
25. D'ONOFRIO, D.L. *Probando software y números de versión.*
- 26 US. Food and drugs administration  
<http://www.fda.gov/oashi/clinicaltrials/SpanishClinTri.html#1>
- 27 LETELIER, P. Departamento Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. *Pruebas Del Software.*
28. MSc. VIADA, C. E. Dpto. Inmunología Clínica. Centro de Inmunología Molecular. *Guía para sistemas computarizados usados en ensayos.*
29. COLLADO, M. *Técnicas de prueba del software. Estrategias de prueba del software.*
30. CHILLAREGE. R. *Software Testing best practices.* Center for software Engineering IBM Research.



31. RUIZ, E. G. *Fases de pruebas*.
32. *Centro nacional de ensayos clínicos*. <http://www.cencec.sld.cu/inicio.htm>.
33. *Centro de inmunología molecular (CIM)* <http://www.cim.sld.cu/>.
34. GUTIERREZ, J. J, ESCALONA, M.J, MEJÍAS, M. y REINA, A.M. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. Escuela Superior de Ingeniería Informática *Modelos de pruebas para pruebas del sistema*.  
<http://www.lsi.us.es/~javierj/publications/MDA14.pdf>.
35. GRIMÁN, A.C, PÉREZ, M, MENDOZA, L. E. *Estrategia de Pruebas para Software OO que garantiza Requerimientos No Funcionales*.  
[http://www.lisi.usb.ve/publicaciones/02%20calidad%20sistemica/calidad\\_09.pdf](http://www.lisi.usb.ve/publicaciones/02%20calidad%20sistemica/calidad_09.pdf)
36. ESTRADA, A. F, FERNÁNDEZ, H. Y. "Calidad de Software y la empresa, enseñanza de un tema imprescindible para el ingeniero."  
<http://www.somece.org.mx/memorias/2000/docs/123.DOC>
37. CORTÉS, O. H. G. "Aplicación práctica del diseño de pruebas de software a nivel de programación."  
[http://www.willydev.net/descargas/oguzman-diseno\\_pruebas.pdf](http://www.willydev.net/descargas/oguzman-diseno_pruebas.pdf)
38. PÚBLICAS, M. d. A. "Implantación y Aceptación del Sistema."  
<http://www.csi.map.es/csi/metrica3/iasproc.pdf>
39. GUTIÉRREZ, J. J., M. J. ESCALONA. "Aplicando Técnicas de Testing en Sistemas para la Difusión Patrimonial."  
[http://www.turismo.uma.es/turitec/turitec2004/docs/actas\\_turitec\\_pdf/15.pdf](http://www.turismo.uma.es/turitec/turitec2004/docs/actas_turitec_pdf/15.pdf)

40. JOSÉ, F. M. and G. J. PABLO. "*Sistemas de Programas.*"  
<http://www ldc.usb.ve/~teruel/ci3711/test1/index.html#3>.

## **GLOSARIO DE TÉRMINOS**

**Calidad:** Calidad de software. Satisfacción de las necesidades de los usuarios.

**Defecto:** Cualquier requerimiento, elemento de diseño o de implementación que si no es cambiado, causará un diseño, implementación, prueba, uso, o mantenimiento inapropiado del producto.

**Interfaz:** Una colección de operaciones que se usan para especificar el servicio de una clase o de un componente. Un juego nombrado de operaciones que caracterizan la conducta de un elemento. La Interfaz hombre-máquina es un canal comunicativo entre el usuario y el ordenador.

**Usuario:** Persona que utiliza normalmente el *software*.

**Prueba:** Prueba de software. Ejecución de un sistema bajo condiciones específicas, se observan y se analizan los resultados realizándose una evaluación de los mismos.

**FDA:** Compañía Americana dedicada a la administración de alimentos y medicamentos para personas y animales.

**Funcionalidad:** Funcionalidad de software. Conjunto de operaciones que realiza el software.

**Cobertura:** Medida porcentual de cuánto código se ha probado.

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

**Proceso:** Secuencia de actividades invocadas para producir un producto de software.