Universidad de las Ciencias Informáticas
Facultad 6

Diseño y aplicación de pruebas al producto Registro Cubano de Discapacitados.

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

Autores: **Lesdey Saavedra López**

Yohandris Pupo Blez

Tutores: **Ing. Reynier García Vistorte**
Lic. Yosdenis Urrutia Badillo

Ciudad de la Habana, julio de 2007

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 _____.

Lesdey Saavedra López
Yohandris Pupo Blez
_____

_____

Ing. Reynier García Vistorte
Lic. Yosdenis Urrutia Badillo
_____

_____

Opinión del tutor del trabajo de diploma.

Título: Diseño y aplicación de pruebas al producto “Registro Cubano de Discapacitados”.

Autores: Lesdey Saavedra López
Yohandris Pupo Blez

El tutor del presente Trabajo de Diploma considera que durante su ejecución los estudiantes mostraron las cualidades que a continuación se detallan:
Manifestaron una actitud disciplinada, independiente y responsable, mucha dedicación, gestionando información y enfocados a la mejora continua de su trabajo**.**
Su trabajo es perfectamente aplicable al entorno de la UCI, los resultados poseen un criterio científico adecuado y traerán beneficios de funcionamiento a los productos que la apliquen y con ello su calidad frente al cliente.
Por todo lo anteriormente expresado considero que los estudiantes están aptos para ejercer como Ingenieros en Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de 5-Excelente. Además, considero que los resultados poseen valor para ser publicados.

_____

Ing. Reynier García Vistorte

Julio 2007.

AGRADECIMIENTOS

Agradecemos a los profesores Yosdenis Urrutia, Alieski Sarmiento, Yanet Villanueva y Maypher Román por la ayuda brindada en la revisión minuciosa de nuestro trabajo y las sugerencias siempre oportunas que nos brindaron; por la incondicionalidad y el interés mostrado, muchísimas gracias.

Agradecemos infinitamente a la Revolución Cubana por la calidad de su sistema de educación; gracias a ella, el alto valor humano y profesional alcanzado en estos años, nos hará transitar victoriosos en la construcción de un futuro mejor.

A nuestro Comandante en jefe, Fidel Castro, por la creación de tan importante proyecto; por la confianza que siempre ha tenido en que su tropa de futuro hará, por el desarrollo de la sociedad más justa de estos tiempos, todo cuanto sea necesario; constituya entonces este trabajo nuestro primer aporte a tan noble empeño, un regalo y el eterno compromiso de que no le fallaremos.

DEDICATORIA

A mi madrina, a Martica y a Brenda por el apoyo constante.

A mi abuela Balbina, cuyo cariño y ternura me han dado fuerzas para no fallarle.

A ti Juli, mi Cesar victorioso, para que siempre luches por alcanzar grandes cosas.

A mis padres, María y Fernando, por vivir siempre orgullosos de mí, por la guía y el apoyo…

Yohandris.

A mis abuelos Eralio y Berta que han sabido guiarme por el camino de la verdad y del estudio.

A mi papá por siempre entenderme y apoyarme en los momentos difíciles.

A mi tía Dalila que siempre ha estado pendiente de mis resultados, dándome ánimo para continuar.

A mi novio Roicxy, por estar siempre ahí, cuando lo necesité.

A mis amistades y en especial a Getma que siempre me ayudó.

En fin a toda mi familia, que siempre me han apoyado, y han confiado en que podía lograrlo.

Lesdey.

Hombre es algo más que ser torpemente vivo: es entender una misión, ennoblecerla y cumplirla.

José Martí.

RESUMEN

La producción de software en la actualidad es tema de interés para especialistas, ingenieros, y comercializadores de estos productos. El desarrollo alcanzado por la sociedad, muestra una tendencia de los clientes hacia la necesidad creciente de mejores productos y en un plazo de tiempo cada vez menor, de ahí los estudios constantes que se realizan para determinar cómo obtener un software de calidad y cómo evaluarla. Las pruebas de software son la actividad más comúnmente realizada para el control de calidad en los proyectos de desarrollo o mantenimiento de aplicaciones y sistemas. La Universidad de las Ciencias Informáticas (UCI) desde la creación de los grupos de calidad, viene dando pasos de avance en el tema.

Dentro del cúmulo de proyectos acogidos por la UCI se desarrolla el Registro Cubano de Discapacitados (RECUDIS) para el Centro Nacional de Genética Médica (CNGM). A dicha institución se le otorgó la importante misión de realizar un estudio psicosocial de todas las personas con discapacidades físicas en el país, de ahí la necesidad de crear una herramienta automatizada para facilitar dicho estudio y que le permita a los especialistas del CNGM controlar el proceso de gestión de la información en el centro y específicamente en el Registro Cubano de Discapacitados.

Este trabajo se centra en el diseño y aplicación de pruebas de software a los módulos Nuevo, Modificar y Reportes, los cuales han sido implementados en la primera versión del RECUDIS, para detectar y poder eliminar los errores que presentan, lograr una aplicación con la menor cantidad de fallas y por consiguiente la mejora en su funcionamiento. Para alcanzar este objetivo se aplicaron pruebas de código (estructurales o de caja blanca) y de interfaz y requerimientos (funcionales o de caja negra) a cada módulo implementado. Se diseñaron para cada tipo de prueba casos de prueba que luego fueron aplicados al sistema, comparándose los resultados obtenidos con los esperados a brindar por el software, concluyendo con un análisis valorativo de los mismos.

TABLA DE CONTENIDOS

Introducción

La calidad del software es punto de atención en empresas y organizaciones, dado que estos productos se han convertido en activos que determinan en gran medida el nivel de funcionalidad de una organización.

"La industria del software en el mundo se desarrolla a un ritmo vertiginoso, aunque la producción sigue siendo aún baja y los costos muy elevados. Esta situación se debe, en la mayoría de los casos, a la no aplicación de técnicas de Ingeniería y Gestión de Software y la no definición de roles y procesos adecuados en el desarrollo de software". (Febles, 2004)
"En la industria de software en Cuba existen problemas con la productividad de los trabajadores, los tiempos de entrega de los productos y de las documentaciones, la calidad de las pruebas y la falta de comunicación efectiva entre los usuarios, desarrolladores, administradores, clientes e investigadores". (Febles, 2001)

Las causas fundamentales de estos problemas están dadas porque no hay disciplina en el cumplimiento de los calendarios del proceso de desarrollo; no se realizan seguimientos de la evolución de los procesos, ni de la productividad de los desarrolladores; no se estimula a crear un espíritu de equipo; los softwares exceden en el tiempo y el costo planificados, existen problemas en la administración de los proyectos, la cual debe contar con planes claros, precisos y que sirvan para cuantificar y medir el estado en cada momento del desarrollo del proyecto. (Pérez, 2002)

En Cuba, la Universidad de las Ciencias Informáticas (UCI) es pionera en la creación de un grupo central dedicado al diseño y aplicación de pruebas de software para validar la calidad de los productos desarrollados. Luego se extienden a grupos por cada una de sus facultades, que beneficiaría en temas de calidad los proyectos productivos nacionales, así como los compromisos con el extranjero. Es conocido que la UCI jugará un papel 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, y para ello se realizan algunos proyectos con el objetivo de mejorar las prestaciones y servicios de

instituciones en las esferas de la educación y la salud fundamentalmente, de ahí que se busquen marcos de trabajo para aumentar la productividad, la calidad, disminuir los costos, mejorar los procesos de software y perfeccionar el diseño de las pruebas realizadas a éstos.

Las pruebas de software son la actividad más comúnmente realizada para el control de calidad en los proyectos de desarrollo o mantenimiento de aplicaciones y sistemas. El aseguramiento de calidad incluye otras técnicas como, inspecciones y revisiones (automatizadas o no) de modelos y documentos no ejecutables de las primeras fases de desarrollo. Las pruebas de software se definen como *"una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran, y se realiza una evaluación de algún aspecto".* (IEEE, 1990)

El desarrollo de una estrategia de prueba de software adecuada incluye la integración de las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. Esta debe incorporar: la planificación de las pruebas, el diseño de los casos de prueba, la ejecución de las pruebas, la recolección y evaluación de resultados.

Dentro del cúmulo de proyectos acogidos por la UCI se desarrolla el Registro Cubano de Discapacitados, en lo adelante RECUDIS, para el Centro Nacional de Genética Médica (CNGM), institución científica que tiene la misión de llevar a cabo acciones asistenciales, docentes y de investigación en el campo de los problemas de salud de carácter genético, encaminadas a elevar la calidad de vida y el bienestar de la población. Este centro crece continuamente, no solo en tamaño, sino en programas e investigaciones. A dicha institución se le otorgó la importante tarea de realizar un estudio psicosocial de todas las personas con discapacidades físicas en el país, de ahí la necesidad de crear una herramienta automatizada para facilitar dicho estudio y que le permita a los especialistas del CNGM controlar el proceso de gestión de la información en el centro y específicamente en el Registro Cubano de Discapacitados. Actualmente esta aplicación se encuentra en la fase terminal de implementación en su primera versión, de ahí la **situación problemática** que supone el desarrollo de este sistema, para el cual, una vez finalizadas las etapas de análisis, diseño e implementación, no existe un diseño de pruebas que validen la calidad del producto, el cumplimiento de los requisitos y especificaciones para ser

entregado al cliente. Por lo que el **problema a resolver** consiste en: ¿Cómo validar la calidad del producto RECUDIS?

Para validar la calidad de este producto se propone realizar un adecuado diseño de pruebas, que una vez aplicado permita conocer los errores sobre los cuales trabajar para lograr un sistema más estable y eficiente. Por tanto el **objeto de estudio** de la investigación es el diseño y aplicación de pruebas a un producto software, enmarcando el **campo de acción** en la calidad del producto RECUDIS mediante un diseño y aplicación de pruebas para el caso específico de este sistema.

Basado en esto, el **objetivo general** de este trabajo es:

Diseñar y aplicar pruebas de software al producto "RECUDIS", que validen la calidad del sistema.

Del cual se derivan los siguientes **objetivos específicos:**

- Fundamentar los temas relacionados con la calidad de software haciendo énfasis en las pruebas realizadas a estos.
- Diseñar las pruebas necesarias para el RECUDIS.
- Validar, mediante la aplicación de pruebas, el diseño propuesto para el RECUDIS.
- Documentar los resultados obtenidos de la aplicación de las pruebas.

Para cumplir los objetivos propuestos se realizaron las siguientes **tareas**:

- Estudio de aspectos importantes vinculados a la calidad del software.
- Investigación sobre las pruebas de calidad de software más aplicadas a nivel mundial.
- Estudio detallado de las pruebas que se aplican en la infraestructura productiva de la universidad.
- Diseño de las pruebas de caja blanca y caja negra para los módulos seleccionados.
- Aplicación de las pruebas al RECUDIS.

El presente trabajo está estructurado en tres capítulos:

Capítulo 1. **Fundamentación teórica.**

En este capítulo se exponen los modelos y estándares existentes para la administración de la calidad en un proyecto software. Se presentan además conceptos, métodos y procedimientos de calidad de software resultado del análisis bibliográfico así como un análisis crítico de la bibliografía consultada. Se

hace referencia a las metodologías de desarrollo de software y las pruebas que se realizan para validar su calidad.

Capítulo 2. Diseño y aplicación de las pruebas al Registro Cubano de Discapacitados.

En este capítulo se describe de modo general el producto, se elabora un plan de pruebas, se diseñan y aplican las pruebas estructurales y funcionales acordes a sus características.

Capítulo 3. Resultados.

En este capítulo se presentan los resultados de la aplicación de las pruebas, valoración de los mismos y planteamiento del resultado final.

Además se relacionan una serie de anexos, que complementan la información expuesta en cada uno de los capítulos.

Capítulo 1. Fundamentación teórica.

1.1 Introducción.

El concepto de calidad ha adquirido un carácter multidimensional, debido a que los diferentes autores, conocidos estudiosos del tema, lo han enfocado desde puntos de vistas diferentes: Deming, como el grado predecible de uniformidad y conformidad a un bajo costo que se ajuste a las necesidades del mercado; Crosby, como cumplir con los requisitos; Feigenbaum, como el conjunto total de las características del producto de marketing, ingeniería, fabricación y mantenimiento a través del cual el producto en uso va a satisfacer las expectativas del cliente; y Juran, como la idoneidad o aptitud para el uso.

En todas estas definiciones se valoran elementos comunes como la satisfacción de necesidades o perspectivas del cliente y la existencia de rasgos o requisitos para satisfacerlas. La calidad en la Ingeniería de software no escapa de todas estas características y presenta quizás muchos más obstáculos que hacen que su control y garantía sea un tema profundo de estudio y perfeccionamiento debido a los avances y cada vez más complejos sistemas informáticos desarrollados en el mundo.

1.2 Calidad de Software. Conceptos y Definiciones.

Según Pressman, la calidad de software es *"la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente".* (Pressman, 1998)

A partir de esta definición es importante señalar que no sólo afecta la calidad el incumplimiento de los requisitos del cliente y los explícitamente definidos por la ingeniería de software, sino que los requisitos implícitos también deben ser considerados, pocas veces el cliente está en condiciones reales de explicitar todo lo que se puede esperar del producto, muchas veces por desconocimiento y otras por la asunción tácita de muchas funcionalidades. De ahí que estos requerimientos se incluyan en el concepto de que la calidad de software es *"El conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas". ISO 8402 (UNE 66-001-92).* (Granja, 2007)

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

La calidad por tanto se convierte en una actitud para poder cumplir siempre con los requisitos exigidos por el cliente. Los requisitos del software son la base de las medidas de calidad de ahí que la falta de concordancia entre estos y la omisión de algunos requisitos implícitos o expectativas que a menudo no se mencionan o se mencionan de forma incompleta resulte en una falta de calidad al igual que si no se sigue ninguna metodología, dada la importancia que presentan los estándares, pues definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software.

Un concepto que está muy vinculado a la calidad de software es el de aseguramiento de calidad del software (Software Quality Assurance (SQA)): *"conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) satisfará los requisitos dados de calidad".* (Granja, 2007)

El aseguramiento de calidad del software se diseña para cada aplicación antes de comenzar a desarrollarla y no después, muchos autores utilizan el término garantía en vez de aseguramiento y quizás el primero no se ajuste al producto software ya que puede confundir con garantía de productos, el segundo por su parte pretende dar confianza en que el producto tiene calidad. El aseguramiento de calidad está presente en los siguientes aspectos:

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

Incluye además las siguientes actividades:

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

La gestión de configuración del software (Software Quality Management) es el *"conjunto de actividades de la función general de la dirección que determina la calidad, los objetivos y las responsabilidades y se implanta por medios tales como la planificación de la calidad, el control de la calidad, el aseguramiento de la calidad y la mejora de la calidad, en el marco del sistema de calidad".* (Granja, 2007)

La gestión de la calidad se puede entender como el conjunto de actividades y medios necesarios para definir e implementar un sistema de calidad, por una parte y responsabilizarse de su control, aseguramiento y mejora continua, por otra. El control está dirigido al cumplimiento de requisitos, el aseguramiento a inspirar confianza en que se cumplirá el requisito pertinente y el mejoramiento al aumento de su eficiencia y eficacia. La gestión de la calidad se aplica normalmente a nivel de empresa aunque también puede haber una gestión de calidad dentro de la gestión de cada proyecto de ahí que la política de calidad en cada caso sean directrices y objetivos generales de una organización relativos a la calidad, tal como se expresan formalmente por la alta dirección. Las actividades realizadas para evaluar la calidad de los productos desarrollados se encuentran dentro del control de calidad del software (Software Quality Control) *"técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en dos objetivos fundamentales: mantener bajo control un proceso y eliminar las causas de los defectos en las diferentes fases del ciclo de vida".* (Granja, 2007)

Existen algunos factores que determinan la calidad del software, los cuales se clasifican en tres grupos:

Operación del producto: características operativas.

- Corrección: Hasta dónde satisface un programa su especificación y logra los objetivos del cliente.
- Fiabilidad: Hasta dónde se puede esperar que un programa lleve a cabo su función con la exactitud requerida.
- Eficiencia: La cantidad de recursos informáticos y de código necesarios para que un programa realice su función.

- Integridad: Hasta dónde se puede controlar el acceso al software o a los datos por personas no autorizadas.
- Usabilidad (facilidad de manejo): El esfuerzo necesario para aprender a operar los datos de entrada e interpretar las salidas de un programa.

Revisión del producto: capacidad para soportar cambios.

- Facilidad de mantenimiento: El esfuerzo necesario para localizar y arreglar un error en un programa.
- Flexibilidad: El esfuerzo necesario para modificar un programa operativo.
- Facilidad de prueba: El esfuerzo necesario para probar un programa y asegurarse que realice la función pretendida.

Transición del producto: adaptabilidad a nuevos entornos.

- Portabilidad: El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
- Reusabilidad: Grado en que partes de una aplicación pueden utilizarse en otras aplicaciones.
- Interoperabilidad: El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos.

1.3 Modelos de calidad.

El desarrollo mundial de las TICs, ha traído consigo la necesidad de controlar la confiabilidad, funcionalidad, eficiencia, usabilidad, portabilidad y mantenibilidad de un software, siendo todas estas características esenciales de la calidad. A la hora de implantar modelos eficientes resulta un poco complicado decidir ante el abanico de modelos de calidad, de procesos y de técnicas de trabajo desplegado en los últimos años. Estos modelos tienen que estar estrechamente relacionados con un entorno de producción sistémico, en cuyo diseño global los componentes tienen que encajar y funcionar armónicamente, alineados con las características, cultura y estrategia de la organización. Para todas las industrias hay marcos de trabajo compuestos por normativas y estándares más o menos estables y

consensuados, que en forma de modelos ayudan a mejorar o evaluar la calidad de sus sistemas de producción.

Al mismo tiempo cada ingeniería tiene delimitadas sus áreas de conocimiento, y reguladas las técnicas de trabajo para ofrecer las garantías necesarias en la construcción de sus respectivos artefactos. Así por ejemplo, una empresa de arquitectura, o de ingeniería aero-espacial, naval o nuclear tiene estándares y conocimientos estables, que si aplica sistemáticamente aportan garantías contrastadas a la robustez de las estructuras de sus edificios o de las naves que construye.
Estos conocimientos, que sirven de referencia y ayuda a los entornos de desarrollo, provienen de dos áreas:

- Currículo de técnicas y conocimientos de las respectivas ingenierías.
- Modelos, normas y estándares de calidad y procesos aplicables a cada industria.

Una diferencia importante entre la industria del software y otras industrias de ingeniería es la falta de consenso o inmadurez en los dos puntos anteriores, que deja a las empresas de este sector en un cierto grado de orfandad a la hora de buscar ayuda en modelos, técnicas o patrones de referencia.
El problema en este momento no es la falta de estándares, modelos o técnicas, sino la abundancia de ellos, por lo que resulta necesario un estudio y conocimiento profundo de las características, ventajas y desventajas de cada uno, teniendo en cuenta las características de la organización en la cual se vayan a implantar.

La necesidad de utilizar normas de calidad se hace presente a mediados del siglo XIX cuando comienza a desarrollarse la producción en masa, evoluciona rápidamente a principios del siglo y se destacan los siguientes hitos:

1900, Inspección como actividad.

1930, Muestreo estadístico.

1950, Prácticas de aseguramiento de calidad en empresas.

1970, Prácticas de aseguramiento de calidad a nivel nacional.

1979, Normas para el aseguramiento de la calidad, BS 5750.

1987, Basadas en la norma BS 5750 se editan las normas ISO serie 9000.

1994, Se realiza una revisión de las normas base.

2000, Se realiza la última revisión de las normas base.

La calidad no ha sido controlada siempre de la forma en que se suele hacer hoy, ya que es un proceso en constante cambio y perfeccionamiento. Hasta la década de 1940 solo se hacía una inspección y detección de errores de forma artesanal mediante un control individual de cada tarea.

Los procesos de fabricación de los años 50 hicieron necesaria la normalización de los procesos de producción para garantizar la consistencia de los resultados sobre unos parámetros o requisitos previamente determinados. A partir de entonces y hasta los años 80 se realizaba un control estadístico de calidad, debido a que existía un mercado poco competitivo y el precio de venta era fijado por el fabricante en función de los costos; se trataba de impedir que el producto defectuoso llegara al cliente así como conseguir uniformidad en el servicio. En estos años controlar la calidad equivalía a resolver un problema y los controles en los departamentos de producción se hacían utilizando técnicas estadísticas. En lo adelante la garantía de calidad es lo que impera, ya existe un mercado competitivo y de oferta donde el precio de venta es fijado por el mercado, surgen las primeras planificaciones y medidas de la calidad, así como algunos modelos, afectando ya no solo al departamento de producción sino a todos.

Se destacan algunos hitos que demuestran el interés creciente tales como:
1980. Interés por la calidad en los EE.UU, Total Quality Management (TQM).
1987. Premio Malcom Baldrige Quality Award.
1987. ISO 9000 a partir de las normas británicas.
1992. Premio Europeo a la calidad, de la European Foundation for Quality Management (EFQM).

Desde mediados de los años 70 se fueron desarrollando los modelos más reconocidos en la industria, siendo hoy en día la base conceptual más importante en las evaluaciones de productos software. En el año 1977 surge el modelo McCall, este se organiza a través de tres características de la calidad: la primera, los factores que describen la vista externa del software, lo visto por el usuario; la segunda, criterios que describen la vista interna del software, según lo visto por el revelador; la tercera, se definen métricas y se utilizan para proporcionar una escala y un método para la medida.

Según McCall los factores que afectan la calidad se pueden categorizar en: factores que se pueden medir directamente, como por ejemplo los defectos por punto de función; factores que se pueden medir sólo indirectamente, como por ejemplo la facilidad de uso o mantenimiento. En todos los casos debe aparecer la medición. Debe ser posible comparar el software (documentos, programas, datos) con una referencia y llegar a una conclusión sobre la calidad. (Granja, 2007)

Sobre la base de los errores y de las necesidades más actuales, surgió un nuevo modelo, creado por el comité técnico de la ISO/IEC, el modelo ISO 9126, siendo una norma internacional, que abarca todos los temas relacionados con la calidad de un software. Es un estándar internacional dividido en cuatro posiciones: Modelo de la calidad, métrica externa, métrica interna y métrica de funcionalidad. (Hoyos, y otros, 2002).

El modelo de la calidad clasifica la calidad del software como un sistema estructurado en características como la *funcionalidad*, que se refiere a las funciones de un sistema que deben satisfacer necesidades indicadas o implicadas. La *confiabilidad* que se refiere a la capacidad del software para mantener su nivel de funcionamiento bajo condiciones indicadas en un período de tiempo, es decir la madurez del software, la recuperabilidad y la tolerancia de avería que debe presentar. La *utilidad del sistema*, referida al esfuerzo que se debe hacer para el uso del mismo. La *eficacia*, referida a la relación entre el nivel de funcionamiento del software y la cantidad de recursos utilizados, teniendo en cuenta el comportamiento del tiempo y el comportamiento de los recursos. La *portabilidad*, referida a la capacidad que debe tener el software para ser transferido de un ambiente a otro, es decir que debe ser adaptable.

Las métricas internas son las que no confían en la ejecución del software, las métricas externas son aplicables al software corriente y las métricas de funcionalidad de la calidad solo están disponibles cuando el producto final se utiliza en condiciones verdaderas. En fin, la calidad interna, determina la calidad externa y la externa, determina la calidad funcionando.

Todo lo anterior derivó en lo que actualmente es la gestión de calidad, *"un conjunto de actividades de la función general de la dirección, que determina la calidad, los objetivos y las responsabilidades y se implanta por medios tales como la planificación de la calidad, el control de la calidad, el aseguramiento (garantía) de la calidad y la mejora de la calidad, en el marco del sistema de calidad".* (Añasco, 2007)

La gestión de la calidad se aplica normalmente a nivel de empresa aunque también puede haber una gestión de calidad dentro de la gestión de cada proyecto. Constituye un impacto estratégico en tanto oportunidad de ventaja competitiva. Posibilita la planificación, fijación de objetivos, coordinación, formación y adaptación de toda la organización. Afecta a la sociedad en general, directivos, trabajadores y clientes convirtiéndose en una filosofía, una estrategia, una cultura, un estilo de gerencia para la empresa actual. La gestión de la calidad se apoya en la norma ISO 9001:2000, que ha sido adoptada como modelo a seguir para obtener la certificación de calidad. Esta especifica los requisitos para un sistema de gestión de la calidad que pueden utilizarse para su aplicación interna por las organizaciones, para certificación o con fines contractuales. Se centra en la eficacia de la gestión de la calidad para dar cumplimiento a los requisitos del cliente.

Esta norma internacional permite a una organización integrar o alinear su propio sistema de gestión de la calidad con requisitos de sistemas de gestión relacionados. Es la única norma de la familia ISO 9001 certificable, tiene una nueva estructura basada en procesos, y consta de los siguientes puntos principales:

- Responsabilidad de la Dirección
- Gestión de recursos
- Realización del Producto
- Medición, análisis y mejora

Esta nueva revisión ISO:

- Se basa en el famoso “Círculo de Deming”: PDCA - acrónimo de Plan, Do, Check, Act (Planificar, Hacer, Verificar, Actuar).
- Está estructurada en cuatro grandes bloques, completamente lógicos, y esto significa que con el modelo de sistema de gestión de calidad basado en ISO se puede desarrollar en su seno cualquier actividad. La ISO 9001:2000 se presenta con una estructura válida para diseñar e implantar cualquier sistema de gestión, no solo de calidad, e incluso, para integrar diferentes sistemas. (Añasco, 2007)

Otros modelos importantes en la certificación de los productos software, son SPICE y CMMI (Capability Maturity Model Integration):

CMMI actualmente se usa en la mayoría de las empresas y organizaciones productoras de software en el mundo. En la actualidad hay varios modelos CMMI, en función de las áreas que integran:

- CMMI-SE/SW/IPPD/SS (Systems Engineering, Software Engineering, Integrated Product and Process Development, Supplier Sourcing).
- CMMI-SE/SW/IPPD (Systems Engineering, Software Engineering, Integrated Product and Process Development).
- CMMI-SE/SE (Systems Engineering, Software Engineering)

Además de la integración de varias disciplinas, los modelos CMMI tienen dos utilidades. Pueden servir tanto como guía para la mejora en una organización, o como criterio para evaluar su nivel; además introduce una segunda dimensión, también válida para guiar las actividades de mejora y para evaluar a las organizaciones: la capacidad de los procesos.

CMMI establece 6 niveles para determinar la capacidad de un proceso:

- Nivel 0- INCOMPLETO. El proceso no se realiza.
- Nivel 1- REALIZADO. Se lleva a cabo el proceso, consiguiendo transformar elementos de entrada identificados, en productos de salida.
- Nivel 2- GESTIONADO. El proceso se ejecuta siempre de la misma manera, de una forma gestionada.
- Nivel 3- DEFINIDO. El proceso está definido en la organización y se ejecuta siempre.
- Nivel 4-CUANTIFICADAMENTE GESTIONADO. La ejecución del proceso tiene institucionalizado en la organización un sistema de medición objetivo y cuantificable de su capacidad.
- Nivel 5.- OPTIMIZADO. El proceso, que se ejecuta siempre, está definido en la organización, se mide y está integrado en un plan, también institucionalizado, de mejora continua basada en las mediciones de los procesos.

De esta forma los modelos CMMI presentan 2 versiones:

- Versión escalonada: Guía a la organización sobre las áreas de procesos que debe ir abordando, las prácticas que debe implantar y los objetivos que debe alcanzar para conseguir los sucesivos niveles de madurez.
- Versión continua: Permite cierta libertad en la organización sobre las áreas de proceso que desea mejorar, le orienta para ir elevando su nivel de capacidad.

Como se puede apreciar, la necesidad de entregar software con calidad, de disminuir costos y ganar prestigio ha propiciado que cada vez tomen mayor auge las normas usadas para el control de la calidad

del software. Todo esto ocurre a nivel internacional, donde el mercado de software es cada vez más exigente. Las empresas se ven obligadas a implantar muchos de estos modelos en busca de confianza y garantía para con el cliente. Podemos destacar en América del Sur algunos países en los cuales existen empresas de certificación de productos software, en orden descendente, Brasil cuenta con 85 empresas, seguida por Colombia con 63, una cifra menor, pero significativa en el desarrollo de la calidad del software. Le siguen Chile con 10, Argentina con 4, Venezuela y Ecuador, con 2 empresas, y Uruguay con una. (Febles, 2006)

En el ámbito nacional, como resultado de los programas de informatización de la sociedad y las aspiraciones de insertarse progresivamente en el mercado internacional de software, se hacen esfuerzos y estudios para aplicar estas normas y modelos a nuestras instituciones. La UCI realiza importantes estudios sobre este tema, aunque la mayoría de los productos son desarrollados sin tener en cuenta estos modelos. Los proyectos se desarrollan siguiendo los pasos y técnicas de la metodología RUP, como es el caso del RECUDIS. De ahí que el control de la calidad en estos proyectos se lleve a cabo mediante el rol de Ingeniero de prueba que propone dicha metodología.

1.4 Metodologías de desarrollo de software.

1.4.1 Xtreme Programming (XP).

Extreme Programming (XP) es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada para proyectos de corto plazo, pequeños equipos y cuyo plazo de entrega sea en un tiempo muy breve. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, siendo este uno de los requisitos para llegar al éxito del proyecto. Es una metodología basada en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de forma tal que adelantándose al futuro, se realicen pruebas de fallas que pueden ocurrir. Es como obtener con antelación posibles errores.

- Re fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- Programación en pares: es una particularidad de esta metodología, propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

1.4.2 Microsoft Solution Framework (MSF).

La metodología Microsoft Solution Framework (MSF) es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de procesos y de equipos, dejando en un segundo plano las elecciones tecnológicas.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

Es una metodología adaptable, escalable, ya que organiza equipos pequeños de entre 3 o 4 personas, así como proyectos que requieren 50 personas o más; flexible, utilizada en el ambiente de desarrollo de cualquier cliente; tecnología agnóstica, porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.4.3 Rational Unified Process (RUP).

La metodología RUP llamada así por sus siglas en inglés Rational Unified Process, posee tres características esenciales:

- Centrado en la arquitectura
- Dirigido por casos de uso
- Iterativo e incremental

Es la metodología utilizada en el desarrollo del RECUDIS, de ahí que las pruebas necesarias para validar la calidad del sistema se realizaran siguiendo las actividades del rol de ingeniero de prueba. Esta metodología divide en 4 fases el desarrollo del software y propone que en cada una de ellas las pruebas se comporten de la siguiente forma:

- Fase de Inicio: El desarrollo del prototipo exploratorio de demostración; no requiere la elaboración de pruebas.

- Fase de Elaboración: Probar los componentes ejecutables que se han implementado y que deben corresponderse con la arquitectura básica de la aplicación.
- Fase de Construcción: Desarrollar los casos de prueba y procedimientos de prueba para hacerlos.
- Fase de Transición: El producto en su entorno de operación por lo que es probado por usuarios reales.

1.4.3.1 Relación con otras Disciplinas.

- La disciplina de **Requerimientos**, captura los requerimientos para el producto software, los cuales son una de las entradas principales para identificar que pruebas ejecutar.
- La Disciplina de **Análisis y Diseño**, determina el diseño apropiado para el software, el cual es otra de las entradas principales para identificar que pruebas ejecutar.
- La Disciplina de **Implementación** produce versiones operacionales del sistema (builds) que son validados por pruebas, dentro de una iteración, múltiples builds serán probados (1 por ciclo de prueba).
- La disciplina de **Despliegue** entrega el producto de software completo al usuario final. Antes el software es validado por Prueba, aunque pruebas de aceptación y pruebas betas son ejecutadas como parte del despliegue.
- La **Gestión de Proyecto** planifica el proyecto y las iteraciones, el Plan de Iteración descrito, artefacto que es una importante entrada usada cuando se va a definir la misión de evaluación para la prueba.
- The **Configuration & Change Management Discipline:** controla los cambios dentro del proyecto. La Prueba verifica que cada cambio ha sido completado apropiadamente.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

El ciclo de vida que se desarrolla por cada iteración, es llevado a cabo bajo dos disciplinas:
Disciplina de Desarrollo

- Ingeniería de Negocios: Entendiendo las necesidades del negocio.
- Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.

- Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

Disciplina de Soporte

- Configuración y administración del cambio: Guardando todas las versiones del proyecto.
- Administrando el proyecto: Administrando horarios y recursos.
- Ambiente: Administrando el ambiente de desarrollo.
- Distribución: Hacer todo lo necesario para la salida del proyecto.

Dentro de la disciplina de desarrollo se encuentran las pruebas, en la que juega un papel fundamental el ingeniero de prueba, uno de los roles que propone la metodología. Este debe tener la capacidad para revisar los requerimientos del sistema en busca de inconsistencias, contradicciones u omisiones críticas; así mismo, debe ser capaz de definir conjuntos de casos de prueba que permitan validar, con cierto nivel de confianza, que el requerimiento se satisface con la implementación propuesta; también, debe ser capaz de implementar aquellos programas para configurar el ambiente de pruebas y ejecutar de la manera más eficiente dichas pruebas.

1.5 Artefactos del flujo de trabajo de prueba.

El Proceso Unificado de Desarrollo de software plantea 7 artefactos fundamentales para el flujo de trabajo de prueba, los cuales deben ser generados dentro de esta fase.

Artefacto modelo de prueba: describe fundamentalmente como se prueban los componentes en el modelo de implementación, ejecutando pruebas de integración y de sistemas; este artefacto puede describir también como han de ser probados aspectos específicos del sistema, por ejemplo si la interfaz de usuario es utilizable y consistente o si el manual de usuario del sistema cumple con su cometido.

Artefacto caso de prueba: especifica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones bajo las que ha de probarse.

Técnicas de diseño de casos de prueba.

Existen tres enfoques para el diseño de casos de prueba:

- El enfoque estructural o de caja blanca.
- El enfoque funcional o de caja negra.
- El enfoque aleatorio, menos conocido y el cual consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

Artefacto componente de prueba: automatizan uno o varios procedimientos de pruebas o partes de ellos, estos componentes pueden ser desarrollados utilizando un lenguaje de guiones o un lenguaje de programación, o pueden ser grabados con una herramienta de automatización de pruebas; se utilizan además para probar los componentes en el modelo de implementación, proporcionando entradas de pruebas, controlando y monitorizando la ejecución de los componentes a probar.

Artefacto procedimiento de prueba: especifica como realizar uno o varios casos de prueba o partes de estos. Un procedimiento de prueba puede ser una instrucción para un individuo sobre como ha de realizar un caso de prueba manualmente o puede ser una especificación de cómo interaccionar manualmente con una herramienta de automatización de pruebas para crear componentes ejecutables de prueba.

Artefacto plan de prueba: describe las estrategias, recursos y planificación de las pruebas. La estrategia de prueba incluye la definición del tipo de prueba a realizar para cada iteración y sus objetivos, el nivel de cobertura de prueba y de código necesario y el porcentaje de pruebas que deberían ejecutarse con un resultado específico.

El plan de pruebas se realiza para determinar el alcance, el enfoque, el calendario, los recursos requeridos, los responsables y el manejo de riesgos de un proceso de pruebas. [Teruel, 2001]

Está constituido por un conjunto de pruebas que indican las propiedades que se quieren probar, por ejemplo, robustez, fiabilidad, etc. Incluye también la forma de medir el resultado, en qué consisten las pruebas, y se define cual es el resultado que se espera.

Si no está claro qué se quiere probar, cómo se debe probar y el análisis de los resultados se hace a simple vista, las pruebas carecen entonces de utilidad. De ahí que los casos de pruebas se constituyan de tres bloques de información:

- El propósito de la prueba.
- Los pasos de ejecución de la prueba.
- El resultado que se espera.

El desarrollo de un plan de pruebas incluye:

- Identificador del plan.

Preferiblemente alguna palabra que permita relacionarlo con su alcance, por ej. TP-Global (plan global del proceso de pruebas. Como todo artefacto del desarrollo, está sujeto a control de configuración, por lo que debe distinguirse adicionalmente la versión y fecha del plan.

- Alcance
  Indica el tipo de prueba, las propiedades y elementos del software a ser probado.

- Ítems a probar

Indica la configuración a probar y las condiciones mínimas que debe cumplir para comenzar a aplicarle el plan. Por un lado, es difícil y riesgoso probar una configuración que aún reporta fallas; por otro lado, si se espera a que todos los módulos estén perfectos, puede que se detecten fallas graves demasiado tarde.

- Estrategia

Describe la técnica, patrón y/o herramientas a utilizarse en el diseño de los casos de prueba. En lo posible la estrategia debe precisar el número mínimo de casos de prueba a diseñar. También explica el grado de automatización que se exigirá, tanto para la generación de casos de prueba como para su ejecución.

- Categorización de la configuración
  Explica las condiciones bajo las cuales el plan debe ser:
  - Suspendido
  - Repetido
  - Culminado

En algunas circunstancias (las cuales deben ser explicitadas) el proceso de prueba debe suspenderse en vista de los defectos o fallas que se han detectado. Al corregirse los defectos, el proceso de prueba previsto por el plan puede continuar, pero debe explicitarse a partir de qué punto, ya que puede ser necesario repetir algunas pruebas. Los criterios de culminación pueden ser tan simples como aprobar el número mínimo de casos de prueba diseñados, o tan complejo como tomar en cuenta no sólo el número mínimo, sino también el tiempo previsto para las pruebas y la tasa de detección de fallas.

- Tangibles

  Explicita los documentos a entregarse al culminar el proceso previsto por el plan, por ej. subplanes, especificación de pruebas, casos de prueba, resumen gerencial del proceso y bitácora de pruebas.

- Procedimientos especiales

  Identifica el grafo de las tareas necesarias para preparar y ejecutar las pruebas, así como cualquier habilidad especial que se requiere.

- Recursos

  Especifica las propiedades necesarias y deseables del ambiente de prueba, incluyendo las características del hardware, el software de sistemas (p. ej. el sistema operativo), cualquier otro software necesario para llevar a cabo las pruebas, así como la colocación específica del software a probar (p. ej. qué módulos se colocan en qué máquinas de una red local) y la configuración del software de apoyo.

  - La sección incluye un estimado de los recursos humanos necesarios para el proceso. También se indican cualquier requerimiento especial del proceso: actualización de licencias, espacio de oficina, tiempo en la máquina de producción, seguridad.

- Calendario

  Esta sección describe los hitos del proceso de prueba y el grafo de dependencia en el tiempo de las tareas a realizar.

- Manejo de riesgos

  Explica los riesgos del plan, las acciones mitigantes y de contingencia.

- Responsables

  Especifica quién es el responsable de cada una de las tareas previstas en el plan.

Artefacto defecto: es una anomalía del sistema, como por ejemplo un síntoma de un fallo de software o un problema descubierto en una revisión. Un defecto puede ser utilizado para localizar cualquier cosa que los desarrolladores necesitan registrar como síntoma de un problema en el sistema que ellos necesitan controlar y resolver.

Artefacto evaluación de prueba: es una evaluación de los resultados de los esfuerzos de prueba, tales como la cobertura del caso de prueba, la cobertura del código y el estado de los defectos.

1.6 Actividades del flujo de trabajo de prueba.

Las actividades fundamentales que se desarrollan en este flujo de trabajo son:

- Planificar las pruebas.
- Diseñar las pruebas.
- Implementar las pruebas.
- Realizar pruebas de integración.
- Realizar pruebas de sistema.
- Evaluar pruebas.

1.7 Pruebas de software.

Las pruebas constituyen actividades fundamentales en muchos procesos de desarrollo, incluyendo el del software. Según el IEEE 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".* [IEEE, 1991]

Otro concepto importante a tomar en consideración es el emitido por Pressman en su edición de 1998, que plantea lo siguiente: *"La prueba del software es un elemento critico para la garantía de calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación".* [Pressman, 1998]

Se puede concluir que la prueba de software permite determinar si el producto satisface las especificaciones establecidas así como detectar errores que pudieran resultar en un mal funcionamiento de la aplicación.

1.8 Antecedentes de las pruebas de software.

El desarrollo de pruebas 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 permite un desarrollo con mayor velocidad. 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.

La tendencia de las pruebas de software es 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. Es decir, sondeos preventivos que permitan ejecutar las pruebas tan pronto como el software esté listo y con ello no sólo descubrir errores, sino evitarlos. El segundo punto se refiere a 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; es 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 hasta entonces, un nuevo proceso en la confección de los mismos: el proceso de prueba.

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 más cara que el propio desarrollo y diseño de los distintos programas que conforman el sistema.

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.

1.9 Características de las pruebas de software.

La etapa de pruebas de componentes comprende las pruebas de funcionamiento de los componentes claramente identificables. Estos son funciones o grupos de métodos conjuntados en módulos o en objetivos. Durante las pruebas de integración, estos componentes se integran para formar subsistemas o el sistema completo. En esta etapa, las pruebas se enfocan en las interacciones entre los componentes y en la funcionalidad y el desempeño del sistema como un todo. Si embargo, inevitablemente los defectos en los componentes que no han sido tomadas en cuenta durante las pruebas iniciales se ponen al descubierto durante las pruebas de integración.

Para muchos sistemas, los programadores tienen la responsabilidad de probar sus propios códigos (módulos u objetos). Una vez realizado, el trabajo se pasa a un equipo de integración que integra los módulos de los diferentes desarrolladores, construye el software y prueba el sistema conjunto, sin embargo, para sistemas críticos se utiliza un proceso más formal, donde probadores independientes son

responsables de todas las etapas del proceso de pruebas. Las pruebas se desarrollan de forma independiente y se lleva a cabo un registro detallado.

Cuando se prueban sistemas críticos, una especificación detallada de cada componente de software es utilizado por el equipo independiente para generar las pruebas del sistema. Sin embargo en muchos casos, llevar a cabo las pruebas es un proceso más intuitivo, puesto que no existe tiempo para redactar las especificaciones detalladas de cada parte de un sistema de software. En su lugar, las interfaces de los componentes principales del sistema se especifican y los programadores individuales y los equipos de programación toman la responsabilidad de diseñar, desarrollar y probar estos componentes. Las pruebas por lo general se basan en una compresión intuitiva de como operan estos componentes.

Si embargo, las pruebas de integración se basan en una especificación escrita del sistema. Un equipo aparte es responsable de las pruebas de integración.

Algunas de las características típicas del desarrollo de software basado en el ciclo de vida son:

- La prueba no puede asegurar la ausencia de defectos; solo pueden demostrar que existen defectos en el software.
- La realización de controles periódicos, normalmente coincidiendo con los hitos de los proyectos o la terminación de documentos. Estos controles pretenden una evaluación de la calidad de los productos generados (especificación de requisitos, documentos de diseño, etc.) para poder detectar posibles defectos cuanto antes. Sin embargo, todo sistema o aplicación, independientemente de estas revisiones, debe ser probado mediante su ejecución controlada antes de ser entregado al cliente. Estas ejecuciones o ensayos de funcionamiento, posteriores a la terminación del código del software, se denominan habitualmente pruebas.
- Las pruebas se deben realizar, en el entorno en el que se utilizará el sistema, lo que incluye el personal que lo maneja.
- La etapa de pruebas no debe ser posterior a la confección de un programa, tiene que ser paralela a la programación.

- Las pruebas no comienzan formalmente hasta que un número mínimo predeterminado ha instalado el software. En particular, usuarios que tardan en comenzar, generalmente nunca lo hacen y ponen en peligro todo el proceso.

1.9.1 Niveles de Prueba.

La Prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Teniendo en cuenta esto, las pruebas se agrupan por niveles, de acuerdo con las diferentes etapas del proceso de desarrollo:

- Prueba de desarrollador: Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas han sido consideradas solo para la prueba de unidad, aunque en algunos casos pueden ejecutar pruebas de integración. Se recomienda que cubran más que las pruebas de unidad.
- Prueba independiente: Es la prueba que es diseñada e implementada por alguien independiente del grupo de desarrolladores. El objetivo de estas pruebas es proporcionar una perspectiva diferente y en un ambiente más rico que los desarrolladores. Una vista de la prueba independiente es la prueba independiente de los stakeholder, que son pruebas basadas en las necesidades y preocupaciones de los stakeholders.
- Prueba de Unidad: Es la prueba enfocada a los elementos testeables más pequeños del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca.
- Prueba de Integración: Es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Se diseñan para descubrir errores en las especificaciones de las interfaces de los paquetes y son responsabilidad de desarrolladores y de independientes, sin solaparse las pruebas.
- Prueba de sistema: Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados.

- Prueba de aceptación: Prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

Es importante tener en cuenta que las pruebas de unidad son implementadas en la iteración más temprana como el primer nivel de prueba. Pero en un proceso iterativo ejecutar todas las pruebas de unidad antes de pasar a niveles siguientes de prueba como regla es inapropiada. Una mejor estrategia es identificar las pruebas de unidad, integración y sistema que ofrecen mayor potencial para encontrar errores y entonces implementarlas y ejecutarlas.

1.10 Tipos de pruebas de software.

1.10.1 Pruebas de caja negra.

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 se apoyan en la especificación de requisitos del módulo.

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 equivalentes a otros. Si logramos dividir 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.

Una vez identificadas las clases de equivalencia significativas en nuestro módulo, se procede a coger un valor de cada clase, que no esté justamente al límite de la clase. Este valor aleatorio, hará las veces de cualquier valor normal que se le pueda pasar en la ejecución real.

Para definir las clases de equivalencia es necesario tener en cuenta un conjunto de reglas:

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

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

a. Escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.
b. Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de pruebas. Con la aplicación de esa técnica se obtiene un conjunto de pruebas que reduce el número de casos de pruebas y nos dicen algo sobre la presencia o ausencia de errores.

A menudo se plantea que las pruebas a los softwares nunca terminan, simplemente se transfiere del desarrollador al cliente. Cada vez que el cliente usa el programa está llevando a cabo una prueba. Aplicando el diseño de casos de pruebas al software en cuestión se puede conseguir una prueba más completa y descubrir y corregir el mayor número de errores antes de que comiencen las “pruebas del cliente”.

1.10.2 Pruebas estructurales o de caja Blanca.

Las pruebas de caja blanca se basan en la creación de casos de prueba, que garanticen se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método, además que

ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa. Ejecutar todos los bucles o ciclos con los límites que se les haya definido y ejecutar las estructuras internas de datos para asegurar su validez. Las pruebas de caja blanca constituyen una de las pruebas más importantes que se le aplican a los productos software, pues disminuyen de forma significativa el número de errores existentes en los sistemas.

1.10.2.1 Técnica del camino básico.

Es una de las técnicas aplicadas en las pruebas de caja blanca. Esta prueba consiste en derivar casos de prueba a partir de un conjunto de caminos básicos independientes por los cuales puede circular el flujo de control.

Los pasos que se siguen son:

- Dibujar el Grafo de Flujo asociado, a partir del código fuente.
- Calcular la complejidad ciclomática.
- Determinar los caminos básicos asociados.
- Preparar los casos de prueba que obligan a la ejecución de cada camino asociado.

Notación del grafo de flujo.

Para la aplicación de la técnica del camino básico, se debe tener en cuenta la representación del flujo de control que representa el Grafo de Flujo.

El grafo de flujo está compuesto por nodos y aristas y estos a su vez crean regiones. *Los nodos* son aquellos que se representan mediante círculos que corresponden a una secuencia de procesos o a una sentencia de decisión. Existen también nodos que no se asocian, son los más usados para el inicio y final del grafo.

Las aristas se representan mediante flechas, y representan el flujo de control. Una arista debe terminar siempre en un nodo, aunque el nodo no represente ninguna secuencia procedimental.

Y *las regiones* son las áreas delimitadas por las aristas y nodos, incluyendo el espacio exterior del grafo. La cantidad de regiones, coincide con la cantidad de caminos independientes del conjunto básico de un programa.

Cada nodo del Grafo corresponde a una o varias sentencias de código, pues cada segmento de código puede traducirse a un Grafo de Flujo.

A continuación se muestran las notaciones de las instrucciones necesarias para construir un grafo así como la representación de un grafo de flujo, en el cual aparecen señalados sus componentes.

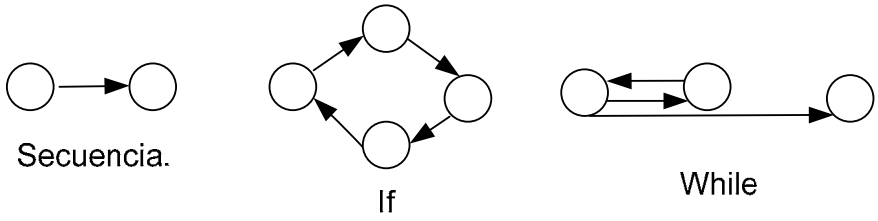


Fig. 1 Notación de grafos de flujo para las instrucciones: *Secuenciales, If, While.*

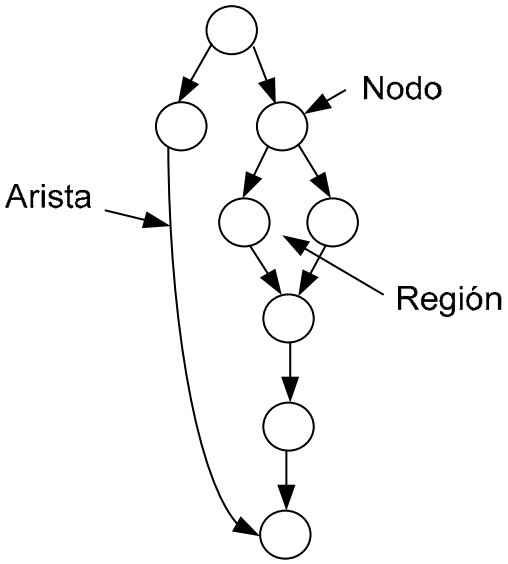
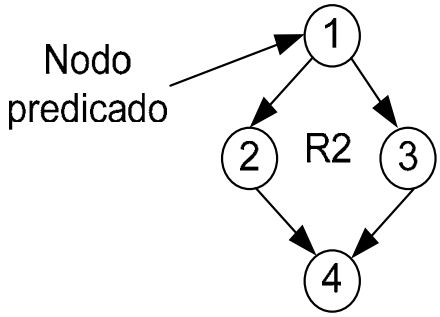


Fig. 2 Grafo de flujo. Su composición.

Ejemplo de cómo elaborar un grafo de flujo.

Para elaborar un grafo de flujo hay que tener en cuenta la particularidad del segmento de código, a partir de cual se genera dicho grafo. Si contiene instrucciones tales como **IF** *a* **OR** *b* **THEN** entonces se crea un nodo aparte para cada una de las condiciones(*a* o *b*); cada nodo que contiene una condición se denomina *nodo predicado* y está caracterizado porque dos o más aristas emergen de él, ver Fig. 3.




```
function ActivarAuditiva(formulario)
{
  if(formulario.auditiva.checked == true) 1
   {
          formulario.sordosi.disabled=false; 2
          formulario.sordono.disabled=false; 2
        }
  else
   {
          formulario.sordosi.disabled=true; 3
          formulario.sordosi.checked = false 3
          formulario.sordono.disabled=true; 3
          formulario.sordono.checked = false 3
        }
} 4
```

Fig. 3 Representación de un grafo de flujo, a partir de un segmento de código del RECUDIS.

Complejidad ciclomática.

La complejidad ciclomática proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Un *camino independiente* es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente.

Ejemplo: En la Fig. 3 un ejemplo de camino independiente sería:

Camino 1-2-4

Camino 1-3-4

Si se diseñan pruebas que fuercen el recorrido de esos caminos, se garantiza que se ejecute al menos una vez cada sentencia del programa y que cada condición se ejecute en sus variantes verdadera y falsa. Se debe tener en cuenta que de un mismo diseño procedimental se pueden derivar varios conjuntos básicos. [Ma Isabel, 2001][Pressman, 2000].

Existen tres formas diferentes para calcular la complejidad ciclomática (**V (G)**).
La primera variante es empleando la fórmula de:
V (G)= A-N+2.
Donde **A** significa el número de aristas y **N** el número de nodos.
Una segunda variante sería:
V (G) = P + 1

Donde **P** significa el número de nodos predicados.
Y una tercera variante, contar el número de regiones, el cual se corresponde con la complejidad ciclomática del grafo.

Ejemplo: Teniendo en cuenta la Fig. 3:
1. El grafo de flujo tiene dos regiones.
2. V (G) = 4 aristas – 4 nodos + 2 = 2.
3. V (G) = 1 nodo predicado + 1 = 2.

Concluyendo que la complejidad ciclomática del grafo representado en la Fig. 3 es dos.

Casos de prueba.

Una vez concluida la representación del grafo, generados los caminos básicos, y calculada la complejidad ciclomática, se procede a la elaboración de los casos de prueba, elaborándose un caso de prueba por cada camino básico.
Un ejemplo de derivación de casos de pruebas basándose en el grafo de la Fig. 3 sería de la siguiente manera:

Camino 1-2-4
Caso de prueba: activar auditiva
Entrada: formulario.auditiva.checked = true.
Resultado: Se activa el formulario “auditiva” para seleccionar si el discapacitado es sordo.
Condiciones: Que exista el discapacitado.

Camino 1-3-4
Caso de prueba: no presenta discapacidad auditiva.
Entrada: formulario.auditiva.checked = false.
Resultado: no se activa el formulario “auditiva”.
Condiciones: Que exista el discapacitado.

Luego de confeccionar los casos de prueba se ejecutan cada uno de estos y se comparan los resultados con los esperados. Una vez terminados todos los casos de prueba, se estará seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez.

Es importante considerar que algunos caminos no se pueden probar de forma aislada. O sea, la combinación de datos requerida para recorrer el camino no se puede obtener con el flujo normal del programa. En tales casos, estos caminos se prueban como parte de otra prueba de camino.

1.10.3 Pruebas de integración.

Una vez que se probaron los componentes individuales del programa, deben integrarse para crear un sistema parcial, o completo. Este proceso de integración comprende la construcción del sistema y probar el sistema resultante. Las pruebas de integración se desarrollan a partir de las especificaciones de los componentes, de la especificación del sistema y dan inicio tan pronto como estén disponibles versiones utilizables de algunos componentes del sistema.

La principal dificultad que surge en la pruebas de integración es localizar los errores que se descubren durante el proceso. Existen interacciones complejas entre los componentes del sistema y cuando se descubre una salida anómala, es difícil encontrar la fuente del error. Para hacer más fácil la localización de errores, se utiliza un enfoque incremental para la integración y pruebas del sistema. De forma inicial, se

debe integrar una configuración mínima del sistema y probarlo. Luego se agregan componentes a esta configuración mínima y se prueban después de agregados en cada incremento.

1.10.3.1 Pruebas descendentes y ascendentes.

La estrategia de pruebas descendentes y ascendentes refleja diferentes enfoques de la integración del sistema. En la integración descendente, los componentes de los niveles altos de un sistema se integran y prueban antes de que se complete su diseño e implementación. En la integración ascendente, los componentes de los niveles bajos se integran y prueban antes de que se desarrollen los de niveles altos.

Las pruebas descendentes son una parte integral del proceso de desarrollo descendente, en el cual este último inicia con los componentes de los niveles altos y desciende en la jerarquía. Estos tienen la misma interfaz que los componentes pero funcionalidad muy limitada. Después que se programa y prueba el primer componente de nivel alto, se implementan y prueban sus subcomponentes, de la misma forma. Este proceso continúa hasta que los componentes de nivel bajo se implementen y así queda completamente probado el sistema.

En contraste, las pruebas ascendentes comprenden integrar y probar los módulos en los niveles bajos, después asciende por la jerarquía de los módulos hasta que el módulo final se prueba. Este enfoque no requiere que el diseño arquitectónico del sistema esté completo, por lo que se puede comenzar en una etapa inicial del proceso de desarrollo. Se emplea donde el sistema reutiliza y modifica componentes de otros sistemas.

Las pruebas de integración descendente y ascendente se comparan teniendo en cuenta lo siguiente:

- Validación arquitectónica: las pruebas descendentes son susceptibles a descubrir errores en la arquitectura del sistema y en el diseño de alto nivel en las etapas iniciales del proceso de desarrollo.
- Demostración del sistema: en el desarrollo descendente se dispone de un sistema funcional limitado en las primeras etapas de desarrollo.
- Implementación de las pruebas.

- Observación de las pruebas: tanto las pruebas ascendentes como descendentes pueden tener problema con la observación de la prueba. En muchos sistemas, los niveles más altos del sistema que se implementan primero en un proceso descendente no generan salidas; sin embargo, para probar estos niveles, se les debe forzar a hacerlo. El probador debe crear un entorno artificial para generar los resultados de la prueba. Para las pruebas ascendentes, también es necesario crear un entorno artificial con el fin de que se pueda observar la ejecución de los componentes de los niveles inferiores.

1.10.4 Pruebas de interfaces.

Las pruebas de interfaces toman lugar cuando los módulos o subsistemas se integran para crear sistemas más grandes. Cada módulo o subsistema tiene una interfaz definida que es llamada por otros componentes del programa. El objetivo de las pruebas de interfaces es detectar las fallas que introdujeron en el sistema debido a errores de las interfaces o suposiciones no válidas acerca de las interfaces.

1.10.5 Pruebas de esfuerzo.

Una vez que el sistema se integra completamente es posible probarlo para verificar propiedades emergentes tales como el desempeño y la fiabilidad. Se tienen que diseñar pruebas de desempeño para asegurar que el sistema puede procesar la carga pretendida. Por lo general esto comprende planear una serie de pruebas donde la carga se incrementa de forma constante hasta que el desempeño del sistema sea inaceptable.

1.10.6 Pruebas de seguridad.

Garantizan que los usuarios están restringidos a funciones específicas o su acceso está limitado únicamente a los datos a que están autorizados; sólo aquellos usuarios autorizados a acceder al sistema son capaces de ejecutar las funciones de este. Su objetivo fundamental es comprobar los niveles de seguridad lógica del sistema.

1.10.7 Banco de trabajo de pruebas.

Llevar a cabo las pruebas es una fase cara y laboriosa del proceso del software. Como resultado, las herramientas de prueba estaban entre las herramientas de software a desarrollar. En la actualidad, estas herramientas ofrecen un cúmulo de recursos y su utilización reduce de forma importante el costo en el proceso de pruebas. Las diversas herramientas de prueba se integran en bancos de trabajo de pruebas.

Las herramientas que se incluyen son:

- Administrador de prueba: Administra la ejecución de las pruebas del programa.
- Generador de datos de prueba: Genera los datos de prueba para el programa a probar.
- Predictor: Genera predicciones de los resultados de prueba esperados.
- Comparador de archivos: Compara los resultados de las pruebas del programa con los resultados de prueba previos y reporta las diferencias entre ellos.
- Generador de informes: Provee la definición del informe y el recurso de generación para los resultados de la prueba.
- Analizador dinámico: Agrega código a un programa para contar el número de veces que se ejecuta cada instrucción.
- Simulador: provee diferentes clases de simuladores. Los simuladores objetivos simulan la máquina sobre la que se ejecuta el programa.

1.11 Herramientas para realizar pruebas de software.

1.11.1 TEST.

Es una unidad de pruebas automatizada y producto del análisis de código estándar, que trabaja sobre clases escritas en la plataforma Microsoft .NET, sin requerir que los desarrolladores realicen un solo caso de prueba.

TEST mejora la fiabilidad, funcionalidad, seguridad, desarrollo y mantenimiento de las aplicaciones .NET. Trabaja con lenguajes de programación que utilizan el marco Microsoft .NET, incluyendo C # y VB.NET. TEST prueba la conformidad con más de 250 reglas de desarrollo. Expone los problemas de fiabilidad, genera y ejecuta casos de pruebas NUnit diseñados para alcanzar una alta cobertura y precisar el código que puede causar excepciones en tiempo de ejecución, expone errores funcionales y asegura una

continua funcionalidad. Las pruebas de regresión automatizadas de TEST capturan el comportamiento de la línea base del código e identifica problemas introducidos por modificaciones del código.
TEST mejora la calidad de las aplicaciones .NET durante todo el ciclo de vida del software y aumenta la productividad en todo el equipo. Los desarrolladores lo pueden utilizar para probar el código mientras lo realizan y los miembros del equipo de calidad lo pueden emplear para identificar problemas críticos antes de un inminente despliegue.

1.11.2 JUnit.

A lo largo de los años, han existido programadores experimentados que han desarrollado métodos y herramientas para escribir las pruebas más cómodamente. Dos de estos desarrolladores fueron Kent Beck y Eric Gamma (dos personalidades en el campo de la informática, el primero por desarrollar la metodología Extreme Programming y el otro por su contribución al libro Design Patterns), quienes desarrollaron una colección de clases para Java llamada JUnit. Con estas clases, se pueden desarrollar casos de pruebas y colecciones de prueba fácilmente, heredando de sus propias clases base. Además esta herramienta ofrece una serie de interfaces gráficas para visualizar estas pruebas, ejecutarlas, ver sus resultados, seleccionar aquellas que queremos ejecutar, etc.

A estas colecciones de clases, junto con sus herramientas se las conoce como "Marcos de pruebas", ya que gracias a ellas, se tiene toda la infraestructura necesaria para desarrollar pruebas unitarias de forma rápida, cómoda, extensible y fiable. JUnit ha tenido tanto éxito que se ha extendido a otros muchos lenguajes de programación, gracias al trabajo desinteresado de muchos programadores. Todos los frameworks heredados de JUnit han recibido la denominación xUnit, con la que se indica que se trata de una migración, y se siguen las normas que marcó JUnit. Entre los frameworks xUnit, existen versiones para C/C++ (CUnit y CPPUnit), Delphi (DUnit), PHP (PHPUnit), HTML (HTMLUnit), NUnit (plataforma .NET), VBUnit (Visual Basic) y otras. El modo de trabajar de todos los frameworks xUnit es parecido entre ellos, aunque cada uno con las peculiaridades de su propio lenguaje. La idea principal es desarrollar una unidad que se encargue de probar a otra unidad.

1.11.3 ChecKing.

Es la herramienta de monitorización del proceso de desarrollo software y sus resultados, que cubre las necesidades de organizaciones que desean controlar la calidad del software antes de su puesta en práctica.

Para ello la herramienta sigue un modelo de métricas en el que se integran medidas obtenidas automáticamente del proceso de desarrollo (actividad, requisitos, defectos y cambios) y de elementos analizables del software: código fuente, documentación del proyecto, scripts de construcción, y scripts de pruebas. ChecKing añade transparencia al ciclo de desarrollo, capturando, integrando y presentando de forma automática dichas métricas, obtenidas automáticamente a través de diversas herramientas y sistemas externos: analizadores, gestores de defectos, sistemas de control de versiones, herramientas de pruebas, IDEs y otras herramientas usadas por los desarrolladores.

1.12 Estado actual de las pruebas en Cuba.

Actualmente en nuestro país no existe una vasta cultura sobre los temas de calidad, específicamente en lo que concierne a las pruebas de software. En los grupos de desarrollo de proyectos no existen personas que desempeñen el rol de probador, de ahí que en el período que corresponde probar el sistema no se cumpla con lo establecido para ello. Por lo general a los sistemas no se le aplican pruebas de caja blanca ni de caja negra, aunque estas últimas se realizan con más frecuencia y la documentación de ambas no se realiza como es debido. Entre las principales causas del inadecuado cumplimiento de los criterios de desarrollo se le confieren mayor importancia a la falta de una adecuada estructura organizativa dentro de las empresas para este efecto, el desconocimiento del tema o técnica a aplicar, el escaso personal capacitado y disponible para ello y el escaso tiempo del que se dispone para este proceso.

En la UCI, sin embargo, los recursos humanos y materiales existentes facilitan el proceso de pruebas a los productos que se desarrollan. La estructura de calidad de software a nivel central y en cada una de las facultades permite la administración de calidad y la inserción desde etapas tempranas del desarrollo, de probadores a los distintos proyectos. Se han alcanzado resultados que demuestran el avance progresivo en este tema, como las pruebas realizadas a los diferentes proyectos con la República Bolivariana de Venezuela, entre los que se destaca Registro y Notarias. Muchos de los compromisos de las facultades con instituciones nacionales también se ven beneficiados, siendo un ejemplo de ello el presente trabajo.

Finalmente se puede plantear que la calidad de software en Cuba es un tema que se encuentra aún en sus inicios, en la fase de estudio. Aunque muchas de las empresas ya están comenzando a profundizar en él, todavía falta mucho por conocer en este sentido para lograr hacer de los sistemas nacionales, productos estables y con una calidad comercializable a cualquier nivel.

1.13 Conclusiones.

El aseguramiento de la calidad de software se ha convertido en una necesidad prioritaria para las organizaciones que desarrollan este tipo de productos, ya sea para uso interno o para implementaciones externas en clientes, porque cada vez más los errores en el software repercuten directa o indirectamente en graves consecuencias para la organización.

Un error que desde el punto de vista de codificación puede ser relativamente simple de corregir puede resultar muy difícil y costoso de detectar y llegar a ocasionar efectos graves. Normalmente el equipo de desarrollo del proyecto se encuentra presionado por la necesidad de cumplir con las fechas establecidas en el cronograma y el proceso de pruebas no se cumple o se ejecuta de una manera desorganizada, sin método y sin considerar los tiempos establecidos para esta fase. El resultado es un software sin las pruebas mínimas requeridas y sin el nivel de calidad esperado.

Capítulo 2. Diseño y aplicación de las pruebas al Registro Cubano de Discapacitados.

2.1 Introducción.

El estudio psicosocial de las personas con discapacidades en Cuba constituye la primera etapa de uno de los programas más sensibles de la Revolución, cuya concepción y ejecución responde a lo definido por el compañero Fidel cuando planteó: “batalla de ideas no significa solo los principios, teoría, conocimientos, cultura, argumentos, réplica y contrarréplica, destruir mentiras y sembrar verdades; significa hechos y realizaciones concretas”. El estudio psicosocial de las personas con discapacidades y estudio psicopedagógico social y clínico genético de las personas con retraso mental, se realiza con el objetivo de profundizar en este universo y sus peculiaridades, conocer posibles causas e identificar necesidades.

Para realizar este estudio es necesario la recolección y almacenamiento de datos, así como la gestión y actualización de dicha información desde cualquier parte de nuestro país en la cual exista una consulta especializada. Actualmente el intercambio de información entre las provincias es muy lento pues se efectúa de forma personal y los datos relacionados con el proyecto se entregan en papeles, al realizar un informe se requiere más esfuerzo y tiempo por parte del equipo investigativo. Fue necesario crear una herramienta automatizada para facilitar dicho estudio y que le permita a los especialistas del CNGM controlar el proceso de Gestión de la Información en el centro y específicamente en el Registro Cubano de Discapacitados. Dada la importancia de este sistema y la necesidad del mismo por parte de esta institución, se diseña un plan de pruebas que abarca los módulos Nuevo, Modificar y los Reportes estadísticos para así validar la calidad del producto, el cumplimiento de los requisitos y especificaciones para ser entregado al cliente.

2.2 Registro Cubano de Discapacitados.

El Registro Cubano de Discapacitados, se realiza con el objetivo de estudiar el universo de personas con discapacidades y sus peculiaridades biopsicosociales, conocer las posibles causas e identificar principales necesidades. Esta investigación no tiene precedentes en el mundo, pues se hace muy difícil contar con el

capital humano necesario para lograr esta actividad, se resaltan así los valores creados por nuestra sociedad.

RECUDIS incluye un módulo *Nuevo*, el cual brinda la posibilidad de insertar a las personas discapacitadas, de acuerdo a la discapacidad que presenten, así como otras características particulares. Para ello es utilizado un único instrumento por paciente, es decir, una planilla que llenará el defectólogo de cada paciente. En caso de que haya sido atendido por otro defectólogo, para modificar cualquier información, deben reunirse ambos doctores.

Para la realización de esta modificación se implementó el módulo *Modificar*, en el cual el defectólogo debe tener cuidado al cambiar información, pues cambiaría los datos respecto a la situación real del paciente.

Además cuenta con una serie de *reportes estadísticos*, que brindan a los usuarios la posibilidad de acceder desde cualquier región del país a informaciones para estudios genéticos. Cada usuario tiene un rol en la aplicación, entre ellos el genetista que obtiene los reportes que necesite para cada estudio.

A los módulos explicados anteriormente se le aplican pruebas de caja blanca y pruebas de caja negra, con el objetivo de comprobar la funcionalidad de los mismos.

Los atributos que se miden varían, y su relevancia y el nivel de detalle dependen del propósito de cada aplicación particular y los objetivos de la prueba. Hay algunos atributos que son comunes a la hora de desarrollar las pruebas de software, por ejemplo si la aplicación cumple sus requerimientos y especificaciones, si se ejecuta sin errores, si es fácil de usar, si es aceptable para el usuario, si la documentación está completa y correcta. También aquellos que determinan si la aplicación es confiable, precisa, completa, rápida, utilizable, flexible y bien documentada. (Napal, y otros, 2003)

Además se pueden medir atributos del código fuente para saber cuán bien estructurado y reutilizable es, ya que estos atributos determinarán el futuro de la aplicación. En RECUDIS se miden principalmente los atributos de funcionamiento, para validar con la comprobación de estos la calidad del sistema.

2.3 Características a probar.

Las características a probar, dependen mucho de la funcionalidad de cada módulo, y del objetivo de las pruebas que se apliquen.

Existen atributos que siempre se miden a la hora de evaluar un software, entre ellos, el cumplimiento de los requerimientos exigidos por el cliente, si se ejecuta sin errores, si es un software amigable, fácil de usar y de comprender para el usuario, si la documentación está completa, y lo más importante si está correcta.

Existen atributos que miden la confiabilidad de la aplicación, si es rápido, flexible, seguro en precisión y muy bien documentado. Así como también existen atributos que miden la efectividad del código fuente, con el objetivo de comprobar si el mismo es reutilizable, y si está bien estructurado, siendo estos dos aspectos fundamentales para una aplicación, pues si no es sólida y robusta tal vez no pueda mantenerse por mucho tiempo, y resulte sumamente costosa.

En la aplicación que se prueba en el presente trabajo (RECUDIS) los principales atributos a medir son los de funcionalidad, trazándose como objetivo principal la detección de errores de este tipo para la mejora de versiones posteriores del producto.

2.4 Plan de pruebas general.

2.4.1 Objetivos.

El objetivo principal de aplicar pruebas al software RECUDIS es ejecutar el programa, habiendo realizado los casos de prueba necesarios con anterioridad, para encontrar la mayor cantidad de fallos en el sistema y corregirlos, de esta manera darle más consistencia al software y una mayor confiabilidad.

Es válido destacar que para realizar un buen caso de prueba debe cumplir dos condiciones fundamentales:

- Debe poseer una alta probabilidad de encontrar un error que no ha sido detectado hasta el momento.
- Un caso de prueba correcto es aquel que descubre un error, alcanzando así el éxito.

2.4.2 Propósito.

Este documento describe el Plan de pruebas para el sistema RECUDIS. En concreto define los siguientes objetivos específicos:

- Identifica los elementos que se van a probar.
- Describe la estrategia de pruebas a seguir en el proceso de prueba.
- Identifica los recursos necesarios para llevar a cabo el proceso de prueba y estima los esfuerzos que conlleva.
- Lista los resultados que se obtienen de las actividades de prueba.

2.4.3 Alcance.

Este Plan de Pruebas describe las pruebas estructurales y funcionales que se aplicarán al sistema software desarrollado. El objetivo es probar los requisitos definidos en la Especificación de requisitos y en el Modelo de casos de uso.

2.4.4 Requerimientos de las pruebas.

Identifica los elementos (casos de uso, requisitos funcionales y requisitos no funcionales) que son objetivos de las pruebas. Es decir, los elementos que vamos a probar.

- **Pruebas de funcionalidad:**
  - ✓ Verificar el caso de uso Insertar Datos del Discapacitado.
  - ✓ Verificar el caso de uso Modificar Discapacitado.
  - ✓ Verificar el caso de uso Buscar Discapacitado.
  - ✓ Verificar el caso de uso Mostrar Reportes Estadísticos.

- **Pruebas de interfaz de usuario.**
  - ✓ Navegar a través de todos los casos de uso, verificando que cada interfaz de usuario se comprende fácilmente.
  - ✓ Verificar que la navegación a través de todo el sistema es fácil.

2.4.6 Estrategia de prueba.

Se reflejan los pasos a seguir para el flujo de trabajo desarrollado, así como la especificación de las pruebas a implementar definiendo en cada una las técnicas utilizadas. Para el caso específico de RECUDIS, que se encuentra en el nivel de pruebas de sistema, se diseñaron e implementaron las siguientes pruebas.

2.4.6.1 Tipos de pruebas y técnicas empleadas.

Prueba aplicada.	**Técnica implementada.**
Caja Negra. Las pruebas funcionales o de caja negra son un enfoque para llevar a cabo pruebas, las cuales se derivan de la especificación del programa o componentes. Pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada, que se produce una salida correcta, y que la integridad de la información externa se mantiene.	*Partición de equivalencia.* Para la implementación de esta técnica, se comenzó por encontrar las clases de equivalencia, y seguido a esto está la realización de los casos de prueba para las clases válidas que son las resaltantes en cantidad en un caso de prueba. Para las no válidas se diseña un caso de prueba por cada clase. Para todas las pruebas realizadas se especifica el posible resultado, comprobando al final la ocurrencia de los errores, comparando el resultado obtenido con el esperado.

Caja Blanca. Son el enfoque de prueba donde éstas se generan a partir del conocimiento de la estructura e implementación del software. Examinan la lógica del programa, garantizan que se ejerciten todos los caminos independientes de cada módulo, todas las decisiones lógicas, todos los bucles, y que se ejecuten las estructuras de datos internas.	*Camino básico.* Para implementar esta técnica se siguieron los siguientes pasos: - Analizar el código fuente del RECUDIS y hacer el grafo asociado. - Calcular la complejidad ciclomática. - Determinar los caminos básicos asociados. - Preparar los casos de prueba que obligan a la ejecución de cada camino asociado.

2.4.6.2 Pasos a seguir en las pruebas.

En el presente trabajo se realizan solamente pruebas de caja negra (CN) y pruebas de caja blanca (CB) a los módulos Nuevo, Modificar y Reporte.

Para la realización de estas pruebas se escogen aquellos casos de uso más significativos, en los que caiga el mayor peso de la funcionalidad del sistema, por ser más propensos a fallos. Se elabora un plan de pruebas por cada módulo a probar, que incluyen los casos de prueba para todos los procedimientos. En el caso de las pruebas de CN también se elabora un plan para cada módulo, en el que se incluye todos los casos para todas las entradas de cada módulo.

En el caso de las pruebas de CB lo primero es hacer el grafo a partir de las instrucciones, le sigue la obtención de todos los caminos básicos, mediante los cuales se puede recorrer el procedimiento sin excluir variantes, y para finalizar se hacen los casos de prueba por cada camino lógico.

Para las pruebas de CN primero se obtienen las clases válidas e inválidas y seguido a esto está la realización de los casos de prueba para cada clase. Para todas las pruebas realizadas se especifica el

posible resultado, comprobando al final la ocurrencia de los errores, comparando el resultado obtenido con el esperado.

2.4.7 Recursos

RRHH	Recursos de software	Recursos de hardware
El equipo está formado por 2 ingenieros de pruebas que desempeñan todos los roles del flujo de trabajo de pruebas.	SO Windows XP PHP Mysql. Office 2003 o superior. Internet Explorer , Netscape, Mozilla o FireFox	Microprocesador Pentium Memoria RAM de 248Mb o superior Disco duro con 20Gb mínimo

2.4.8 Calendario

Rol	Actividad	Fecha de inicio	Fecha de Fin
Administrador de pruebas	Generar el plan de pruebas.	**01/04**	**05/04**
Diseñador de pruebas	Diseñar los casos de pruebas.	**20/04**	**30/04**
Probador	Ejecutar pruebas.	**02/05**	**10/05**
Probador	Encontrar los errores.	**02/05**	**10/05**
Analista de pruebas	Documentar los defectos.	**15/05**	**25/05**
Administrador de pruebas.	Evaluación de los resultados.	**15/05**	**25/05**

2.5 Plan de pruebas específico.

Caja negra.

Con las pruebas de CN se comprueba la funcionalidad del sistema, además de la validez de las entradas por cada módulo, viendo también si posee una interfaz amigable para el usuario. Para el desarrollo de las mismas, se aplica la técnica de partición de equivalencia que resulta efectiva a la hora de comprobar la validez de cada entrada en los diferentes módulos.

Pruebas de requerimiento e interfaz.

Para la realización de las pruebas, se tienen en cuenta los requerimientos, que dan al traste con la funcionalidad del sistema, pues constituyen condiciones o capacidades que debe cumplir el sistema. En la documentación elaborada por los desarrolladores de RECUDIS se especificaron una serie de requisitos que enmarcan la funcionalidad del software. Los requisitos mostrados a continuación, son a modo general, extraídos de cada módulo, para dar una idea de la funcionalidad del sistema.

Requerimientos generales del sistema.

1. Modificar los datos de un discapacitado existente.
2. Insertar un discapacitado nuevo.
3. Buscar datos de un discapacitado.
4. Determinar la cantidad de discapacitados según la clasificación de su discapacidad.
5. Determinar la cantidad de discapacitados según el sexo.
6. Determinar la cantidad de discapacitados según el grupo de causas referidas de su discapacidad.
7. Determinar la cantidad de discapacitados según la causa en el período posnatal.
8. Determinar la cantidad de discapacitados según evaluación funcional.
9. Determinar la cantidad de discapacitados según su ocupación.
10. Determinar la cantidad de discapacitados según la ubicación laboral.
11. Determinar la cantidad de discapacitados con o sin amparo filial.
12. Determinar la cantidad de discapacitados sin control de esfínteres.
13. Determinar la cantidad de discapacitados que consumen alcohol según su clasificación.
14. Determinar la cantidad de discapacitados fallecidos.
15. Determinar la cantidad de discapacitados que son PCI

2.6 Plan de pruebas de caja negra.

2.6.1 Caja negra para el requerimiento “Insertar un discapacitado nuevo”, del Mod-Nuevo.

Descripción general.

El caso de uso que da cumplimiento a este requerimiento se inicia cuando el defectólogo desea insertar un discapacitado en el sistema. El discapacitado ofrece los datos que se van a incluir en la aplicación y el defectólogo procede a incluirlo en la base de datos.

A este requerimiento se le realizaron las siguientes pruebas:
Pruebas de funcionalidad mediante la técnica partición de equivalencia.

Descripción de la funcionalidad.

Inserta un nuevo discapacitado según los datos necesarios brindados por el paciente con el objetivo de mantener un control sobre los mismos.

Flujo central.

1. Autenticarse como usuario municipal.
2. Buscar en el Menú Discapacitados la opción Nuevo.
3. Al presionar Nuevo aparece la interfaz con los datos a insertar por el defectólogo.
4. Una vez entrado todos los datos presionar el botón Insertar.
5. Se inserta toda la información del discapacitado en la BD y termina el caso de uso.

Condiciones de ejecución.

1. Estar autenticado como usuario municipal.
2. Llenar todos los campos necesarios para insertar un nuevo discapacitado.

2.6.1.1 Iteración 1 para “Insertar datos generales”.

Clases Válidas	Clases Inválidas	Resultado Esperado	Condición de entrada
Nombre Se entra una cadena de letras.		Inserta correctamente.	Caso 1 para nombre. Nombre = Yohandris
Nombre Se entran dos cadenas de letras separadas por un espacio		Inserta correctamente	Caso 2 para nombre. Nombre = julio cesar
	Nombre Se entra junto con las cadenas de letras otro tipo de caracteres (números, especiales).	Muestre un mensaje advirtiendo que entre solo letras en este campo.	Caso 3 para nombre. Nombre = Yohandris.
	Nombre El campo se deja vacío.	Muestre un mensaje advirtiendo que entre el nombre.	Caso 4 para nombre. Nombre =
Apellidos Se entran dos cadenas de letras separadas por un espacio.		Inserta correctamente.	Caso 1 para apellidos. Apellidos = Pupo Blez
Apellidos Se entran más		Inserta correctamente	Caso 2 para apellidos. Apellidos = Pupo Montes

de dos cadenas de letras separadas por más de un espacio			de Oca
	Apellidos Se entra una cadena de letras.	Muestre un mensaje advirtiendo que entre dos apellidos	Caso 3 para apellidos. Apellidos = pupo
	Apellidos Se entra junto con las cadenas de letras separadas por espacios algún otro carácter (números y especiales)	Muestre un mensaje advirtiendo que entre solo letras en este campo	Caso 4 para apellidos. Apellidos = Pupo Blez.
	Apellidos El campo se deja vacío.	Muestre un mensaje advirtiendo que entre los apellidos.	Caso 5 para apellidos. Apellidos =
Dirección Se entran varias cadenas de caracteres		Inserta correctamente.	Caso 1 para dirección. Dirección = edificio 69, apto 205, uci
	Dirección Se deja el campo vacío.	Muestre un mensaje advirtiendo que entre la dirección.	Caso 2 para dirección. Dirección =
Edad Se entra un número en el rango de 1 a		Inserta correctamente.	Caso 1 para edad. Edad = 34

100.			
	Edad Se entra un número mayor que 100.	Muestre un mensaje advirtiendo el rango en que debe estar la edad.	Caso 2 para edad. Edad = 200
	Edad Se deja el campo vacío.	Muestre un mensaje advirtiendo que entre la edad.	Caso 3 para edad. Edad =
Año de nacimiento Se entra un número en el rango de 1900 a 2099		Inserta correctamente.	Caso 1 para año de nacimiento. Año de nacimiento = 1984
	Año de nacimiento Se entra un número menor que 1900 o mayor que 2099.	Muestre un mensaje advirtiendo el rango en que debe estar el año	Caso 2 para año de nacimiento. Año de nacimiento = 1899
	Año de nacimiento El campo se deja vacío.	Muestre un mensaje advirtiendo que entre al año.	Caso 3 para año de nacimiento. Año de nacimiento =
Identificación Se entra una cantidad de números igual a 11.		Inserta correctamente.	Caso 1 para identificación. Identificación = 84080526349
	Identificación Se entra una cantidad de números menor que 11.	Muestre un mensaje advirtiendo que debe entrar 11 números.	Caso 2 para identificación. Identificación = 8408052639
	Identificación	Muestre un mensaje	Caso 3 para identificación.

	Se entra una cadena de números y otro tipo de caracteres (letras o especiales)	advirtiendo que entre solo números.	Identificación = 8408052639.
	Identificación Se deja el campo vacío.	Muestre un mensaje advirtiendo que entre el número de identificación.	Caso 4 para identificación. Identificación =
Área de salud Se entran varias cadenas de caracteres.		Inserta correctamente.	Caso 1 para área de salud. Área de salud = uci residencia 2
	Área de salud Se deja el campo vacío.	Muestre un mensaje advirtiendo que entre el área.	Caso 2 para área de salud. Área de salud =
CMF(Consultorio Médico de la Familia) Se entran varias cadenas de caracteres.		Inserta correctamente.	Caso 1 para CMF. CMF = residencia facultad 6
	CMF Se deja el campo vacío.	Muestre un mensaje advirtiendo que entre el CMF	Caso 2 para CMF. CMF =

2.6.1.2 Iteración 1 para “Condiciones de vida y Apoyo Familiar”.

Clases Válidas	Clases Inválidas	Resultado Esperado	Condición de entrada
Número de personas en el dormitorio Se entra un número en el rango de 0 a 5		Inserta satisfactoriamente.	Caso 1 para número de personas en el dormitorio. Número de personas en el dormitorio = 3
	Número de personas en el dormitorio Se entra un número mayor que 5	Muestre un mensaje advirtiendo que el número es incorrecto.	Caso 2 para número de personas en el dormitorio. Número de personas en el dormitorio = 6
	Número de personas en el dormitorio Se deja el campo vacío	Muestre un mensaje advirtiendo que entre el número de personas en el dormitorio.	Caso 3 para número de personas en el dormitorio. Número de personas en el dormitorio =
N° de convivientes en el núcleo. Se entra un número en el rango de 0 a 15		Inserta satisfactoriamente.	Caso 1 para número de convivientes en el núcleo. Número de convivientes en el núcleo = 4
	N° de convivientes en el núcleo. Se entra un número mayor que 15	Muestre un mensaje advirtiendo que el número es incorrecto.	Caso 2 para número de convivientes en el núcleo. Número de convivientes en el núcleo = 16

	N^{o} de convivientes en el núcleo. Se deja el campo vacío	Muestre un mensaje advirtiendo que entre el número de convivientes en el núcleo.	Caso 3 para número de convivientes en el núcleo. Número de convivientes en el núcleo =
Ingreso total. Se entra una cadena de números		Inserta correctamente.	Caso 1 para ingreso total. Ingreso total = 5000
	Ingreso total. Se entra una cadena con números y otro tipo de caracteres.	Muestre un mensaje advirtiendo que entre solo números en este campo.	Caso 2 para ingreso total. Ingreso total = 5000 pesos Ingreso total = $5000
	Ingreso total. Se deja el campo vacío	Muestre un mensaje advirtiendo que entre el ingreso total.	Caso 3 para ingreso total. Ingreso total =

2.6.1.3 Iteración 1 para “Observaciones”.

Clases Válidas	Clases Inválidas	Resultado Esperado	Condición de entrada
Observaciones Se entran varias cadenas de caracteres.		Inserta satisfactoriamente.	Caso 1 para observaciones. Observaciones = proceso de pruebas funcionales al módulo nuevo del RECUDIS

2.6.1.4 Iteración 1 para “Situación laboral”.

Clases Válidas	Clases Inválidas	Resultado Esperado	Condición de entrada
Salario Se entra una cadena de números.		Inserta satisfactoriamente.	Caso 1 para salario. Salario = 350
	Salario Se entra una cadena de números y otro tipo de caracteres.	Muestre un mensaje advirtiendo que entre solo números en este campo.	Caso 2 para salario. Salario = $500 Salario = 10 CUC
	Salario Se deja el campo vacío	Muestre un mensaje advirtiendo que entre el salario.	Caso 3 para salario. Salario =
¿Tiene ayuda económica? Se entra una cadena de números.		Inserta satisfactoriamente.	Caso 1 para ¿Tiene ayuda económica? ¿Tiene ayuda económica? = 100
	¿Tiene ayuda económica? Se entra una cadena de números y/u otro tipo de caracteres.	Muestre un mensaje advirtiendo que entre solo números en el campo “Ayuda Económica”.	Caso 2 para ¿Tiene ayuda económica? ¿Tiene ayuda económica? = 100 pesos ¿Tiene ayuda económica? = no
	¿Tiene ayuda económica? Se deja el campo vacío.	Muestre un mensaje advirtiendo que entre la ayuda económica.	Caso 3 para ¿Tiene ayuda económica? ¿Tiene ayuda económica? =

2.6.2 Caja negra para los requerimientos “Buscar y Modificar los datos de un discapacitado existente”, del Mod-Modificar.

Descripción general.

Se buscan los discapacitados según los datos entrados y se modifican los aspectos correspondientes.
A este requerimiento se le realizaron las siguientes pruebas:
Pruebas de funcionalidad mediante la técnica de clases de equivalencia.

Descripción de la funcionalidad.

Busca los discapacitados, según los datos que se entren en los campos propuestos, se seleccionan el correspondiente y se modifican los datos.

Flujo central.

1. Buscar en el Menú Discapacitados la opción modificar.
2. Al pinchar modificar aparece la opción buscar con los diferentes campos de búsqueda.
3. Se llenan los campos o el campo para realizar la búsqueda.
4. Pichar en el botón buscar.
5. Aparece un listado con los discapacitados que cumplan con los campos llenados.
6. En el listado, encontrar el nombre del discapacitado a modificar.
7. Pinchar en la opción de modificar.
8. Aparecen opciones de los datos que pueden modificarse por discapacitados.
9. Al pinchar en la opción se despliega, y aparecen los posibles campos a modificar.
10. Se cambia el campo y se pincha en el botón modificar.

Flujo alterno

4.1 Si no existe ningún discapacitado con esas características, no muestra resultados.

Condiciones de ejecución.

1. Estar autenticado antes de realizar la búsqueda.
2. Llenar al menos un campo para realizar la búsqueda.

2.6.2.1 Iteración 1 para “Buscar discapacitado”.

Clases Válidas	Clases Inválidas	Resultado Esperado	Condición de entrada
Nombre Se entran cadenas de letras.		Búsqueda correcta	
	Nombre Se entran cadenas de letras y otro tipo de caracteres (números o especiales)	Muestre un mensaje advirtiendo que en este campo solo deben entrarse letras.	
Apellido Se entran cadenas de letras.		Búsqueda correcta	
	Apellido se entran cadenas de letras y otro tipo de caracteres (números o especiales)	Muestre un mensaje advirtiendo que en este campo solo deben entrarse letras.	

2.7 *Plan de pruebas de caja blanca.*

Caja blanca

Para la aplicación de las pruebas de caja blanca se utilizó la técnica del camino básico mediante la cual se puede comprobar el funcionamiento de cada segmento de código del producto software. Con la aplicación de esta prueba se verifica si el sistema cumple con el resultado esperado en cada una de sus variantes, comprobando si los mensajes de error, que se deben mostrar al cliente, aparecen en el momento justo y si el programa luego de emitir un error mantiene la estabilidad requerida.

2.7.1 CB-Mod-Nuevo

2.7.1.1 Insertar apoyo familiar.

Insertar_apoyofamiliar($Discapac,$_REQUEST[padre],$_REQUEST[madre],$_REQUEST[hijos],$_REQUEST[hermanos],$_REQUEST[esposo],$_REQUEST[otro]);

if($_REQUEST[padre] != 1) **1**

{$_REQUEST[padre] = 0;} **2**

if($_REQUEST[madre] != 1) **3**

{$_REQUEST[madre] = 0;} **4**

if($_REQUEST[hermanos] != 1) **5**

{$_REQUEST[hermanos] = 0;} **6**

if($_REQUEST[hijos] != 1) **7**

{$_REQUEST[hijos] = 0;} **8**

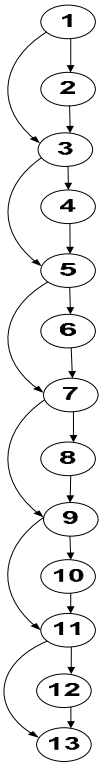
if($_REQUEST[esposo] != 1) **9**

{$_REQUEST[esposo] = 0;} **10**

if($_REQUEST[otro] != 1) **11**

{$_REQUEST[otro] = 0; } **12**

$apoyofam = $db1->Insert_apoyofamiliar($Discapac,$_REQUEST[padre],$_REQUEST[madre],$_REQUEST[hijos],$_REQUEST[hermanos],$_REQUEST[esposo],$_REQUEST[otro]); **13**



Caminos básicos

C1: 1-2-3-4-5-6-7-8-9-10-11-12-13

C2: 1-3-4-5-6-7-8-9-10-11-12-13

C3: 1-3-5-6-7-8-9-10-11-12-13

C4: 1-3-5-7-8-9-10-11-12-13

C5: 1-3-5-7-9-10-11-12-13

C6: 1-3-5-7-9-11-12-13

C7: 1-3-5-7-9-11-13

Complejidad ciclomática

V (G) = A-N+2

V (G) = 18-13+2=7

Casos de prueba.

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

Caso de prueba: Insertar apoyo familiar.

Entrada: Apoyo Familiar (padre=0, madre=0, hijo=0, hermano=0, esposo=0, otro=0).

Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. En este caso recibe ayuda de todos los integrantes del núcleo y de otras personas.

Condiciones: Que exista el discapacitado.

Camino : 1-3-4-5-6-7-8-9-10-11-12-13

Caso de prueba: Insertar apoyo familiar de: madre, hijo, hermano, esposo, otro.

Entrada: Apoyo Familiar (padre=1, madre=0, hijo=0, hermano=0, esposo=0, otro=0).

Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. Para el presente caso no recibe el apoyo del padre.
Condiciones: Que exista el discapacitado.

Camino: 1-3-5-6-7-8-9-10-11-12-13
Caso de prueba: Insertar apoyo familiar de: hijo, hermano, esposo, otro.
Entrada: Apoyo Familiar (padre=1, madre=1, hijo=0, hermano=0, esposo=0, otro=0).
Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. Para el presente caso no recibe el apoyo del padre ni de la madre.
Condiciones: Que exista el discapacitado.

Camino: 1-3-5-7-8-9-10-11-12-13
Caso de prueba: Insertar apoyo familiar de: hijo, esposo, otro.
Entrada: Apoyo Familiar (padre=1, madre=1, hijo=0, hermano=1, esposo=0, otro=0).
Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. Para el presente caso no recibe el apoyo del padre, la madre, ni de los hermanos.
Condiciones: Que exista el discapacitado.

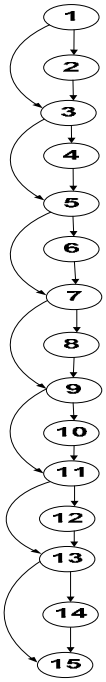
Camino: 1-3-5-7-9-10-11-12-13
Caso de prueba: Insertar apoyo familiar de: esposo, otro.
Entrada: Apoyo Familiar (padre=1, madre=1, hijo=1, hermano=1, esposo=0, otro=0).
Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. Para el presente caso recibe el apoyo del esposo(a) y de otro familiar.
Condiciones: Que exista el discapacitado.

Camino: 1-3-5-7-9-11-12-13
Caso de prueba: Insertar apoyo familiar de: otro.
Entrada: Apoyo Familiar (padre=1, madre=1, hijo=1, hermano=1, esposo=1, otro=0).

Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. Para el presente caso recibe el apoyo de otro familiar.
Condiciones: Que exista el discapacitado.

Camino: 1-3-5-7-9-11-13
Caso de prueba: Insertar apoyo familiar de ninguno.
Entrada: Apoyo Familiar (padre=1, madre=1, hijo=1, hermano=1, esposo=1, otro=1).
Resultado: El sistema registra aquellas personas que componen el núcleo familiar y de las cuales el discapacitado recibe apoyo. Para el presente caso no recibe el apoyo de ningún familiar.
Condiciones: Que exista el discapacitado.

2.7.1.2 Insertar persona discapacitada (Clasificación en relación con el consumo de bebidas alcohólicas).



Caminos básicos:

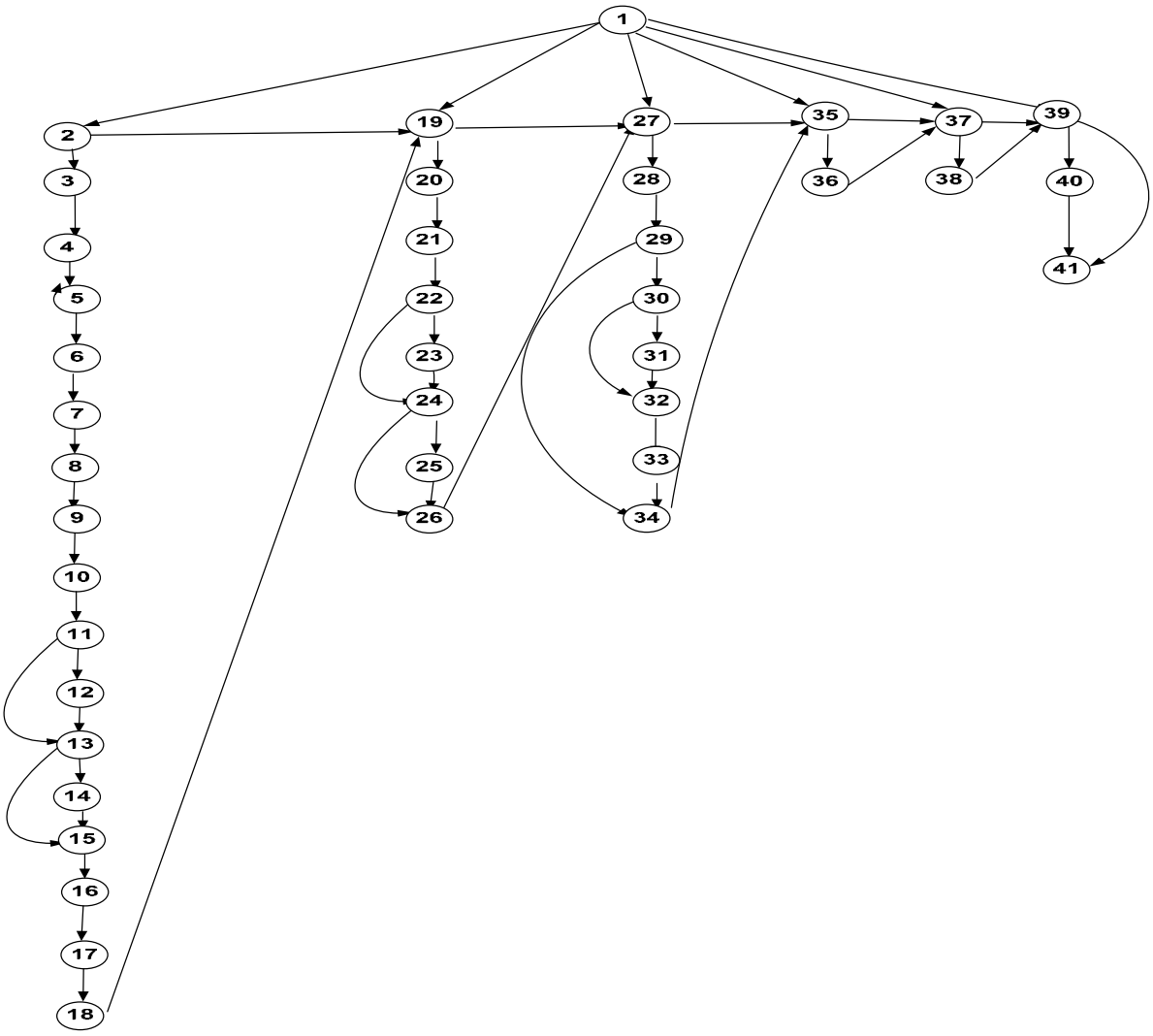
C1:1-2-3-4-5-6-7-8-9-10-11-12-13-14-15
C2: 1-3-4-5-6-7-8-9-10-11-12-13-14-15
C3: 1-3-5-6-7-8-9-10-11-12-13-14-15
C4: 1-3-5-7-8-9-10-11-12-13-14-15
C5: 1-3-5-7-9-10-11-12-13-14-15
C6: 1-3-5-7-9-11-12-13-14-15
C7: 1-3-5-7-9-11-13-14-15
C8:1-3-5-7-9-11-13-15

Complejidad ciclomática

V (G) = A-N+2
V (G) = 21-15+2
V (G) = 8

Para casos de prueba ver Anexo I.

2.7.1.3 Insertar tipo de discapacidad.

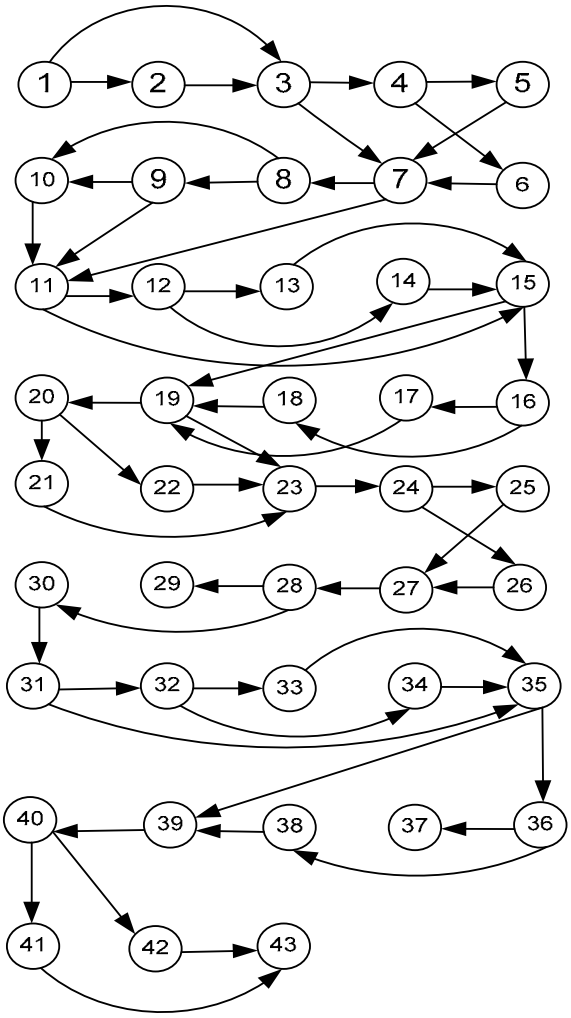


Complejidad ciclomática.

V (G) = A-N+2
V (G) = 62-41+2
V (G) = 23

Para caminos básicos y casos de prueba ver Anexo I.

2.7.1.4 Función Chequear Número.



Complejidad ciclomática

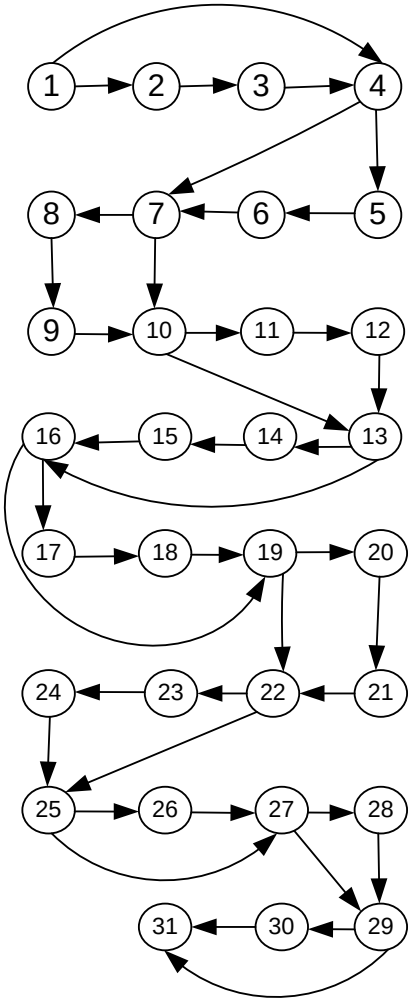
V (G)= A-N+2

V (G) =63-43+2

V (G) =22

Para caminos básicos y casos de prueba ver Anexo I.

2.7.1.5 Función validar.



Complejidad ciclomática

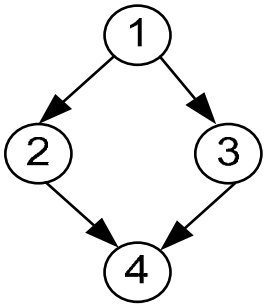
V (G) = A-N+2

V (G) = 41-31+2

V (G) =12

Para caminos básicos y casos de prueba ver Anexo I.

2.7.1.6 Función Sordo.



Caminos básicos:

C1: 1-2-4

C2: 1-3-4

Complejidad ciclomática.

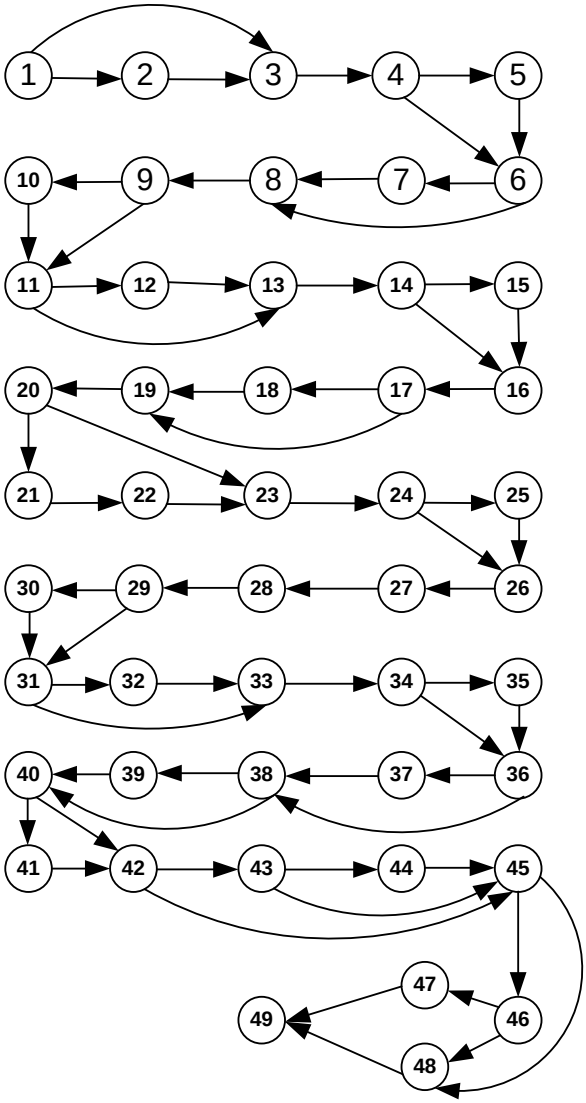
V (G)=A-N+2

V (G)=4-4+2

V (G)=2

Para casos de prueba ver Anexo I

2.7.1.7 Función validar Datos Generales.



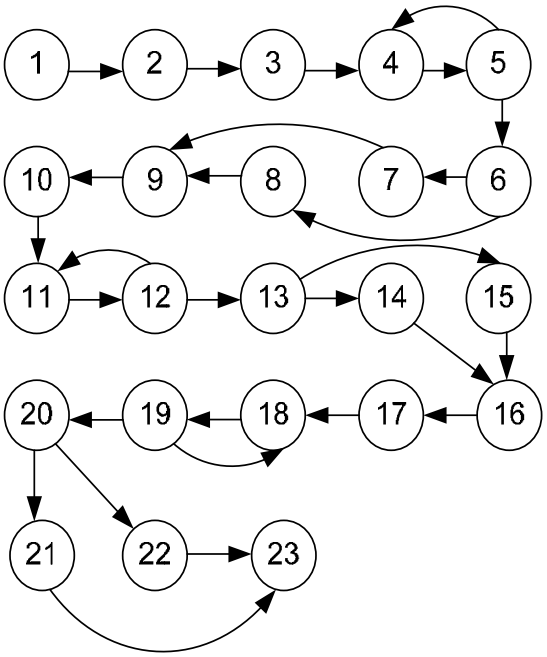
Complejidad ciclomática

V (G)= A-N+2
V (G)= 67-49+2
V (G)=20

Para caminos básicos y casos de prueba ver Anexo I.

2.7.2 CB-Mod-Reportes

2.7.2.1 Crear amparo filial, nivel nacional.



Caminos básicos:

C1: 1-2-3-4-5-6-7-9-10-11-12-13-14-16-17-18-19-20-22-23

C2: 1-2-3-4-5-6-8-9-10-11-12-13-15-16-17-18-19-20-21-23

C3: 1-2-3-4-5-6-7-9-10-11-12-13-14-16-17-18-19-18-19-20-22-23

C4: 1-2-3-4-5-6-7-9-10-11-12-11-12-13-15-16-17-18-19-20-21-23

C5: 1-2-3-4-5-4-5-6-7-9-10-11-12-13-15-16-17-18-19-20-22-23

C6: 1-2-3-4-5-4-5-6-8-9-10-11-12-13-14-16-17-18-19-20-21-23

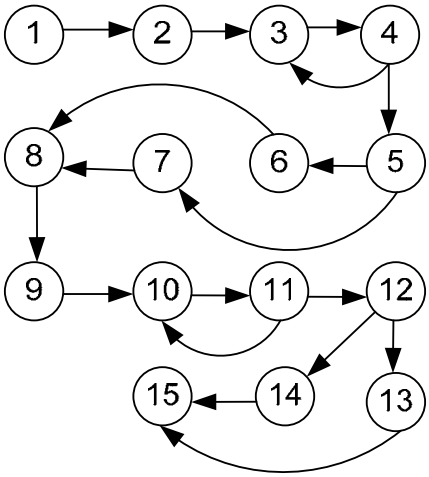
C7: 1-2-3-4-5-6-8-9-10-11-12-11-12-13-14-16-17-18-19-20-22-23

Complejidad ciclomática

V (G)= A-N+2
V (G)= 28-23+2
V (G)= 7

Para casos de prueba ver Anexo II.

2.7.2.3 Crear amparo filial, nivel provincial.



Caminos básicos.

C1: -2-3-4-3-4-5-6-8-9-10-11-12-13-15

C2: 1-2-3-4-5-6-8-9-10-11-12-13-15

C3: 1-2-3-4-5-7-8-9-10-11-12-14-15

C4: 1-2-3-4-5-6-8-9-10-11-10-11-12-13-15

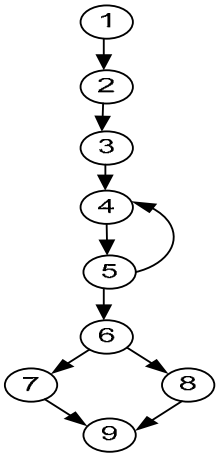
C5: 1-2-3-4-5-6-8-9-10-11-12-14-15

Complejidad ciclomática

V (G)= A-N+2
V (G)= 18-15+2
V (G)= 5

Para casos de prueba ver Anexo II.

2.7.2.4 Crear Amparo filial, nivel municipal.



Caminos básicos.

C1: 1-2-3-4-5-6-7-9

C2: 1-2-3-4-5-6-8-9

C3: 1-2-3-4-5-4-5-6-7-9

Complejidad ciclomática:
V (G)= A-N+2
V (G)=10-9+2
V (G)= 3

Para casos de prueba ver Anexo II.

2.8 Conclusiones.

En este capítulo se describe muy brevemente el sistema RECUDIS, se incluye una explicación de cada uno de sus módulos, a los cuales se les diseñaron pruebas de caja negra y caja blanca con el objetivo de detectar errores para su posterior corrección.

Para desarrollar las pruebas, se tienen en cuenta una serie de aspectos, algunos de los cuales se enfatizan en este capítulo, como son:

- Elaboración del plan de pruebas.
- Las pruebas se diseñan para comprobar que existen errores, no para demostrar que no existen.
- A partir de esto se determinan las características más importantes a probar.
- Las pruebas de caja negra se basan en la especificación del sistema.
- Las pruebas de código identifican casos de prueba que permiten ejecutar todos los "caminos" en un programa, asegurando que todas las sentencias se ejecutan al menos una vez.

Capítulo 3. Resultados

3.1 Introducción.

En el proceso de pruebas es muy importante el registro de la documentación pertinente. Los resultados obtenidos en cada iteración de pruebas, en términos de cantidad de no conformidades por tipo deben ser registradas en un entregable para el equipo de desarrollo, donde adicionalmente se presentan hallazgos generales relevantes y recomendaciones. Como responsabilidad de este proceso está el análisis de los resultados obtenidos en cada una de las pruebas versus los resultados esperados, con base en lo cual se determina la presencia de no conformidades. El grupo de proyecto que desarrolla el sistema debe asimilar e interiorizar el análisis realizado por el equipo de pruebas, debido a que de esto depende el mejoramiento del producto de software pero a la vez la formación de bases para el mejoramiento de aspectos relevantes en el proceso de desarrollo como: especificación de requerimientos de software, estandarización, control de configuración, etc. La retroalimentación realizada por el grupo de pruebas debe basarse en medidas del proceso de pruebas y del software, pero también en análisis cualitativos como la descripción de las causas más frecuentes de los defectos, tendencia del proceso de pruebas de software, etc.

En la documentación elaborada para mostrar los resultados de las pruebas al equipo de desarrollo se deben separar los resultados por tipo de pruebas para un mejor entendimiento y posterior corrección. Para las pruebas de caja negra deben registrarse los resultados alcanzados por cada módulo, de las pruebas de requerimientos y de las pruebas de interfaz; en el caso de las pruebas de caja blanca los resultados se deben dar por cada procedimiento inspeccionado, cada Unit, cada módulo y el sistema en general. Es muy importante considerar cuáles son los errores que más pueden ocurrir ya sea en la interfaz, los requerimientos, por la parte del código o las entradas de datos.

“Esto puede hacerse de varias formas, una de ellas es listando defectos conocidos en la práctica del desarrollo, generando suposiciones a partir de la naturaleza del software, teniendo en cuenta complejidad de algunos métodos, etc. De cualquier forma tener una idea previa de los defectos que pueden existir será aprovechable para todos los encargados de probar el software.” También se puede tener en cuenta la opinión de cada programador acerca de su módulo, ya que saben donde se pueden encontrar puntos complejos propensos a errores, donde puede haber algún error aun no corregido, etc.

Los resultados pueden ser reflejados de formas diferentes, en este trabajo específicamente se describen por módulos de cada tipo de prueba, los resultados finales en general y una comparación con los resultados esperados.

3.2 Resultados de caja negra.

Las pruebas de caja negra (CN) se ejecutaron utilizando una PC con sistema operativo Microsoft Windows XP Profesional, versión 2002, Service Pack 2, con 248 MB de memoria RAM y microprocesador Pentium IV. Se instaló la aplicación web en la PC y se probaron cada uno de los módulos implementados, los cuales requerían conexión a una base de datos que se encontraba en la propia PC. Las pruebas fueron realizadas por dos probadores que no pertenecen al equipo de desarrollo del sistema. Los resultados de los errores comprenden los de los casos diseñados para las entradas, los de las pruebas de requerimientos y los errores de interfaz.

Para las pruebas de caja negra se elabora una tabla resumen, con el siguiente formato:

Elemento	**No**	**No conformidad**	**Aspecto correspondiente**	**Etapa de detección Código del CP**	**Importancia**	**Recomendación**

Donde:

Elemento: especifica el tipo de error (de la aplicación, en la interfaz, otro).
No: un número que identifique al error.
No conformidad: se especifica el error detectado, la no conformidad encontrada.
Aspecto correspondiente: se describe la no conformidad, cualquier otro aspecto relevante y se especifica su localización.
Etapa de la detección: en que etapa del ciclo de vida se detecto el error.
Importancia: se marca con una X si el error tiene una influencia alta en el proyecto, se suele establecer también de forma cualitativa (baja, media y alta).
Recomendación: aclaraciones y sugerencias del equipo de prueba para una mejor comprensión del desarrollador.

3.2.1 Resultados CN-Mod-Nuevo.

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
Interfaz	1	Falta de ortografía “Área ’	Menú discapacitado/Nuevo/ Datos Generales/Área de salud	Prueba de interfaz	Media	Revisar y corregir
Interfaz	2	Falta de ortografía “Mestizo”	Menú discapacitado/Nuevo/Datos Generales/ Piel	Prueba de interfaz	Media	Revisar y corregir
Interfaz	3	Falta de ortografía en los mensajes de advertencia. “las tildes se representan como caracteres especiales”	En todo el módulo.	Prueba de interfaz	Media	Revisar y corregir
Aplicación	4	Error en el campo de la edad. “acepta números muy grandes”	Menú discapacitado/Nuevo/Datos Generales/ Edad	Prueba de entrada de datos. Caso 2 para edad.	alta	Revisar y corregir, se sugiere limitar el mayor número a entrar en este campo.
Interfaz	5	Falta de ortografía	Menú discapacitado/Nuev	Prueba de interfaz	Media	Revisar y corregir

		“Convivientes”	o/Condiciones de vida y apoyo familiar/ Nº de Convivientes en el núcleo			
Interfaz	6	Falta de ortografía “Arquitectónicas”	Menú discapacitado/Nuevo/Condiciones de vida y apoyo familiar/ Barreras Arquitectónicas/mensaje de error	Prueba de interfaz	Media	Revisar y corregir
Interfaz	7	Falta de ortografía “Económica”	Menú discapacitado/Nuevo/Situación laboral/ ¿Tiene Ayuda Económica?	Prueba de interfaz	Media	Revisar y corregir
Aplicación	8	Como respuesta a una pregunta de si o no, se validó números	Menú discapacitado/Nuevo/Situación laboral/ ¿Tiene Ayuda Económica?	Prueba de funcionalidad Caso 2 de ¿Tiene Ayuda Económica?	alta	Se recomienda dar respuesta a la pregunta, con las opciones: si o no, si la respuesta es positiva se active entonces la opción para especificar la

						ayuda recibida, de lo contrario hacer la pregunta de forma tal que el usuario se de cuenta que tiene que entrar números.
Interfaz	9	Falta de ortografía “Ocupación”	Menú discapacitado/Nuevo/Situación laboral/mensaje “Escriba la ocupación del discapacitado”	Prueba de interfaz	Media	Revisar y corregir, es el mensaje de advertencia cuando no se marca la opción de “Ocupación”
Interfaz	10	Falta de ortografía “vínculo”	Menú discapacitado/Nuevo/Situación laboral/mensaje “Escriba el vínculo laboral del discapacitado”	Prueba de interfaz	Media	Revisar y corregir, es el mensaje de advertencia cuando no se marca la opción de “Vínculo Laboral Actual”
Interfaz	11	Falta de	Menú	Prueba de	Media	Revisar y

		ortografía “números”	discapacitado/Nuevo/Situación laboral/mensaje “Escriba solo números en el campo salario”	interfaz		corregir, es el mensaje de advertencia cuando no se entra el “Salario”
Interfaz	12	Falta de ortografía “números y económica”	Menú discapacitado/Nuevo/Situación laboral/ ¿Tiene Ayuda Económica?/mensaje “Escriba solo números en el campo Ayuda Económica”	Prueba de interfaz	Media	Revisar y corregir, es el mensaje de advertencia cuando no se entra la “Ayuda Económica”
Interfaz	13	Falta de ortografía “ausencia”	Menú discapacitado/Nuevo/Estado de salud/Tiene ausencia de miembros	Prueba de interfaz	Media	Revisar y corregir
Interfaz	14	Falta de ortografía ”semiválido”	Menú discapacitado/Nuevo/Estado de salud/Evaluación funcional	Prueba de interfaz	Media	Revisar y corregir
Interfaz	15	Falta de ortografía “cuña”, no reconoce la	Menú discapacitado/Nuevo/Necesidad de implementos/Neces	Prueba de interfaz	Media	Revisar y corregir

		“ñ” en los mensajes	ita pato o cuña/mensaje “Escriba si necesita pato o cuña”			
Interfaz	16	Falta de ortografía e incoherencia en el mensaje “Escriba si está ingresado en el hogar de ancianos”	Menú discapacitado/Nuevo/Necesidad de servicio de salud/Se encuentra ingresado/mensaje “Escriba si está ingreso en hogar de ancianos”	Prueba de interfaz	Media	Revisar y corregir
Interfaz	17	Falta de ortografía y tildes no reconocidas en las causas por factor prenatal, perinatal y postnatal. “biológico, genético, químico, físico, mecánico, congénitas, tránsito, sistémica,	Menú discapacitado/Nuevo/Causa de la discapacidad/Factor //Causa	Prueba de interfaz	Media	Revisar y corregir

		desnutrición ”				
Aplicación	18	El sistema debe exigir el factor y la causa de la discapacidad.	Menú discapacitado/Nuevo/Causa de la discapacidad/	Prueba de funcionalidad	Alta	Revisar y corregir, se sugiere forzar la entrada de un factor y su causa, si no, cómo fue que el paciente llego a esa discapacidad.
Aplicación	19	En el campo nombre no permite escribir un nombre compuesto	Menú discapacitado/Nuevo/Datos Generales/Nombre	Prueba de funcionalidad. Caso 2 para nombre.	alta	Revisar y corregir, dada la existencia de este tipo de nombres.
Aplicación	20	En el campo apellidos no se permiten más de dos cadenas de letras y un solo espacio.	Menú discapacitado/Nuevo/Datos Generales/Apellidos	Prueba de funcionalidad Caso 2 para apellidos.	alta	Revisar y arreglar, pues al igual que los nombres, existen apellidos compuestos que no se podrían entrar. (Ej. Montes de Oca; De la Fuente.)

Aplicación	21	En el campo nombre se puede entrar una cadena de letras infinita.	Menú discapacitado/Nuevo/Datos Generales/Nombre	Prueba de funcionalidad	alta	Se sugiere limitar el número de caracteres en este campo.
Aplicación	22	En el campo número de personas en su dormitorio no especifica el rango en que debe estar el número.	Menú discapacitado/Nuevo/Condiciones de vida y apoyo familiar.	Prueba de funcionalidad	alta	Se sugiere especificar el rango del número a entrar, y ampliarlo, puede darse un caso que un dormitorio por alguna característica haya más de 5 personas entonces ya 6 por ejemplo no podría entrarse
Aplicación	23	No se forzó a marcar la opción de “Composición del núcleo familiar”	Menú discapacitado/Nuevo/Condiciones de vida y apoyo familiar.	Prueba de funcionalidad	alta	Revisar y corregir, pues en caso de que se especifiquen número de personas en

						el dormitorio y número de convivientes en el núcleo, debe haber algún tipo de composición.
Aplicación	24	Después de llenados los campos para insertar un nuevo discapacitado, da un error, provocado por la no existencia del campo 'SolicEmpleo Antes', en la base de datos.	Menú discapacitados/nuevo/insertar.	Prueba requerimientos.	alta	Revisar y corregir.

3.2.2 Resultados CN-Mod-Modificar.

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
Interfaz	1	Aparece la consulta a la BD, una vez presionado el botón buscar.	Menú discapacitado/modificar/buscar discapacitado/campo/buscar	Prueba de interfaz	Alta	Revisar y corregir
Aplicación	2	Acepta números en los campos nombre y apellidos	Menú discapacitado/modificar/buscar discapacitado/	Prueba de funcionalidad	Alta	Revisar y arreglar, se sugiere advertir al usuario cuando pinche en buscar que introduzca los datos correctos para estos campos.
Aplicación	3	Al dejar los campos de búsqueda vacíos, no muestra un aviso advirtiendo	Menú discapacitado/modificar/buscar discapacitado campo/buscar	Prueba de funcionalidad	Alta	Revisar y corregir, se sugiere advertir al usuario que al menos llene uno de

		que debe llenar al menos un campo.				los campos cuando pinche en buscar
Interfaz	4	Falta de ortografía “Físico”	Menú discapacitado/modificar/buscar discapacitado/ Tipo Discapacidad/Físico Motora	Prueba de interfaz	Media	Revisar y corregir
Interfaz	5	Falta de ortografía en las provincias, “sin tildes(Pinar del Río, Ciego de Ávila, Holguín, Guantánamo y Sancti Spíritus) y le falta la diéresis de Camagüey”	Menú discapacitado/modificar/buscar discapacitado/ Provincia	Prueba de interfaz	Media	Revisar y corregir
Interfaz	6	Falta de ortografía en los municipios de todas las provincias.	Menú discapacitado/modificar/buscar discapacitado/ Provincia/Municipio	Prueba de interfaz	media	Revisar y corregir
Interfaz	7	Falta de	Menú	Prueba de	Media	Revisar y

		ortografía en los consejos populares de todos los municipios	discapacitado/modificar/buscar discapacitado/ Provincia/Municipio/ Consejo Popular	interfaz		corregir
Aplicación	8	El usuario luego de seleccionar la provincia a buscar puede presionar el botón buscar directamente y el sistema no le advierte que debe seleccionar el municipio y el consejo popular.	Menú discapacitado/modificar/buscar discapacitado/ Provincia	Prueba de funcionalidad	alta	Puede que un usuario que por primera vez se enfrente al sistema le suceda esto y se sienta perdido, se sugiere advertir que debe seleccionar los campos necesarios para poder realizar la búsqueda.
Aplicación	9	El sistema no realiza la búsqueda por el campo “Tipo de discapacidad”	Menú discapacitado/modificar/buscar discapacitado/Tipo de discapacidad	Prueba de funcionalidad	alta	Revisar y corregir
Aplicació	11	Al seleccionar	Menú	Prueba de	alta	Revisar y

n		un discapacitado a modificar, los datos aparecen intercambiados de campo (Ej. en el nombre aparece el número de identificación) así como campos vacíos sin los cuales no se habría podido insertar previamente.	discapacitado/modificar/buscar discapacitado/Listado de los discapacitados/Modificar	funcionalidad		corregir
Aplicación	12	Cuando se presiona el botón “Modificar” no se guardan los cambios.	Menú discapacitado/modificar/buscar discapacitado/Listado de los discapacitados/Modificar	Prueba de funcionalidad	Alta	Revisar y corregir
Aplicación	13	Al presionar buscar se muestra un mensaje	Menú discapacitado/modificar/buscar discapacitado/Busc	Prueba de funcionalidad	alta	Revisar y arreglar, no se le ve funcionalidad

		“Nacional y luego al dar aceptar aparece otro mensaje en blanco”.	ar			alguna a este mensaje.

3.2.3 Resultados CN-Mod-Reportes.

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
Interfaz	1	Falta de ortografía. El sistema no reconoce las tildes, las sustituye por caracteres especiales.	Menú Reportes/Sexo	Pruebas de interfaz	Media	Revisar y corregir
Interfaz	2	Falta de ortografía en el nombre de las provincias “Guantánamo, Holguín, Ciego de Ávila, Pinar del Río” y la diéresis de	Menú Reportes/Sexo	Pruebas de interfaz	Media	Revisar y corregir

		Camagüey.				
Aplicación	3	No se especifican las ocupaciones.	Menú Reportes/Ocupación	Pruebas de funcionalidad	alta	Revisar y corregir, ya que no se identifican los números romanos con la ocupación que le corresponde
Aplicación	4	El proceso de elaboración del reporte es muy lento y muestra el error en la interfaz.	Menú Reportes	Pruebas de funcionalidad	alta	Revisar y corregir
Interfaz	5	Cuando se realiza el reporte por clasificación, en la interfaz se muestra como nombre del reporte: “Reporte por sexo”, aun siendo los datos mostrados por	Menú Reportes/Clasificación	Pruebas de funcionalidad	alta	Revisar y corregir

		clasificación.				
Interfaz	6	Cuando se realiza el reporte por vínculo laboral, en la interfaz se muestra como nombre del reporte: “Reporte por sexo”, aun siendo los datos mostrados por vínculo laboral.	Menú Reportes/Vínculo laboral	Pruebas de funcionalidad	alta	Revisar y corregir
Interfaz	7	En la barra de información donde está la ayuda no siempre se muestra el camino completo hasta el reporte actual. Ej. Correcto: (**Inicio-Reportes-**	Menú Reportes	Pruebas de interfaz	alta	Revisar y corregir

		Reportes por sexo), y en otros casos muestra (**Inicio-Reportes por sexo**) Ej. Incorrecto.				
Interfaz	8	Cuando se realiza el reporte por capacidad laboral, en la interfaz se muestra como nombre del reporte: “Reporte por sexo”, aun siendo los datos mostrados por capacidad laboral.	Menú Reportes/Capacidad Laboral	Pruebas de funcionalidad	alta	Revisar y corregir

3.3 Resultados de caja blanca.

Las pruebas de caja blanca se ejecutaron utilizando una PC con sistema operativo Microsoft Windows XP Profesional, versión 2002, Service Pack 2, con 248 MB de memoria RAM y microprocesador Pentium IV. Por ser una primera versión del software no se dispone de la documentación completa y la existente cambia constantemente, de ahí que no sea estable y no se encuentren todos los artefactos de modelación del sistema, por ello no se utilizó la herramienta de la suite del Rational para probar el software, sino que se probó directamente en el Zend Studio Client 5.0.0 depurando cada uno de los procedimientos, cada módulo exige una conexión a una base de datos que se encuentra en la misma PC. Las pruebas fueron realizadas por dos probadores que no pertenecen al grupo de desarrollo.

Para las pruebas de caja blanca se elabora una tabla resumen, con el siguiente formato:

ID Módulo	Errores	Tipo de errores	Observaciones

Donde:

ID Módulo: es el identificador del procedimiento o módulo.

Errores: especifica el error detectado, la descripción debe ser lo más precisa y completa posible.

Tipo de errores: clasifica el error según el tipo que haya arrojado el sistema (catastrófico, funcional o de propiedad).

- Catastrófico: el caso no logra producir salidas porque el ambiente de pruebas se cae o porque el software bajo prueba termina anormalmente sin producir las salidas o sin producirlas completas.
- Funcional: los valores de las salidas de la ejecución discrepan de los valores esperados.
- Propiedad: se incumple una propiedad. Por ejemplo, un error debido a que el tiempo de respuesta del sistema es mayor a lo exigido, es una falla de propiedad (desempeño).

Observaciones: aclaraciones, comentarios y sugerencias del equipo de prueba para una mejor comprensión del desarrollador.

3.3.1 Resultados CB-Mod-Nuevo.

ID Módulo	Errores	Tipo de errores	Observaciones
CB-Mod-Nuevo	**1)** Variables que son usadas antes de haber sido definidas. **2)** Error con la conexión a la BD, no se encuentra la clase mysqli. **3)** Error con la conexión a la Base de Datos, a la hora de insertar, no encuentra el campo 'SolicEmpleoAntes'. **4)** Uso del **else**, después de dos **if** anidados, sin usar llaves para delimitarlos. Cuando un **else** se usa luego de dos **if** anidados sin usar llaves para delimitarlos, uno de ellos puede desigualar la declaración del **else** con la preposición del **if**. Se recomienda siempre usar llaves en estos casos.	Los errores que hay son del tipo funcional.	**1)** No existen comentarios explicativos del código, para que el probador pueda comprender bien a la hora de realizar las pruebas. **2)** Verificar la conexión con la base de datos, pues a la hora de probar la clase acceso a datos, da error. **3)** La capa de presentación no está separada de la capa de lógica del negocio. **4)** El uso de Java Script para validar hace vulnerable el sistema, pues los datos son interpretados del lado del cliente.

3.3.2 Resultados CB-Mod-Reportes.

ID Módulo	Errores	Tipo de errores	Observaciones
CB-Mod-Reportes.	**1)** Algunos reportes no muestran los valores en el tiempo adecuado, los muestran con datos de otros campos, no con los que les corresponde o con datos insuficientes. **2)** Existen códigos de algunas funciones que no son alcanzados. **3)** Variables con valores asignados que nunca se usan. **4)** Uso del **else**, después de dos **if** anidados, sin usar llaves para delimitarlos. Cuando un **else** se usa luego de dos **if** anidados sin usar llaves para delimitarlos, uno de ellos puede desigualar la declaración del **else** con la preposición del **if**. Se recomienda siempre usar llaves en estos casos.	Los errores que hay son del tipo funcional, y de propiedad.	**1)** No existen comentarios explicativos del código, para que el probador pueda comprender bien a la hora de realizar las pruebas. **2)** En ocasiones existen errores en el tiempo de ejecución de los procedimientos, este aspecto atenta contra la estabilidad del sistema.

3.4 Resultados generales del sistema.

Una vez obtenidos los resultados de la ejecución de las pruebas de caja blanca y caja negra, se puede concluir que los errores detectados tienen un nivel de importancia considerable a pesar de no ser catastróficos. Errores que pueden hacer que el sistema no funcione correctamente, tales como procedimientos que generen fallas y no muestren las salidas esperadas o datos mostrados incorrectamente, atentando así contra la confiabilidad y funcionalidad del sistema. La búsqueda de información genera resultados intercambiados, y el tiempo de ejecución se hace en ocasiones muy lento.

En cuanto a la interfaz, los principales errores están dados por faltas de ortografía que presenta en todos sus módulos, incluyendo los mensajes mostrados por el sistema. No son errores de una importancia alta que conlleven a una incorrecta ejecución del programa o a un mal funcionamiento del mismo, pero que si afectan la calidad integral y la estética del software una vez instalado por el cliente. Además el usuario no tiene opción de cerrar la sesión, cambiar de nivel de acceso, o salir del sistema. Los reportes generados no siempre coinciden con la información mostrada y el nombre del reporte en la parte superior. Carece de una ayuda para guiar al usuario en la utilización del software y los mensajes del sistema no son lo suficientemente aclarativos para guiar y corregir al usuario en su trabajo.

3.5 Comparación entre resultados reales y esperados.

El producto RECUDIS aun no se encuentra instalado de forma definitiva en ningún centro de la red de genética del país, está en fase de prueba de aceptación por el cliente, por lo que los resultados que se esperaban dado que era la primera revisión formal a la que se sometía eran altos. No obstante, comparando los resultados esperados con los obtenidos, podemos decir que el software realiza la mayor parte de las funciones que tiene implementadas, solo contiene algunas fallas de validación y de tratamiento de errores.

Principalmente se esperaban la mayor cantidad de errores en el módulo Nuevo, específicamente en los datos de entrada, a la hora de añadir un nuevo discapacitado, siendo este el módulo de mayor cantidad de fallas, cumpliendo con lo que se esperaba. De forma genérica se puede plantear que se previeron la mayor parte de los resultados que generó el sistema, entregando los mismos para su revisión y corrección para posteriores versiones. Las fallas que no fueron previstas, se consideraron también como bastante serias, agregándose a los resultados para la fase de reparación. Las pruebas de caja blanca se

consideran satisfactorias, pues se esperaba que el código no presentara errores graves, comprobándose en los resultados obtenidos.

3.6 Conclusiones.

Al finalizar este capítulo, podemos concluir que el proceso de pruebas se llevó a cabo de forma satisfactoria, pues los casos de pruebas desarrollados, mostraron muchos de los errores del sistema, estando expuestas anteriormente todas sus particularidades. Debido a que los errores encontrados no tienen un nivel catastrófico, podemos afirmar que el software puede ser desplegado para pruebas de aceptación de usuarios, aunque es válido destacar que solo se encuentra en primera versión. Se prevé que los errores detectados pueden solucionarse en un corto período de tiempo, pues están debidamente documentados, para una rápida comprensión del equipo de desarrolladores.

Se da así por concluida la primera iteración del proceso de prueba por parte del grupo de calidad, de forma clara y entendible, documentándose todos los defectos, siendo de gran ayuda para el equipo de desarrollo en posteriores versiones.

Conclusiones

El desarrollo alcanzado por la industria de software y la necesidad de productos confiables, precisos y funcionales, hace que aumente cada vez más la exigencia por parte de clientes y usuarios de que los sistemas hayan concebido correctamente las técnicas de Ingeniería de software y las pruebas adecuadas en la etapa de comprobación en vista a lograr el nivel de calidad requerido.

Al planificarse y aplicarse las pruebas al RECUDIS, se llegaron a las siguientes conclusiones:

- Producto de la competencia en la industria del software a nivel mundial y que nuestra facultad por su perfil de bioinformática está enfrascada en proyectos críticos para la salud, las pruebas a productos en etapa de desarrollo es un factor de gran influencia en la calidad y por ende en la aceptación en el mercado.
- Se logró por parte de los probadores un amplio conocimiento acerca de las técnicas de pruebas existentes, las estrategias para su aplicación y pasos a seguir.
- Se obtuvieron casos de pruebas para las técnicas de Caja Blanca y Caja Negra que cubrieron la mayor parte de los caminos que puede recorrer el sistema durante su ejecución.
- Las pruebas realizadas se llevaron a cabo con un alto nivel de seriedad en cada uno de los módulos, teniendo en cuenta la importancia que reviste el software para el estado cubano.
- Se logró desarrollar una adecuada documentación de todo el proceso de prueba, que servirá de apoyo para la corrección de los errores encontrados en cada uno de los módulos probados.
- El sistema, a partir de estas pruebas, poseerá mayor calidad y confiabilidad.

Por otra parte podemos plantear que constituye una necesidad para la industria cubana de software, definir y aplicar un riguroso sistema que incluya todos los criterios para determinar la calidad, específicamente el criterio de pruebas, pues es un punto clave en la determinación del nivel de calidad de estos productos. De esta forma las empresas tienen la oportunidad de insertarse ventajosamente en el mercado internacional de Software, aportar al país un gran beneficio económico, así como garantizar un alto nivel de los softwares para la salud, por su complejidad y nivel de importancia para seguimiento de casos, estudios científicos y sociales.

Con el estudio realizado y la aplicación de las pruebas que se han llevado a cabo en el trabajo “Diseño y aplicación de pruebas al Registro Cubano de Discapacitados”, se ha cumplido el objetivo propuesto al inicio del trabajo, el cual consistió en diseñar y aplicar las pruebas de software al RECUDIS, con vistas a la obtención del mayor número de faltas existentes en el mismo, logrando con su ulterior eliminación obtener una aplicación con un mínimo de errores y por consiguiente mayor calidad.

Recomendaciones

Una vez concluido el proceso de pruebas, y dada la importancia que reviste la detección de errores en el momento oportuno, se recomienda:

- Al equipo de desarrolladores corregir lo más pronto posible los errores detectados.
- Una vez realizada la corrección de los defectos, continuar con más iteraciones, hasta lograr la menor cantidad de fallas.
- Debido a que el RECUDIS no está implementado en su totalidad, iniciar desde etapas tempranas el control de calidad en aquellos módulos que aun no se han desarrollado.
- Realizarle otro tipo de pruebas al producto, como pruebas de robustez, de integración, de sistema, y otras.
- Hacer extensiva la fase de prueba a los demás proyectos de la facultad, llevando la documentación necesaria cada vez que se le apliquen.
- Utilizar los conceptos, ejemplos y resultados de este trabajo para lograr que los grupos de desarrollo conozcan las técnicas de prueba y que las incluyan como una fase más dentro del desarrollo de los sistemas.
- Que el documento obtenido como resultado de este trabajo sirva de complemento a la bibliografía de las asignaturas de Ingeniería de Software y cursos del grupo de control de calidad en la facultad.

Referencias bibliográficas.

Añasco, Juan Carlos. [En línea] 2007. http://www.pilar.com.ar/industrias/temasgenerales/normas.htm#MARCO%20HISTORICO.

Febles, Ailin. *Case Corporativo para el proceso de control de cambios.* Tesis presentada en opción al título de Máster en Informática Aplicada. Ciudad de la Habana, 2001.

Febles, Ailin. *MConfig.PM, Modelo de referencia para la Gestión de Configuración en la pequeña y mediana empresa de software.* CUJAE, Ciudad de la Habana, 2004.

Febles, Ailin. *UCI, Calidad del software.* Conferencia impartida en el Congreso FEU, UCI. Ciudad de la Habana, 2006.

Garces, Cecilia y Sánchez, Karina. *Sistema automatizado para el control de la calidad en el laboratorio clínico.* Tesis presentada en opción al título de Ingeniero Informático. Ciudad de la Habana, 2005, p3.

García, Joaquín. CMM-CMMI. [En línea] 2005. http://www.ingenierosoftware.com/calidad/cmm-cmmi.php 14/08/2005.

Granja, Juan Carlos. *Métricas, técnicas del software. Auditoría.* Conferencia impartida en la Universidad de Ciencias Informáticas, Cuidad de la Habana, 2007.

Hoyos, Patricia y Gómez, Liliana. *Propuesta sistema neurodifuso de clasificación para optimización de un modeo de evaluación de calidad de productos de software.* 2002.

IEEE. *Computer Dictionary, Computer Society.* 1990.

IEEE. *Metrics.* 1991.

Napal, Irina y Brito, Irina. *Las pruebas de software, su aplicación al Config.CASE.* Tesis presentada en opción al título de Ingeniero Informático, Ciudad de la Habana , 2003.

Pérez, Isabel. *Métricas para el control de proyectos de software.* Tesis presentada en opción al título de Ingeniero Informático. Ciudad de la Habana, 2002, p143.

Pressman, Roger. *Can Internet-based Aplications Be Engineered.* 1998.

Teruel, Alejandro. [En línea] 2001. http://www.ldc.usb.ve/~teruel/ci4713/clases2001/pruebasRep.html#bitacora.

Bibliografía

1. **General Principles of Software Validation**, **Final Guidance for Industry and FDA Staff**. http://www.fda.gov/cdrh/comp/guidance/938.html#_Toc517237936 , Updated 1/11/2002.

2. **JACOBSON, Ivar**; **RUMBAUGH, James, BOOCH, Grady**. "El lenguaje unificado de modelado", 2000.

3. **Other Software QA and Testing Resources**, http://www.softwareqatest.com/qatlnks1.html, última actualización, marzo de 2007.

4. **PRESSMAN, Roger**. "Ingeniería del Software. Un enfoque práctico", 2002.

5. **R. Lyu, Michael**. "Handbook of Software Reliability Engineering", Published by IEEE Computer Society Press and McGraw-Hill Book Company, http://www.cse.cuhk.edu.hk/~lyu/book/reliability/, last update Mon May 23 15:11:10 EST 2005.

6. **Software Testing Glossary**, http://www.aptest.com/glossary.html, Last updated 01/13/2007 21:42:13

7. **Software Testing Types**, http://www.aptest.com/testtypes.html, Last updated Sun, 14 Jan 2007 02:44:04 GMT.

8. **Software QA and Testing Frequently-Asked-Questions, Part 1**, http://www.softwareqatest.com/qatfaq1.html, última actualización, marzo de 2007.

9. **Software QA and Testing Frequently-Asked-Questions Part 2**, http://www.softwareqatest.com/qatfaq2.html, última actualización, marzo de 2007.

10. **Software QA and Testing Less-Frequently-Asked-Questions**, http://www.softwareqatest.com/qat_lfaq1.html, última actualización, marzo de 2007.

11. **Software QA and Testing Tools Info**, http://www.softwareqatest.com/qattls1.html, última actualización, marzo de 2007.

12. **Software QA Testing and Test Tool Resources**, http://www.aptest.com/resources.html, Last updated Fri, 08 Jun 2007 23:06:11 GMT.

13. **Web Site Test Tools and Site Management Tools**, http://www.softwareqatest.com/qatweb1.html, última actualización, marzo de 2007.

ANEXOS

Anexo I.CB-Mod-Nuevo.

Casos de prueba para "Insertar Persona Discapacitado" (Clasificación en relación con el consumo de bebidas alcohólicas).

Camino 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15
Caso de prueba*:* comprobar consumo de alcohol.
Entrada*:* padre=1, madre=2, hermano=3, hijo=4, esposo=1, otro=2, discapacitado=1.
Resultado: Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, todos consumen alcohol, pero de diferentes formas.
Condiciones*:* Que exista el discapacitado.

Camino 1-3-4-5-6-7-8-9-10-11-12-13-14-15
Caso de prueba: La madre no consume alcohol.
Entrada: padre=1, madre=0, hermano=3, hijo=4, esposo=1, otro=2, discapacitado=1.
Resultado*:* Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, todos consumen alcohol de diferentes formas excepto la madre.
Condiciones*:* Que exista el discapacitado.

Camino 1-3-5-6-7-8-9-10-11-12-13-14-15
Caso de prueba*:* No consumen alcohol la madre y el padre.
Entrada*:* padre=0, madre=0, hermano=3, hijo=4, esposo=1, otro=2, discapacitado=1.
Resultado*:* Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, todos consumen alcohol de diferentes formas excepto la madre y el padre.
Condiciones: Que exista el discapacitado.

Camino 1-3-5-7-8-9-10-11-12-13-14-15
Caso de prueba*:* No consumen alcohol la madre, el padre y los hermanos.

Entrada*:* padre=0, madre=0, hermanos=0, hijo=4, esposo=1, otro=2, discapacitado=1.
Resultado: Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, todos consumen alcohol de diferentes formas excepto la madre, el padre y los hermanos.
Condiciones: Que exista el discapacitado.

Camino 1-3-5-7-9-10-11-12-13-14-15
Caso de prueba*:* Consumen alcohol los hijos, otros familiares y el discapacitado.
Entrada*:* padre=0, madre=0, hermano=0, hijos=4, esposo=0, otro=2, discapacitado=1.
Resultado: Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, los hijos, otro familiar y el discapacitado consumen alcohol de diferentes formas.
Condiciones: Que exista el discapacitado.

Camino 1-3-5-7-9-11-12-13-14-15
Caso de prueba*:* Consumen alcohol otros familiares y el discapacitado.
Entrada: padre=0, madre=0, hermano=0, hijo=0, esposo=0, otro=2, discapacitado=1.
Resultado*:* Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, el discapacitado y otros familiares consumen alcohol de diferentes formas.
Condiciones: Que exista el discapacitado.

Camino 1-3-5-7-9-11-13-14-15
Caso de prueba*:* Consume alcohol solo el discapacitado.
Entrada*:* padre=0, madre=0, hermano=0, hijo=0, esposo=0, otro=0, discapacitado=1.
Resultado*:* Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, ninguno de los familiares del discapacitado consume alcohol, solo él lo hace.
Condiciones: Que exista el discapacitado.

Camino 1-3-5-7-9-11-13-15

Caso de prueba: No hay consumo de alcohol.
Entrada: padre=0, madre=0, hermano=0, hijo=0, esposo=0, otro=0, discapacitado=0.
Resultado: Se entran los datos del discapacitado y de los familiares integrantes del núcleo en cuanto a consumo de alcohol. Para este caso, nadie consume alcohol en ninguna modalidad.
Condiciones*:* Que exista el discapacitado.

Casos de prueba para "Insertar tipo de discapacidad."

Camino 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: insertar tipo de discapacidad.
Entrada: tipo de discapacidad (físico motora=1, visual=2, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: El sistema registra el tipo de discapacidad que presenta el paciente. En el presente caso presenta todas las posibles discapacidades.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: no presenta ausencia del miembro 1.
Entrada: tipo de discapacidad (físico motora=1, aus1=0, visual=2, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de un miembro, el 1.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: no presenta ausencia de los miembros: 1 y 2.
Entrada: tipo de discapacidad (físico motora=1, aus1=0, aus2=0, visual=2, auditiva=3, mental=4, visceral=5, habla=6).

Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros 1 y 2.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: no presenta ausencia de los miembros: 2 y 3.
Entrada: tipo de discapacidad (físico motora=1, aus2=0, aus3=0, visual=2, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros 2 y 3.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: no presenta ausencia de los miembros.
Entrada: tipo de discapacidad (físico motora=1, visual=2, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: no presenta ausencia de los miembros, ni parálisis del miembro 1.
Entrada: tipo de discapacidad (físico motora=1, para1=0, visual=2, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros y la parálisis del miembro 1.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41

Caso de prueba: no presenta ausencia de los miembros, ni parálisis de los miembros 1 y 2.

Entrada: tipo de discapacidad (físico motora=1, para1=0, para2=0, visual=2, auditiva=3, mental=4, visceral=5, habla=6).

Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros y la parálisis del miembro 1 y 2.

Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-18-19-20-21-22-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41

Caso de prueba: no presenta ausencia de los miembros, ni parálisis de los miembros 1, 2, 3 y no es ciego del ojo derecho.

Entrada: tipo de discapacidad (físico motora=1, para1=0, para2=0, para3=0, visual=2, ojod=0, auditiva=3, mental=4, visceral=5, habla=6).

Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros y la parálisis del miembro 1,2 y 3, además de no ser ciego del ojo derecho.

Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-18-19-20-21-22-24-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41

Caso de prueba: no presenta ausencia de los miembros, ni parálisis de los miembros 1, 2, 3 y no es ciego del ojo derecho. (Se llega al mismo resultado que el anterior)

Entrada: tipo de discapacidad (físico motora=1, para1=0, para2=0, para3=0, visual=2, ojod=0, auditiva=3, mental=4, visceral=5, habla=6).

Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros y la parálisis del miembro 1,2 y 3, además de no ser ciego del ojo derecho.

Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-19-20-21-22-24-26-27-28-29-30-32-33-34-35-36-37-38-39-40-41

Caso de prueba: no presenta ausencia de los miembros, ni parálisis de los miembros 1, 2, 3 y no es ciego.

Entrada: tipo de discapacidad (físico motora=1, para1=0, para2=0, para3=0, visual=2, auditiva=3, mental =4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto la ausencia de los miembros y la parálisis del miembro 1,2 y 3, además de no ser ciego de ningún ojo.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-19-20-21-22-24-26-27-28-29-34-35-36-37-38-39-40-41
Caso de prueba: no presenta discapacidad visual ni auditiva.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=0, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto visual y auditiva.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-19-20-21-22-24-26-27-28-29-34-35-36-37-38-39-41
Caso de prueba: no presenta discapacidad visual, auditiva ni de habla.
Entrada: tipo de discapacidad (físico motora=1, visual 0, auditiva=0, mental=4, visceral=5, habla=0).
Resultado: el discapacitado presenta todas las posibles discapacidades, excepto visual, auditiva y de habla.
Condiciones: Que exista el discapacitado.

Camino 1-2-19.27-35-37-39-40-41
Caso de prueba: presenta discapacidad físico motora y de habla.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=0, mental=0, visceral=0, habla=6).
Resultado: el discapacitado presenta la discapacidad físico motora y de habla.
Condiciones: Que exista el discapacitado.

Camino 1-2-19.27-35-37-39-41
Caso de prueba: presenta discapacidad físico motora.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=0, mental=0, visceral=0, habla=0).
Resultado: el discapacitado presenta la discapacidad físico motora.
Condiciones: Que exista el discapacitado.

Camino 1-2-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: presenta todas las discapacidades sin especificar las físico motoras.
Entrada: tipo de discapacidad (físico motora=1, visual=2, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las discapacidades, a pesar de no especificar las físico motoras.
Condiciones: Que exista el discapacitado.

Camino 1-2-19-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41
Caso de prueba: no presenta la discapacidad visual ni especifica las físico motoras.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las discapacidades, menos la especificación de las físico motoras y la discapacidad visual.
Condiciones: Que exista el discapacitado.

Camino 1-2-19-27-35-36-37-38-39-40-41
Caso de prueba: presenta la discapacidad visceral, físico motora y de habla.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=0, mental=0, visceral=5, habla=6).
Resultado: el discapacitado presenta las discapacidades de habla, visceral y no especifica las físico motoras.
Condiciones: Que exista el discapacitado.

Camino 1-2-19-27-35-37-38-39-40-41
Caso de prueba: presenta la discapacidad visceral, física motora, mental y de habla.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=0, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta las discapacidades visceral, mental y de habla y no especifica las físico motoras.
Condiciones: Que exista el discapacitado.

Camino 1-2-19-27-28-29-30-31-32-33-34-35-36-37-38-39-41
Caso de prueba: no presenta discapacidad visual ni de habla.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=3, mental=4, visceral=5, habla=0).

Resultado: el discapacitado presenta todas las discapacidades, excepto la visual, de habla y la especificación de las físico motoras.
Condiciones: Que exista el discapacitado.

Camino 1-2-19-20-21-22-24-26-27-28-34-35-36-37-38-39-40-41
Caso de prueba: no presenta discapacidad visual ni especifica físico motora.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=3, mental=4, visceral=5, habla=6).
Resultado: el discapacitado presenta todas las discapacidades, menos la especificación de las físico motoras y la discapacidad visual.
Condiciones: Que exista el discapacitado.

Camino : 1-2-19-20-21-22-24-26-27-28-29-30-32-33-34-35-36-37-38-39-41
Caso de prueba: no presenta discapacidad visual ni de habla.
Entrada: tipo de discapacidad (físico motora=1, visual=0, auditiva=3, oidoi=1, mental=4, visceral=5, habla=0).
Resultado: el discapacitado presenta físico motoras sin especificar cuales, sordo del oído izquierdo, las discapacidades visceral y mental.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-18-19-27-35-37-39-40-41
Caso de prueba: presenta parálisis del miembro 4 y discapacidad del habla.
Entrada: tipo de discapacidad (físico motora=1, para4=1, visual=0, auditiva=0, mental=0, visceral=0, habla=6).
Resultado: el discapacitado presenta físico motoras presentando parálisis del miembro 4, y discapacidad del habla.
Condiciones: Que exista el discapacitado.

Camino 1-2-3-5-7-9-11-13-15-17-18-19-27-35-37-39-41
Caso de prueba: presenta parálisis del miembro 4.
Entrada: tipo de discapacidad (físico motora=1, para4=1, visual=0, auditiva=0, mental=0, visceral=0, habla=0).

Resultado: el discapacitado presenta físico motoras presentando parálisis del miembro 4.
Condiciones: Que exista el discapacitado.

Casos de prueba para "function Sordo".

Camino 1-2-4
Caso de prueba*:* activar sordo.
Entrada*:* formulario.sordosi.checked = true.
Resultado: Se activa el formulario sordo para seleccionar el oído con problemas del discapacitado.
Condiciones: Que exista el discapacitado.

Camino 1-3-4
Caso de prueba*:* no es sordo.
Entrada*:* formulario.sordosi.checked = false.
Resultado: no se activa el formulario sordo.
Condiciones*:* Que exista el discapacitado.

Anexo II.CB-Mod-Reportes.

Casos de prueba para "Crear amparo filial nivel provincial".

Camino 1-2-3-4-3-4-5-6-8-9-10-11-12-13-15
Caso de prueba*:* reporte.
Entrada: iddere (dere== "provincial").
Resultado*:* El sistema muestra la cantidad de discapacitados con y sin amparo filial a nivel provincial. Para este caso se realiza el cálculo por municipio y consejo popular.
Condiciones: Que existan discapacitados registrados.

Camino 1-2-3-4-5-6-8-9-10-11-12-13-15
Caso de prueba: reporte provincial.

Entrada*:* iddere (dere== “provincial”).
Resultado: El sistema muestra la cantidad de discapacitados con y sin amparo filial a nivel provincial. Para este caso se realiza el cálculo por municipio y consejo popular.
Condiciones*:* Que existan discapacitados registrados.

Camino 1-2-3-4-5-7-8-9-10-11-12-14-15
Caso de prueba*:* provincial.
Entrada*:* iddere (dere== “provincial”).
Resultado: El sistema muestra la cantidad de discapacitados con y sin amparo filial a nivel provincial. Para este caso se realiza el cálculo por municipio y consejo popular, mostrando el por ciento del municipio y de los consejos populares.
Condiciones: Que existan discapacitados registrados.

Camino 1-2-3-4-5-6-8-9-10-11-10-11-12-13-15
Caso de prueba*:* cálculo provincial.
Entrada*:* iddere (dere== “provincial”).
Resultado*:* El sistema muestra la cantidad de discapacitados con y sin amparo filial a nivel provincial. Para este caso se realiza el cálculo por municipio y consejo popular.
Condiciones*:* Que existan discapacitados registrados.

Camino 1-2-3-4-5-6-8-9-10-11-12-14-15
Caso de prueba*:* reporte: por ciento por consejo popular.
Entrada: iddere (dere== “provincial”).
Resultado: El sistema muestra la cantidad de discapacitados con y sin amparo filial a nivel provincial. Para este caso se realiza el cálculo por municipio y consejo popular, mostrando solo el por ciento de los consejos populares.
Condiciones: Que existan discapacitados registrados.

GLOSARIO

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

Camino Independiente*:* es aquel que introduce por lo menos una sentencia de procesamiento (o condición) que no estaba considerada en el conjunto de caminos independientes calculados hasta ese momento.

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.

Defecto: Un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa.

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

Artefacto: Pieza de información tangible que es creada, modificada y usada por los trabajadores al realizar las actividades; representa un área de responsabilidad y es candidata a ser tenida en cuenta para el control de la configuración. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento.

Desarrollador: Persona que trabaja directamente en el desarrollo de un proyecto de software, dígase: analista, programador, diseñador, etc.

Fase: Son los pasos en que se descomponen las metodologías. Cada fase puede o no estar subordinada a otra fase, pudiendo existir entre ellas relaciones de dependencia de inicio, fin y paralelismo.

Grupo de desarrollo: Dos personas o más que trabajan para lograr una meta, objetivo o misión común, donde cada individuo tiene asignado un rol específico y donde el completamiento de la misión depende de los miembros del equipo.

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.

Metodología: Es un conjunto de procedimientos, técnicas, instrumentos y documentos para ayudar a los analistas y programadores a obtener un nuevo sistema informático. Consiste en fases que guían al diseñador en la elección de las técnicas más apropiadas en cada paso del proyecto, a planificar, dirigir, controlar y evaluar el mismo.

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

Usuario: Persona que utiliza normalmente el *software*.

Iteración: Es un ciclo repetitivo de un proceso.