UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

Diseño y aplicación de pruebas al producto “Árbol Genealógico.”

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

AUTORA: Eliane Paz Noa

TUTORES: Ing. Reynier García Vistorte

Lic. Yosdenis Urrutia Badillo

Ciudad de la Habana, julio del 2007.
”Año 49 de la Revolución”

DECLARACIÓN DE AUTORÍA.

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

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

Eliane Paz Noa	Ing. Reynier García Vistorte	Lic. Yosdenis Urrutia Badillo
_____	_____	_____
Firma del autor	Firma del tutor	Firma del tutor

DATOS DE CONTACTO.

Tutores: Ing. Reynier García Vistorte.

Universidad de las Ciencia Informáticas, La Habana, Cuba.

reyniergv@uci.cu.

Lic. Yosdenis Urrutia Badillo.

Universidad de las Ciencia Informáticas, La Habana, Cuba.

yosdenis@uci.cu.

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA.

Título: Diseño y aplicación de pruebas al producto “Árbol Genealógico.”
Autora: Eliane Paz Noa.
Fecha: Julio 2007.

Los tutores del presente Trabajo de Diploma consideramos que durante su ejecución la estudiante mostró las cualidades que a continuación se detallan:

La estudiante manifestó una actitud disciplinada, independiente y responsable, mucha dedicación al trabajo, gestionando información y enfocada 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, con ello su calidad frente al cliente.

Por todo lo anteriormente expresado consideramos que la estudiante está apta para ejercer como Ingeniero en Ciencias Informáticas; y proponemos que se le otorgue al Trabajo de Diploma la calificación de 5-Excelente. Además, consideramos que los resultados poseen valor para ser publicados.

Ing. Reynier García Vistorte

_____

Firma del tutor

Lic. Yosdenis Urrutia Badillo

_____

Firma del tutor

PENSAMIENTO

"El éxito de los hombres no se mide por su éxito inmediato, sino por su éxito definitivo, no se mide por el dinero que acumularon, sino por el resultado de sus obras".

José Julián Martí Pérez.

AGRADECIMIENTOS

No cabe duda de que una tesis es un punto culminante en el mundo académico de cualquier estudiante, pero también es cierto que no deja de ser un eslabón más en la cadena de la vida, sin duda alguna esta tesis representa un punto crucial de una etapa muy. En toda la experiencia universitaria y la conclusión del trabajo de tesis, han existido personas que merecen las gracias porque sin su valiosa aportación no hubiera sido posible este trabajo y también hay quienes la merecen por haber plasmado sus huellas en mi camino.

Deseo agradecer profundamente a la casualidad que la vida me otorgó al haberme puesto en un hogar maravilloso al nacer. Sin el apoyo en todo sentido de mis padres y familiares, el placer cotidiano de vivir sería simple monotonía. Es difícil imaginar cómo sería el andar cotidiano sin recordar su comprensión, su apoyo inmenso y su amor.

A mis padres, hermana, tías y abuelas por compartir y dedicar gran parte de sus vidas conmigo y por darme aliento para la ardua tarea de caminar hacia la perspectiva de un nuevo día que sin dudas serán inolvidables.

A mi novio Eduardo por haberme ayudado infinitamente, ayudándome en todo momento incondicionalmente.

A mis amigos con quienes he compartido momentos increíbles que siempre llevaré en mi corazón, quienes han enriquecido mi vida con su cariño y alegría. Gracias por recordarme que hay personas valiosas en el mundo y gracias por estar en el mío.

A mis profesores, que compartieron conmigo sus conocimientos y su amor por la informática, especialmente a los compañeros Alieski Sarmiento Almenares y Aurelio Antelo Collado por brindarme todo su apoyo en la realización de esta tesis.

A mis queridos tutores Yosdenis Urrutia Badillo y Reynier García Vistorte por el apoyo tan grande y la amistad que me brindaron. Ustedes han hecho posible que hoy termine mi carrera en la Universidad de las Ciencias Informáticas, gracias por confiar en mí.

A la Revolución cubana por haber inculcado tantos principios y valores en nosotros los jóvenes y haber creado esta hermosa cuidad universitaria.

A nuestro invicto Comandante en Jefe Fidel Castro Ruz, gracias por enseñarnos a conservar la libertad con la que hoy contamos.

Muchas Gracias a todos…

DEDICATORIA

A mis padres quienes me infundaron la ética y el rigor que guían mi transitar por la vida.

RESUMEN

La investigación surge debido a que en la Universidad de las Ciencias Informáticas (UCI), a pesar de que se ha comenzado a trabajar con fuerza en los temas referentes a la calidad del software en los últimos años no se tiene ninguna experiencia en haber probado ningún software del perfil de bioinformática por lo que se hace imprescindible realizar las pruebas para el software "Árbol Genealógico" de la facultad # 6, centrándose la investigación fundamentalmente en la confección de un plan de prueba además del diseño y aplicación de pruebas de caja blanca (pruebas del código) y caja negra al software, para lograr eliminar la mayor cantidad de errores posibles y así entregar un producto con calidad. Se realizan casos de pruebas para cada tipo de prueba, obteniendo resultados y siendo estos comparados con los previstos.

PALABRAS CLAVES

- ✓ Pruebas
- ✓ Caja Negra
- ✓ Caja Blanca
- ✓ Bioinformática
- ✓ Casos de Prueba
- ✓ Calidad de software

INTRODUCCIÓN

El software se ha convertido en un tema crítico en la sociedad moderna mundial. Todos parecen necesitar mejores software en el menor tiempo posible y a menor costo. Los métodos intuitivos de desarrollo de software que se usan actualmente son, básicamente, aquellos que los propios individuos “artesanalmente” siguen, los cuales sólo servirán mientras la sociedad pueda tolerar la falta de predicción que ellos acarrean, es por ello que se hace cada vez más imponente el tema de la calidad de los productos de software que se producen en todo el mundo.

Existe una tendencia internacional a la producción de software avalado por sistemas de calidad, pero esta práctica se puede decir que no está generalizada en todos los países. En el caso de Cuba, atendiendo al nivel de desarrollo propio, se trabaja en una fase “artesanal” dónde aún no se definen procedimientos para medir los estándares de calidad.

La coyuntura económica de nuestro país favorece en este momento el desarrollo de sistemas, no solamente para consumo interno de una organización sino también para su comercialización a terceros e incluso para la exportación. La evolución permanente de la tecnología informática, y consecuentemente, de los principios y técnicas de desarrollo de sistemas, impacta constantemente la formación de los profesionales en informática, entre ellos, quienes construyen software.

La Calidad del Software consiste en desarrollar productos lógicos que, cumpliendo con las normas establecidas, satisfagan las necesidades del usuario, los requisitos implícitos, y que tiendan a cero defectos. Es todavía una disciplina minoritaria, aunque todo el mundo hable de ella y se crea capacitado para opinar, como ocurre con la política, la religión o la filosofía.

Una característica esencial del software de calidad es su capacidad para ser reutilizado. El problema de ámbito universal más caro de la historia tiene que ver con el software, con la falta de calidad y con la imposibilidad de la reutilización del software por su incapacidad de adaptarse a circunstancias que sobrevienen de manera inevitable.

La calidad del software es uno de los puntos de atención de las organizaciones actuales, ya que el software se ha convertido en un activo que determina en gran medida la operatividad de la organización.

El desarrollo vertiginoso de algunas áreas de las ciencias biológicas ha obligado al desarrollo de otros campos como el de la bioinformática de manera que ha sido necesario a su vez el desarrollo de software que facilite el trabajo de científicos que incursionan en este campo de la ciencia. Actualmente la mayor parte del software que se usa para la bioinformática es desarrollado por universidades que distribuyen gratuitamente estos programas a instituciones que lo necesiten para el desarrollo de sus investigaciones.

En los últimos años la empresa del software ha tomado gran auge a nivel mundial, de modo que nuestro país ha querido incursionar de manera más estable en este fascinante mundo del desarrollo de software con la gran idea de que en un futuro se cuente en el país con la gran Industria Cubana de Software.

El mercado del software que existe hoy funciona extremadamente bien generando soluciones innovadoras para todas las esferas de negocios. Actualmente el tema de la calidad del software tanto en Cuba, como en nuestra Universidad, esta dando sus primeros pasos, de modo que falta todavía mucho por hacer, por esta razón surge la necesidad de realizar una investigación para afianzar poco a poco nuestros conocimientos en cuanto a este aspecto tan importante que tiene que ver con el desarrollo del software.

Se ha presentado entonces la **situación problemática** de que La Universidad de las Ciencias Informáticas, como estrategia del programa de informatización del país, es uno de los centros de desarrollo del talento humano en esta área, aunque también se concibe como una entidad de producción al asimilar proyectos de software en distintos sectores del país. La facultad 6 cuyo perfil va encaminado a la Bioinformática está desarrollando proyectos con instituciones prestigiosas de nuestro país como el CIGB (Centro de Ingeniería Genética y Biotecnología) y el CIM (Centro de Inmunología Molecular).

Para la comercialización de estos software como productos de mercado, tanto nacional como internacional, es necesario que salgan con excelente calidad, no obstante, en la Universidad no se tiene ninguna experiencia en haber probado ningún software de bioinformática por lo que se hace imprescindible realizar el diseño y la aplicación de los casos de prueba para el software “Árbol Genealógico” de nuestra facultad.

Entonces ¿Cómo validar la calidad del producto “Árbol Genealógico”? daría paso al **problema** que nos conduce a realizar esta investigación.

El **Objeto de estudio** planteado es: El diseño y aplicación de prueba para validar la calidad de software.

El **Campo de acción:** El diseño y aplicación de pruebas de calidad al producto “Árbol Genealógico”.

Como **objetivo general** del trabajo tenemos:

- ✓ *Diseñar y Aplicar* las pruebas de software al producto “Árbol Genealógico”.

Como **objetivos específicos** del trabajo tenemos:

- ✓ *Desarrollar* los casos de pruebas de calidad para el producto “Árbol Genealógico”.
- ✓ *Validar* mediante la aplicación de las pruebas, el diseño de prueba que se proponen para el producto “Árbol Genealógico”.
- ✓ *Fundamentar* las pruebas de software especificando la estrategia a seguir a la hora de aplicarlas y cómo se debe hacer su planificación.
- ✓ *Exponer* los resultados obtenidos de las pruebas realizadas.

Para el desarrollo de la investigación y dar cumplimiento a los objetivos propuestos se llevaron a cabo una serie de **tareas y técnicas** entre las cuales podemos mencionar:

- ✓ *Investigación* acerca de las pruebas de software que tienen mayor aplicabilidad, cómo funcionan y los pasos a seguir para su correcta puesta en práctica.
- ✓ *Confección* del plan de prueba, así como toda la documentación que lleva.
- ✓ *Interpretación* del tema no sólo desde el punto de vista informático, sino también desde su perfil biológico para comprender la funcionalidad deseada con su puesta en práctica.
- ✓ *Estudio* de las actividades del Flujo de trabajo de prueba de RUP para su mejor entendimiento.
- ✓ *Aplicación* de las pruebas escogidas al software “Árbol Genealógico”.
- ✓ *Comparación* de los resultados obtenidos con los esperados.

La tesis consta de 3 Capítulos:

Capítulo 1: Fundamentación teórica. Donde se expone un basamento teórico y conceptual acerca de la situación actual de la calidad de software, además de las características fundamentales de las pruebas que se llevan a cabo para garantizar un exitoso despliegue del sistema informático que se vaya a implantar.

Capítulo 2: Diseño de las pruebas. En este capítulo se presentará el diseño de las pruebas que serán aplicadas al software "Árbol Genealógico", partiendo del plan de prueba que se desarrolla, mostrando además una descripción completa del producto al cual se la aplicarán las pruebas.

Capítulo 3: Resultados. En este capítulo se exponen de manera clara y entendible un resumen de todos los fallos que se detectaron a la hora de aplicar las pruebas de caja blanca y caja negra al software Árbol Genealógico así como las recomendaciones que se hacen al respecto.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

La calidad de un software no sólo se logra con un buen diseño del sistema o con un bajo nivel de riesgo. Para asegurar la calidad es necesario además, revisar la documentación asociada al software con el objetivo de verificar su cobertura, corrección, confiabilidad, facilidad de mantenimiento entre otros aspectos relevantes que son claves para medir la calidad de un software. De esta forma se abordarán en este capítulo aspectos importantes referentes a la calidad del software así como las características fundamentales de las pruebas que se llevan a cabo para garantizar un exitoso despliegue del sistema informático que se vaya a implantar.

1.2 Calidad de Software.

Uno de los problemas que se han afrontado desde los inicios en la esfera de la computación es la calidad del software. A finales de los sesenta, cada vez que se quería afrontar la mejora del software todo se dirigía a la mejora del código dejando olvidado todo lo que le rodeaba, ya en la década del setenta, este tema fue motivo de preocupación para especialistas, ingenieros, investigadores y comercializadores de software, los que fueron mejorando los procesos que se habían definido hasta el momento con el objetivo de mejorar la calidad que seguía siendo aún un tema crítico en el mercado del software, ya en los años ochenta se comenzaron a tener en cuenta los aspectos de especificaciones, diseño, evaluación y gestión del software. Pese a estos cambios realizados, en la actualidad persiste la problemática, ya que el nivel de complejidad del software va aumentando y no se han alcanzado los niveles de calidad deseados.

Para nadie es un secreto que la calidad es sinónimo de eficiencia, flexibilidad, corrección, portabilidad, usabilidad, seguridad, integridad, etc. La calidad del software es medible y varía de un sistema a otro o de un programa a otro. Es posible medir los principales atributos que forman o caracterizan a un software de buena calidad. La idea es que la calidad del software se caracteriza por ciertos atributos: fiabilidad, flexibilidad, robustez, comprensión, adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reutilización, eficiencia, y sin lugar a dudas es posible medir cada uno de ellos, y por consiguiente, caracterizar o medir la calidad del software en cuestión. Para cada atributo a medir, hay una medición confiable (objetiva) que puede llevarse a cabo. La idea es que, dado que el atributo deseable es difícil de medir, se mide otro atributo que está correlacionado con el primero, por ejemplo:

✓ La fiabilidad (software confiable, pocos errores) se mide a través del número de mensajes de error, mientras más mensajes existan, contiene entonces menos errores este software.

✓ La flexibilidad (capacidad de adaptación a diferentes tipos de uso, a diferentes ambientes de uso) o adaptabilidad.

✓ La robustez (pocas fallas catastróficas, el sistema “no se cae”) se mide a través de pruebas y uso prolongado.

✓ La comprensión (capacidad de entender lo que el sistema hace) se mide viendo la cantidad de comentarios que posee el software, y la extensión de sus manuales de usuarios.

✓ El tamaño se mide en bytes, o sea el espacio que ocupa en memoria, (esta medición no tiene objeción, se mide lo que se quiere medir).

✓ La eficiencia de un programa se mide en segundos, es la rapidez en su ejecución (esta medición no tiene objeción, se mide lo que se quiere medir).

✓ La modularidad de un programa se mide contando el número de módulos que lo conforman.

✓ La complejidad de un programa se mide contando el número de anidaciones en expresiones o postulados, (es lo que se le llama complejidad ciclomática).

✓ La portabilidad es la facilidad con que se eche a andar en otro sistema operativo distinto al que fue creado. Se mide preguntando a usuarios que han hecho estos trabajos.

✓ La usabilidad de un programa es alta cuando el programa aporta gran valor agregado a nuestro trabajo. “Es indispensable contar con él”. Se mide viendo que porcentaje de nuestras necesidades cubre ese programa.

✓ La reutilización de un programa se mide por la cantidad de veces que partes del mismo se han reutilizado en otros proyectos de desarrollo de software.

✓ La facilidad de uso (ergonomía) caracteriza a los programas que no cuesta trabajo aprender, que se amoldan al modo intuitivo de hacer las cosas. Se mide observando la cantidad de pantallas que interaccionan con el usuario, y su sofisticación.

Podemos entonces llegar a la conclusión que en vez de medir la calidad del producto, podemos medir la calidad del proceso. Tener un buen proceso implica producir software de buena calidad. El problema es que no se sabe cómo deducir cuál proceso producirá buena calidad en el software. En ocasiones se recurre a procesos que suenan o se ven razonables, o que han sido ensayados en otros lados con éxito, o que están dados por algún estándar o comité internacional, sin embargo mientras el proceso que se lleve a cabo sea adaptable a nuestro proyecto y sea totalmente fiable entonces obtendremos un software de calidad.

La obtención de un software con calidad implica la utilización de metodologías o procedimientos, estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software. La innovación tecnológica y la adecuación a las nuevas tecnologías plantean el gran reto futuro. Pese a lo arriesgado de hacer predicciones, está claro que la medición se plantea como uno de los principales campos de investigación en Calidad del Software.

La especificación y evaluación integral de la calidad de los productos de software es un factor clave para asegurar que la calidad sea la adecuada. Esto se puede lograr definiendo de manera apropiada las características de calidad, teniendo en cuenta el propósito del uso del producto de software. Es importante especificar y evaluar cada característica relevante de la calidad de los productos de software, mientras esto sea posible, utilizando mediciones validadas o de amplia aceptación, que hagan técnicamente transparente esta actividad.

1.3 Conceptos y Definiciones.

Todos dicen poseer y dominar la calidad de software, pero cómo conocer y dominar un concepto tan amplio, subjetivo y muchas veces ambiguo, donde ni siquiera la mayoría de los investigadores que han hecho aportes considerables en este campo han podido, ya que cada cual ve la calidad desde su punto de vista, es por ello que dar una definición exacta de ¿qué es calidad? es un tarea bien difícil.

No existe entonces una definición estándar ni universal de que es la calidad de software pero algunas instituciones y organismos como ISO, IEEE, SEI, brindan definiciones aceptables pero por su puesto que éstas no son homogéneas, dando como resultado que cada profesional o institución utilice su propia versión de calidad, sin embargo expondremos algunas definiciones que personalidades importantísimas han popularizado por todo el mundo, por ejemplo:

✓ *La calidad del software esta dada por la calidad de los procesos usados para desarrollarlo y mantenerlo,* ***Watts S. Humphrey.*** (1)

✓ ***Juran (1970)*** *definió la calidad del software como" adaptabilidad de uso" la cual implica dos parámetros fundamentales: calidad de diseño y calidad de conformidad, de manera que sea adaptable a los todos los usuarios. (1)*

✓ ***Crosby (1979)*** *definió la calidad del software como la" conformidad con los requerimientos". (1)*

✓ *"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente"* ***R.S. Pressman (1992).*** *(2)*

Luego de haber analizado los conceptos dados por grandes personalidades de la ingeniería y la calidad del software se puede plantear que obtener un software con calidad implica velar fielmente por que el proceso de desarrollo de software tenga la claridad y calidad requeridas, desde el inicio hasta el fin. El producto debe cumplir con todos los requerimientos ya sean funcionales o no funcionales que fueron analizados en conjunto con los clientes, además de regirse por las normas internacionales de calidad ya establecidas.

1.4 Modelos de Calidad del Producto.

Existen en el mundo infinidades de modelos de calidad por los cuales se rigen todas las empresas productoras de software para poder vender sus productos, en nuestro país las empresas que no están certificadas para poder utilizar estas normas y vender los productos de software en el mercado mundial, pero muchas empresas del mundo las utilizan y obtienen resultados satisfactorios.

A fines de la década del ochenta e inicios de la década del noventa se ha puesto mucho énfasis en los conceptos de calidad de producto y satisfacción del usuario desde diversos enfoques, y particularmente a la valoración y certificación de la calidad de procesos con los bien conocidos CMMI y SPICE (entre otros), sin embargo, también es conocido que los modelos de calidad ya eran reconocidos en la comunidad científica a fines de la década del 70 como los descriptos por McCall y Boehm.

Estos modelos describen a la calidad del producto usando un enfoque de descomposición. El modelo de McCall, por ejemplo fue originalmente desarrollado para la Fuerza Aérea de los EE.UU. y se promocionó su uso para evaluar la calidad del software dentro del DoD (Departamento de defensa de los EE.UU.). En estos modelos, los evaluadores se concentran en los atributos de calidad claves para el producto de software, en consideración de un punto de vista de usuario.

Dada la naturaleza lógica del producto, se asume que la calidad de un sistema de software depende sobremanera de la calidad del proceso usado para desarrollarlo. Los modelos de evaluación y mejora

de procesos y su estandarización, han tomado un papel determinante en la identificación, integración, medición y optimización de las buenas prácticas existentes en la organización y desarrollo del software.

Han surgido continuamente con el desarrollo de la informática y las comunicaciones una serie de herramientas, técnicas y modelos que facilitan a las organizaciones, encargadas de las tecnologías de la información, generar productos que cumplan las expectativas del cliente e incluso las rebasen, herramientas que prometen ser la solución a los problemas de calidad, costo y tiempos de desarrollo.

Otras normas, directivas, modelos o estándares más usados en la actualidad son básicamente las siguientes:

Familia ISO 9000

ISO 9000 es un conjunto de estándares internacionales para sistemas de calidad. Diseñado para la gestión y aseguramiento de la calidad, especifica los requisitos básicos para el desarrollo, producción, instalación y servicio a nivel de sistema y a nivel de producto.

CMMI

El modelo CMMI constituye un marco de referencia de la capacidad de las organizaciones de desarrollo de software en el desempeño de sus diferentes procesos, proporcionando una base para la evaluación de la madurez de las mismas y una guía para implementar una estrategia para la mejora continua de los mismos.

SPICE

SPICE es un emergente estándar internacional de evaluación y determinación de la capacidad y mejora continua de procesos de ingeniería del software, con la filosofía de desarrollar un conjunto de medidas de capacidad estructuradas para todos los procesos del ciclo de vida y para todos los participantes. Es el resultado de un esfuerzo internacional de trabajo y colaboración y tiene la innovación, en comparación con otros modelos, del proceso paralelo de evaluación empírica del resultado.

De acuerdo con la investigación realizada se puede apreciar que el estándar de la familia ISO y CMMI son los más populares en el mundo de la ingeniería del software, seguido por SPICE.

1.5 Actualidad en la calidad de SW. Cuba y la Universidad de las Ciencias Informáticas.

Hoy en día las pequeñas y medianas empresas son una pieza muy importante en el engranaje de la economía mundial. En las últimas dos décadas la industria del software ha emergido, crecido y fortalecido a tal punto que representa actualmente una actividad económica de suma importancia para todos los países del mundo. La industria del software en la mayoría de los países esta formada por tejido industrial compuesto en gran parte por desarrolladoras de software que favorecen al crecimiento de las economías nacionales. *La mayoría de empresas desarrolladoras de software son pequeñas (tienen menos de 50 empleados) y desarrollan productos significativos que, para su construcción, necesitan prácticas eficientes de Ingeniería del Software adaptadas a su tamaño y tipo de negocio.* (3)

Países como Argentina ofrecen condiciones para convertirse en un productor de software en el mercado mundial, muchas son las investigaciones realizadas respecto al tema y muchos son los resultados alcanzados. Las Universidades juegan un papel protagónico en este sentido por ejemplo *la Universidad Austral (UA) de argentina creó en octubre del 2006 un laboratorio de calidad del software con el propósito de realizar un aporte diferencial y ayudar así a que la industria nacional del software sea respetada en todo el mundo.* (4)

La industria del software hoy presenta tres graves problemas: costo de proyectos superiores a los estimados, duración de proyectos también superiores e insatisfacciones de las expectativas de los clientes, y la verdadera causa de estos inconvenientes es que todavía se considera el desarrollo del software como una tarea artesanal en vez de una ingeniería, de manera que toda industria para ser efectiva y eficiente necesita una ingeniería que la soporte y a su vez esta ingeniería trabaja con modelos de calidad, la clave del éxito no radica en crear nuevos modelos, que ya existen en abundancia, sino saber ponerlos en práctica.

Según los estudios realizados, se ha podido llegar a la conclusión que las empresas europeas aún tienen mucho camino por recorrer en lo que a la práctica de utilización de metodologías para medir la calidad de su software se refiere. No obstante, las empresas españolas se sitúan a la cabeza de los países europeos que siguen un buen modelo de calidad. Más de la mitad de las empresas hoy, fallan a la hora de aplicar una metodología para asegurar la calidad del software que producen.

Entre algunas de las principales causas de estos fracasos se destaca el hecho de que las organizaciones admiten que los equipos de desarrollo que tienen destinados a trabajar en la calidad

del software carecen de formación y experiencia y sin embargo no emplean recursos para la capacitación de este personal, sin darse cuenta que el costo por los errores cometidos puede ser mayor.

Con el objetivo de lograr la informatización de la sociedad cubana, en el país se está trabajando fehacientemente en el desarrollo de la Industria Cubana del Software, que nos permita proveernos de sistemas de información no sólo en beneficio de la sociedad cubana, sino también que podamos exportar nuestros productos y que sean reconocidos internacionalmente, sin embargo *las empresas que producen software hoy requieren aún de un sistema que cree un entorno organizado donde sean aplicados procedimientos de ingeniería de software que guíen el control de los procesos para desarrollar y mantener el software con una total calidad, para lograr evolucionar hacia una cultura de ingeniería de software y de administración de excelencia.* (5)

Dentro de esta esfera de la producción de software se enmarca nuestra Universidad, la cual fue creada con dos objetivos básicos, la informatización de la sociedad y la exportación. Países como Venezuela se ha beneficiado ya con los logros obtenidos, podemos plantear entonces sin temor a equivocarnos que aún se están limando detalles en lo que a calidad de software se refiere.

En la Universidad de las Ciencias Informáticas (UCI) se creó un grupo de trabajo que es el encargado de garantizar la calidad de los productos de software que se han y están desarrollando, de manera que se puede apreciar que un tema tan importante como la calidad no se ha dejado atrás y de esta forma se han logrado grandes aportes por parte de los estudiantes y profesores.

1.6 Procesos de mejora en el software.

A partir de principios de los años noventa la comunidad de Ingeniería del Software ha expresado especial interés en la mejora de procesos de software, conocida internacionalmente como SPI (Software Process Improvement). Esto se evidencia por el creciente número de artículos que tratan el tema según el análisis de la tendencia de las publicaciones de mejora de proceso presentado, así como por la aparición de un gran número de iniciativas internacionales relacionadas con SPI.

Entre estas iniciativas se encuentran CMM, CMMI, ISO/IEC y SPICE, además de que la norma ISO 9001:2000 está siendo utilizada para ser aplicada en este campo de la mejora de procesos (6). Tanto en las universidades como en las industrias hay una tendencia generalizada a resaltar que los programas de mejora de procesos de software exitosos sólo son posibles para empresas grandes que

cuentan con los recursos suficientes para enmarcarse en este tipo de prácticas. Tal percepción se basa en que estos programas son prohibidos para las pequeñas y medianas empresas debido a la estructura organizacional de las mismas, al costo que los programas de mejora implican ya que los estándares de mejora propuestos internacionalmente por organismos como el SEI (Software Engineering Institute) e ISO (International Organization for Standardization) no han sido creados para este tipo de empresas sino para empresas grandes. *"Casi todos los autores están de acuerdo en que las características especiales de las pequeñas empresas hacen que los programas de mejora de procesos deban aplicarse de un modo particular y visiblemente diferente a como se hace en las grandes organizaciones y que esto no es tan sencillo como el hecho de considerar dichos programas de mejora versiones a escala de las grandes compañías".* (7)

Es importante resaltar que en un programa de mejora se involucran diferentes tipos de modelos y métodos, entre los que se encuentra el modelo que conduce la mejora, el método de evaluación de procesos y el modelo de procesos a seguir. El modelo de mejora describe la infraestructura, actividades, ciclo de vida y consideraciones prácticas para la evolución de los procesos. El método de evaluación de procesos especifica la ejecución de la evaluación para producir un resultado cuantitativo que caracterice la capacidad del proceso o la madurez de la organización. El modelo de procesos de referencia describe cuáles son reconocidas como las mejores prácticas que una organización debe implementar para el desarrollo de software.

*La adaptación del modelo IDEAL es la forma más utilizada para conducir la mejora de los procesos siendo éste un modelo de mejora organizacional que sirve como mapa para iniciar, planificar e implementar acciones tendientes a mejorar los procesos (*8), *utilizándose también modelos como MESOPYME el cual se centra en reducir el tiempo y el esfuerzo en la implementación de la mejora de procesos basado en el concepto de paquetes de acción* (9) (10).

1.7 Calidad de producto y el ciclo de vida.

Las vistas de calidad interna, calidad externa y calidad en uso cambian durante el ciclo de vida del software. Por ejemplo, la calidad especificada, como requisito de calidad al comienzo de un ciclo de vida, es mayormente observada desde el punto de vista externo y de usuario, y se diferencia de la calidad del producto intermedio, como la calidad del diseño, la cual es mayormente observada desde el punto de vista interno del desarrollador. Las tecnologías usadas para alcanzar el nivel de calidad necesario, así como la especificación y evaluación de calidad, necesitan soportar estos diversos

puntos de vista. Es necesario definir estas perspectivas y las tecnologías asociadas a la calidad, para manejarla apropiadamente en cada etapa del ciclo de vida.

La meta es alcanzar la calidad necesaria y suficiente para cumplir con las necesidades reales de los usuarios. La norma ISO 8402 define calidad en términos de la habilidad de satisfacer necesidades explícitas (declaradas/descritas/especificadas) e implícitas. Los requisitos de calidad no pueden ser completamente definidos antes de empezar con el diseño. Sin embargo, es necesario entender las necesidades reales del usuario tan al detalle como sea posible, y representarlas en los requerimientos. La meta no es obtener la calidad perfecta, pero sí la calidad necesaria y suficiente para cada contexto específico de uso, cuando el producto sea entregado y utilizado por los usuarios.

Necesidad de Calidad del Usuario.

Puede ser especificada como requerimientos de calidad por las métricas de calidad en uso, por métricas externas y a veces por métricas internas. Estos requerimientos especificados por las métricas, deberían ser usados como criterios cuando un producto es validado. Lograr un producto que satisfaga las necesidades del usuario, normalmente requiere de un enfoque interactivo en el desarrollo de software, con una continua retroalimentación desde la perspectiva del usuario.

Requerimientos de Calidad Externos.

Especifican el nivel de calidad requerido desde una perspectiva externa. Estos incluyen requerimientos derivados de las necesidades de calidad de usuarios, incluyendo calidad en requerimientos de uso. Los requerimientos de calidad externos son usados como los objetivos para la validación en varias etapas de desarrollo. Los requerimientos de calidad externos para todas las características de calidad definidas en esta parte, deben ser establecidos en la especificación de requerimientos de calidad usando métricas externas, deben ser transformados en requerimientos de calidad internos y deben ser usados como criterios cuando un producto es evaluado.

Requerimientos de Calidad Internos.

Especifican el nivel de calidad requerido desde la perspectiva interna del producto. Los requerimientos de calidad internos son usados para especificar propiedades internas de productos. Estos pueden incluir modelos estáticos y dinámicos, otros documentos y código fuente. Los requerimientos de calidad internos pueden ser usados como objetivos para la validación en varias etapas de desarrollo. Ellos también pueden ser usados para definir estrategias de desarrollo y criterios de evaluación y verificación durante el desarrollo. Esto puede incluir el uso de métricas adicionales (por ejemplo:

reusabilidad). Los requerimientos específicos de calidad interna deben ser especificados cuantitativamente usando métricas internas.

Calidad Interna.

Es la totalidad de características del producto de software desde una perspectiva interna. La calidad interna es medida y evaluada en base a los requerimientos internos de calidad. Los detalles de la calidad del producto de software pueden ser mejorados durante la implementación, revisión y prueba del código fuente del software, pero la naturaleza fundamental de la calidad del producto de software representada por la calidad interna, permanece sin cambios a menos que sea rediseñado.

Calidad Externa.

Es la totalidad de las características del producto de software desde una perspectiva externa. Es la calidad cuando el software es ejecutado, la cual es típicamente medida y evaluada en un ambiente simulado, con datos simulados y usando métricas externas. Durante las pruebas, muchas fallas serán descubiertas y eliminadas. Sin embargo, algunas fallas todavía pueden permanecer después de las pruebas. Como es difícil corregir la arquitectura del software u otros aspectos fundamentales del diseño del software, el diseño fundamental permanece sin cambios a través de las pruebas.

Calidad en Uso.

Es la perspectiva del usuario de la calidad del producto de software cuando éste es usado en un ambiente específico y en un contexto de uso específico. Ésta mide la extensión en la cual los usuarios pueden conseguir sus metas en un ambiente particular, en vez de medir las propiedades del software en sí mismo. El nivel de calidad en el ambiente del usuario puede ser diferente del ambiente de desarrollo, debido a diferencias entre las necesidades y capacidades de diversos usuarios y diferencias entre hardware y ambientes de soporte. El usuario evalúa sólo aquellos atributos de software que son usados para sus tareas. Algunas veces, los atributos de software especificados por un usuario final durante la fase de análisis de requerimientos, ya no cumplen con los requerimientos del usuario cuando el producto está en uso, debido a cambiantes requerimientos del usuario y a la dificultad de especificar necesidades implícitas.

No es generalmente práctico medir la calidad en el uso para todos los escenarios posibles de las tareas del usuario. Los recursos para la evaluación necesitan ser asignados entre los diversos tipos de medida, dependiente de los objetivos de la institución y de la naturaleza del producto y diseño de procesos.

1.8 El costo de la calidad de software.

Todos sabemos que la calidad tiene un costo, pero ¿cuánto es? Si bien existe una definición estándar de costo de la calidad para el desarrollo de software, esta métrica puede resultar engañosa si no se interpreta en su debido contexto. De hecho, un uso descontextualizado de esta métrica desalienta aumentos de productividad.

Medir el costo de la calidad resulta fundamental en un programa de mejoramiento de calidad. Todos sabemos que la mala calidad sí tiene costos, tanto por el costo a la hora de la corrección como por los problemas que les acarrea a nuestros clientes, que bien podrían dejar de serlo. El costo de la calidad tiene dos componentes fundamentales: lo que invertimos en obtener buena calidad y lo que pagamos por no lograrla. La primera componente es decidida por nosotros y controlada; la segunda no la decidimos sino que se manifiesta en las fallas de nuestro producto.

Si se realiza el ejercicio de cuantificar en costes las actividades requeridas para garantizar la calidad del producto final, se comprueba que se puede incrementar sensiblemente el coste del producto, pero es una percepción engañosa, ya que la realización de estos ejercicios de calidad supondrán a lo largo del ciclo del proyecto un impacto en costes de prevención (planificación del proyecto, revisiones técnicas, equipo de pruebas y formación), costes de evaluación (actividades que se realizan para tener una visión más profunda del producto) y costes de fallos (referido a los fallos una vez que se ha enviado el producto a los clientes y que obligan a nuevas versiones del producto), además *los costes relativos al desarrollo del producto aumentan dependiendo de la fase de detección de los defectos* (11), dependiendo de que se encuentren en la fase de prevención o en la fase de fallos, y dentro de la fase de fallos si el defecto se ha descubierto en una fase de pruebas internas o es relativo a fallos externos. *Hay que considerar que los costes por reparación en casa del cliente serán más altos mientras más clientes finales hayan, y que los costes por corrección de errores serán menores en tanto los productos sean personalizados a un cliente.* (12)

La forma estándar de medir el costo de la calidad en el desarrollo de software es registrar el esfuerzo en horas dedicadas a actividades asociadas a la calidad (13). Esto suele ser muy barato si ya se cuenta con un buen sistema de reporte de horas de proyectos, en ese caso, a cada actividad del proyecto se le asigna una categoría en dependencia al costo de la calidad.

1.9 Metodologías.

1.9.1 MSF.

Esta 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 proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. MSF tiene las siguientes características:

- ✓ **Adaptable:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- ✓ **Escalable:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- ✓ **Flexible:** es utilizada en el ambiente de desarrollo de cualquier cliente.
- ✓ **Tecnología Agnóstica:** porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

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.

1.9.2 XP.

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo y corto equipo. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito el proyecto. La metodología se basa en:

- ✓ **Pruebas Unitarias:** se centran en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca.
- ✓ **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- ✓ **Programación en pares:** una particularidad de esta metodología es que 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. El desarrollo bajo XP tiene características que lo distinguen claramente de otras metodologías:

- ✓ Los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- ✓ Los diseños del software se mantienen sencillos y libres de complejidad o pretensiones excesivas.
- ✓ Se obtiene retroalimentación de usuarios y clientes desde el primer día gracias a las pruebas.
- ✓ El software es liberado en entregas frecuentes tan pronto como sea posible.
- ✓ Los cambios se implementan rápidamente tal y como fueron sugeridos.
- ✓ Las metas en características, tiempos y costos son reajustadas permanentemente en función del avance real obtenido.

Con estas características no es sorprendente que XP sea la metodología más apropiada para un entorno caracterizado por requerimientos cambiantes originados por un mercado fluctuante y los propios avances de la tecnología y los negocios.

1.9.3 RUP.

RUP es el proceso unificado de desarrollo de software, plantea quién hace qué, cuándo y cómo, tiene como objetivos asegurar la producción de software de calidad dentro de plazos y presupuestos predecibles, es dirigido por casos de uso, centrado en la arquitectura, iterativo (mini-proyectos) e incremental (versiones), es actualizado constantemente para tener en cuenta las mejores prácticas y utiliza además UML como lenguaje de modelado.

La metodología RUP se divide en 4 fases el desarrollo del software:

- ✓ **Inicio:** El Objetivo en esta etapa es determinar la visión del proyecto.
- ✓ **Elaboración:** En esta etapa el objetivo es determinar la arquitectura óptima.
- ✓ **Construcción:** En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.
- ✓ **Transmisión:** El objetivo es llegar a obtener el release del proyecto.

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. RUP aumenta la productividad de los desarrolladores mediante acceso a base de conocimiento, plantillas y herramientas, se centra en la

producción y mantenimiento de modelos del sistema más que en producir documentos, pretende implementar las mejores prácticas actuales en la ingeniería de software teniendo en cuenta el desarrollo iterativo del software, la administración de requerimientos, el uso de arquitecturas basadas en componentes, el modelamiento visual del software, la verificación de la calidad del software y el control de cambios.

RUP describe cómo planear y ejecutar las pruebas teniendo en cuenta un grupo de artefactos, actividades y trabajadores. Dentro de los roles que propone se encuentra el de ingeniero de prueba el cual cumple con ciertas responsabilidades por ejemplo, debe trabajar con el equipo de desarrollo para validar los requerimientos en aras de resolver las incidencias del diseño y los defectos del software, establecer los juegos de prueba basándose en los requerimientos del cliente y el diseño, ejecutar planes de prueba en los entornos de integración y sistema de los nuevos release y de regresión para asegurar la integridad de la aplicación, ejecutar los planes de prueba detallados, dirigir el análisis detallado de los resultados de las pruebas tanto correspondientes a pruebas manuales como a pruebas automáticas, colaborar con el equipo de trabajo para rectificar los errores encontrados, llevar a cabo pruebas de integración, funcionales, de aceptación y de interfaz de usuario, debe tener además 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.

1.10 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, pero específicamente en el presente trabajo se generan los artefactos de caso de prueba, plan de prueba y evaluación de las pruebas.

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. (14)

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. (14)

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. (14)

Artefacto procedimiento de prueba.

Especifica como realizar uno o varios casos de prueba o partes de estos, por ejemplo 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. (14)

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. (14)

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. (14)

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. (14)

1.10.1 Actividades fundamentales del flujo de trabajo de prueba.

Las actividades fundamentales que se desarrollan en el flujo de trabajo de pruebas son las siguientes:

- ✓ Planificar las pruebas.
- ✓ Diseñar las pruebas: Dentro de esta actividad se realizan pruebas de integración, de sistema y de regresión.
- ✓ Implementar prueba.
- ✓ Realizar pruebas de integración.
- ✓ Realizar pruebas de sistema.
- ✓ Evaluar pruebas.

1.11 Pruebas de software.

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

El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado, además, de que esta etapa implica la verificación de la interacción de componentes, integración adecuada de los componentes, verificación de que todos los requisitos se han implementado correctamente, identificación y aseguramiento de que los defectos encontrados se han corregido antes de entregar el software al cliente, diseño de pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida del proyecto ya que desde la fase de inicio se puede hacer una parte de la planificación de las pruebas probando la funcionalidad de los primeros prototipos; la estabilidad, cobertura y rendimiento de la arquitectura; sin embargo la realización de las pruebas se centra fundamentalmente en la fase de elaboración cuando está planteada ya la línea base de la arquitectura y en la fase de construcción cuando prácticamente el software está implementado, ya en la fase de transición el eje fundamental de las pruebas va dirigido a la corrección de los defectos.

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas, es un proceso de ejecución de un programa con la intención de descubrir un error. *Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces.* (15) Una prueba tiene éxito si descubre un error no detectado hasta entonces. La prueba no puede asegurar la ausencia de defectos; sólo puede demostrar que existen defectos en el software.

1.11.1 Tipos de pruebas de software.

Pruebas Unitarias.

Las pruebas de unidad están orientadas principalmente a validar el cumplimiento de los estándares de presentación y demás características visuales de la aplicación como la salida de los reportes. La prueba de unidad se centra en el módulo, usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. Estas pruebas son comprobaciones que hacemos a las unidades lógicas de nuestro programa. Verificando que una unidad funcione correctamente por sí misma, sin tener en cuenta las relaciones que pueda tener con otras partes del sistema. Estas pruebas son pequeños módulos auxiliares, que se encargan de verificar el funcionamiento de otras unidades lógicas del sistema. Las unidades lógicas de un programa son aquellas partes en que lo hemos dividido para entenderlo mejor. Pueden ser los módulos, paquetes, clases, subsistemas, funciones o cualquier otro mecanismo que nos ofrezca el lenguaje de programación que estamos utilizando.

Pruebas de requerimientos.

Los requerimientos de software deben tener una explicación clara, precisa y completa del problema que facilite el análisis de errores y la generación de casos de prueba. Un asunto de gran importancia es asegurar la corrección, coherencia y exactitud de los requisitos. Durante el proceso de elicitación de requerimientos, una persona, designada por el Equipo de Aseguramiento de Calidad, revisará el documento de especificación de requerimientos, con la lista de chequeo general del documento y la lista de chequeo de requerimientos. La corrección del contenido del documento será responsabilidad del analista y el usuario líder, quienes son los encargados de aprobar los requerimientos definidos en el documento.

En el proceso de elicitación de requerimientos se determinan si los objetivos son claros, verificables y necesarios y el resultado de esta investigación se recoge en una lista de chequeo de objetivos, mediante un proceso iterativo se define la funcionalidad esperada del software y esta información debe ser recogida en el documento de requisitos del sistema, se debe verificar además el documento de

requerimientos usando la lista de chequeo general del documento de especificación de requerimientos, lista de chequeo del documento general del documento de elicitación y análisis de los requerimientos. El proceso de verificación de los requerimientos comienza con el análisis de esos requerimientos y una inspección en la cual se busca evaluar la consistencia, completitud y factibilidad de los requerimientos, tanto individualmente como juntos. Adicionalmente los requerimientos deben ser revisados y validados por los distintos actores involucrados con el sistema. Para evitar sorpresas de variada índole a la hora de entregar el software, es conveniente especificar claramente qué vamos a hacer para determinar que el sistema satisface sus requerimientos.

El diseño de estas pruebas requiere los siguientes pasos:

- ✓ Revisar la verificabilidad del requerimiento.
- ✓ Especificar el criterio de verificación.
- ✓ Hacer visible las propiedades o elementos del software necesarios para verificar el cumplimiento del requerimiento.
- ✓ Hacer controlable los elementos del software necesarios para llevar a cabo las pruebas.
- ✓ Elaborar el plan de pruebas.
- ✓ Ejecutar el plan de pruebas y reportar sus resultados.

El objetivo de las pruebas de verificación de requerimientos es buscar discrepancias entre los requerimientos y la ejecución del software.

Pruebas de Integración.

Las pruebas de integración se llevan a cabo durante la construcción del sistema, involucran a un número creciente de módulos y terminan probando el sistema como conjunto.

El objetivo de este tipo de prueba es coger los módulos probados en la prueba de unidad o unitarias y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Existen dos formas de integración:

- ✓ Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.
- ✓ Integración incremental: El programa se construye y se prueba en pequeños segmentos.

Estas pruebas se pueden plantear desde un punto de vista estructural o funcional. Las pruebas estructurales de integración son similares a las pruebas de caja blanca; pero trabajan a un nivel conceptual superior. En lugar de referirnos a sentencias del lenguaje, nos referiremos a llamadas entre módulos.

Se trata pues de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura de segmentos o de ramas. Las pruebas funcionales de integración son similares a las pruebas de caja negra. Se tratan de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). Según nos vamos acercando al sistema total, estas pruebas se van basando más y más en la especificación de requisitos del usuario.

Las pruebas finales de integración cubren todo el sistema y pretenden cubrir plenamente la especificación de requisitos del usuario. Además, a estas alturas ya suele estar disponible el manual de usuario, que también se utiliza para realizar pruebas hasta lograr una cobertura aceptable. En este tipo de prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

Pruebas de Validación.

El software totalmente ensamblado se prueba como un todo para comprobar si cumple los requisitos funcionales y de rendimiento, facilidad de mantenimiento, recuperación de errores, etc.

El objetivo de estas pruebas es obtener información útil para la validación de la implementación de los algoritmos implementados. Se asume para esta parte que el software ha cumplido la etapa de verificación, por lo tanto está libre de errores de tiempo de ejecución, lo que no significa que esté libre de errores lógicos.

Pruebas de Aceptación.

Las pruebas de aceptación, son las que se hacen con los clientes y define su aceptación del sistema. Son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable de caras al cliente; sino una vez pasadas todas las pruebas de integración por parte del desarrollador. La experiencia ha mostrado que aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente se pone a usarlo.

Sus características principales son las siguientes:

- ✓ Participación del usuario.
- ✓ Está enfocada hacia la prueba de los requisitos de usuario especificados.
- ✓ Está considerada como la fase final del proceso para crear un confianza en que el producto es el apropiado para su uso en explotación.

Pruebas de Sistema.

En el diseño de sistemas, éstos no se toman como sistemas completos ni se prueban como sistemas únicos, razón por la cual deben hacerse pruebas parciales y del sistema en su totalidad. Entre las pruebas especiales de sistemas se pueden considerar las siguientes:
Prueba de carga máxima, prueba de almacenamiento, prueba del tiempo de ejecución, prueba de recuperación, prueba de procedimientos, prueba de factores humanos.

El objetivo de las pruebas del sistema es comprobar la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica. En la realización de estas pruebas es importante comprobar la cobertura de los requisitos y el total cumplimiento de los mismos, dado que su incumplimiento puede comprometer la aceptación del sistema por el equipo de operación responsable de realizar las pruebas de implantación del sistema, que se llevarán a cabo en el proceso de implantación y aceptación del sistema.

Pruebas Alfa y Beta.

Cuando es entregado el software al cliente nunca o casi nunca se encuentra conforme con los resultados. Decir que los requisitos no estaban claros, o que el manual es ambiguo puede salvar la cara de los desarrolladores; pero ciertamente no deja satisfecho al cliente. Por estas razones, muchos desarrolladores ejercitan unas técnicas denominadas "pruebas alfa" y "pruebas beta". Las pruebas alfa y beta son habituales en productos que se van a vender a muchos clientes. Algunos de los potenciales compradores se prestan a estas pruebas bien por ir entrenando a su personal con tiempo o bien a cambio de alguna ventaja económica. La experiencia muestra que estas prácticas son muy eficaces.

Originalmente el término prueba Alfa indica la primera fase de pruebas, que incluye pruebas unitarias, pruebas de componentes y pruebas de sistemas. Las pruebas alfa consisten en invitar al cliente a que venga al entorno de desarrollo a probar el sistema. Se trabaja en un entorno controlado y el cliente siempre tiene un experto a mano para ayudarle a usar el sistema y para analizar los resultados. Una

prueba beta se utiliza en el desarrollo de software para marcar la segunda fase de pruebas de manera que vienen después de las pruebas alfa, en esta segunda fase se incorpora un grupo selecto de usuarios como probadores de las aplicaciones para obtener no solo corrección de errores de codificación sino también apreciaciones sobre la funcionalidad o sea se desarrollan en el entorno del cliente. Aquí el cliente se queda a solas con el producto y trata de encontrarle fallos (reales o imaginarios) de los que informa al desarrollador. La prueba Beta se puede considerar como la prueba anterior a la liberación del producto.

Pruebas Funcionales.

Entre todos los tipos de pruebas que se realizan en un sistema está el tipo que evalúa la funcionalidad de éste, éstas son las llamadas pruebas funcionales ya que se centran en las funciones, las entradas y las salidas. Su objetivo fundamental es asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados, estas pruebas tienen como meta:

- ✓ Verificar el procesamiento, recuperación e implementación adecuada de las reglas del negocio.
- ✓ Verificar la apropiada aceptación de datos.

Estas pruebas se realizan aplicando las pruebas de caja negra ejecutando cada caso de uso, flujo de caso de uso, o función, usando datos válidos e inválidos, para verificar lo siguiente:

- ✓ Que se aplique apropiadamente cada regla de negocio.
- ✓ Que los resultados esperados ocurran cuando se usen datos válidos.
- ✓ Que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.

Pruebas de Seguridad.

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

Prueba de Caja blanca.

Fue propuesta por Tom McCabe en el año 1976, permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que se ejercitan todos los caminos independientes de cada módulo, que se ejercitan todas las decisiones lógicas, que se ejecutan todos los bucles, y que se ejecutan las estructuras de datos internas, en este tipo de prueba normalmente se realizan las siguientes tareas:

- ✓ Análisis de la configuración de todos los sistemas operativos implantados: usuarios, ficheros, etc.
- ✓ Análisis de la robustez de las contraseñas utilizadas.
- ✓ Análisis de la configuración del software base (Web, mail, cortafuegos, etc.).
- ✓ Análisis del código fuente de las aplicaciones instaladas o desarrolladas a medias.
- ✓ Determinación de las vulnerabilidades presentes en los sistemas debido a la desactualización en la aplicación de parches de seguridad.

Una de las pruebas más importantes que se realizan a sistemas informáticos son las de caja blanca y uno de los métodos o herramientas que utiliza es la prueba del camino básico de la cual estaremos hablando. La idea es confeccionar casos de prueba que garanticen que se verifican todos los caminos independientes existentes y para verificar cada camino independiente debemos:

- ✓ Probar sus dos facetas desde el punto de vista lógico, es decir, verdadera y falsa.
- ✓ Ejecutar todos los bucles en sus límites operacionales.
- ✓ Ejercitar las estructuras internas de datos.

<u>Pasos para realizar las pruebas del camino básico.</u>

1.- Para aplicar la técnica del camino básico debemos representar un Grafo de Flujo, el cual está constituido de la siguiente manera:

- ✓ Arista: Representa un flujo de control.
- ✓ Nodo: Representa un conjunto de sentencias.
- ✓ Región: Área limitada por aristas y nodos.

Para la construcción del grafo se tienen en cuenta además las siguientes notaciones para cada instrucción. *Ver Anexo de Figuras* (*Fig. 1.1, 1.2, 1.3, 1.4*).

2.- Luego de haber construido el *Grafo de Flujo* se calcula la complejidad ciclomática que es aquella medida cuantitativa que nos define la complejidad de un programa, y lo que mide es el número de caminos independientes que tiene el programa (cuando hablamos de caminos independientes nos referimos a cualquier camino que introduzca un nuevo conjunto de sentencias o un conjunto de decisiones, expresado en el grafos por un nodo). La complejidad ciclomática da un límite superior que define la cantidad de pruebas que tendríamos que hacer para asegurar que cada línea/sentencia se ejecute al menos una vez, y se calcula de la siguiente manera:

- ✓ El número de regiones del grafo de flujo.
- ✓ V (G) = A - N + 2, donde A es el número de aristas y N es el número de nodos.
- ✓ V (G) = P + 1, donde P es el número de nodos predicado (nodo del cual salen más de dos aristas).

3.- Se determina un conjunto básico de caminos independientes.

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

Pruebas de Caja Negra.

Son las pruebas que se llevan a cabo sobre la interfaz del software, y es completamente indiferente al comportamiento interno y la estructura del programa. Los casos de prueba de la caja negra 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, existen varias técnicas para la aplicación de estas pruebas se pueden entonces mencionar algunas:

<u>Análisis de Valores Límites</u>

Pressman dice que no hay una identificación clara de los valores límite, dice que suele haber más errores en los valores límites que en los típicos. Cuando hablamos de Valores Límites lo que se dice es que se eligen casos de prueba en los límites o bordes de la clase que estamos probando. (por ejemplo: Si una condición de entrada o salida exige un rango entre B y R, se realizan los casos de prueba para los valores B y R, y además a los valores que están por encima de ellos, es decir A y S).

Partición de equivalencia

Es una técnica de simplificación de pruebas. Hay que dividir el dominio de entrada de un programa en clases de datos que tengan algo en común, de ahí derivan clases de prueba y por lo tanto también derivan clases de errores, de esta forma no hay necesidad de ejecutar muchas pruebas para encontrar errores genéricos.

Para lograr ésta se evalúan las clases (conjuntos de estados, válidos o no, para las condiciones de entrada) posibles para una condición de entrada (valor numérico, rango de valores, condición booleana (si o no)).

Pasos para realizar las pruebas de Caja negra: Partición de equivalencia

1.- Se examina cada condición de entrada y se divide en dos o más grupos.

2.-Se identifican dos tipos de clases: clases válidas y clases no válidas.

Condición de entrada	Clases válidas	Clases no válidas

Si la condición de entrada es un:

✓ Rango (se define una clase de equivalencia válida y dos no válidas). Si se especifica un rango de valores para los datos de entrada (por ejemplo, “el número estará comprendido entre 1 y 49”), se creará una clase válida (1 " número " 49) y dos clases no válidas (número < 1 y número > 49).

✓ Valor específico (se define una clase de equivalencia válida y dos no válidas). Si se especifica un número de valores (por ejemplo, “se pueden registrar de uno a tres propietarios de un piso”), se creará una clase válida (1 " propietarios " 3) y dos no válidas (propietarios < 1 y propietarios > 3).

✓ Miembro de conjunto (se define una clase de equivalencia válida y otra no válida). Si se especifica un conjunto de valores admitidos (por ejemplo, “pueden registrarse tres tipos de inmuebles: pisos, chalés y locales comerciales”) y se sabe que el programa trata de forma diferente cada uno de ellos, se identifica una clase válida por cada valor (en este caso son tres: piso, chalé y local) y una no válida (cualquier otro caso: por ejemplo, plaza de garaje).

✓ Lógica (se define una clase válida y otra no válida). Si se especifica una situación del tipo “debe ser” o booleana (por ejemplo, “el primer caracter debe ser una letra”), se identifican una clase válida (“es una letra”) y una no válida (“no es una letra”).

3.- Asignar un número único a cada clase de equivalencia.

4.- Escribir casos de prueba que cubran tantas clases válidas no incorporadas como sea posible hasta que se cubran todas las clases válidas.

5.- Escribir casos de prueba que cubran una sola clase no válida no incorporada hasta que se cubran todas las clases no válidas.

1.12 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.

1.13 Técnicas de prueba de software.

Bajo el nombre de pruebas de software se agrupan un conjunto de prácticas correctivas, frente a las prácticas preventivas que se aplican durante el proceso de construcción de software, cuyo objetivo es determinar la calidad de los sistemas software, existen diferentes técnicas de pruebas de software y se pueden agrupar en las siguientes categorías:

✓ Técnicas Aleatorias: Los casos que se prueban se generan aleatoriamente.

✓ Técnicas Funcionales: Se utilizan las especificaciones del problema para generar los casos de prueba. El programa se ve como una caja negra.

✓ Técnicas de Flujo de Control: Se requiere conocimiento del código fuente. Se seleccionan caminos dentro del programa que deben ser ejecutados al ingresar los casos de prueba.

✓ Técnicas de Flujo de Datos: También requiere conocimiento del código fuente. En este caso, los caminos se eligen en forma de explorar secuencias de eventos relacionadas con el estado de las variables.

✓ Técnicas de Mutación: En la mutación se introducen fallas al programa creando varios mutantes, cada uno con una falla. Los casos de prueba se hacen pasar por los mutantes con la intención de hacer fallar al programa.

1.14 Estrategia de prueba del software.

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a la construcción correcta del software, los objetivos fundamentales para trazarnos una estrategia de prueba son:

✓ Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de unidad, integración y las pruebas de sistema. Las pruebas de unidad y de integración son necesarias dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.

✓ Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.

✓ Realizar diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Los productos de desarrollo de software en los que se detectan defectos son probados de nuevo y posiblemente devueltas a otra etapa, como diseño o implementación, de forma que los defectos puedan ser arreglados.

Para conseguir estos objetivos el flujo de trabajo de la etapa de pruebas consta de las etapas de planificación, diseño, implementación ejecución y evaluación de las pruebas y además los productos de desarrollo del software fundamentales que se desarrollan en la etapa de pruebas son el plan de pruebas, los casos de prueba, el informe de evaluación de las pruebas, el modelo de prueba que incluye a su vez las clases de prueba, el entorno de configuración de pruebas, componentes de prueba y datos de las pruebas.

1.15 Plan de prueba.

Un plan de prueba deberá cumplir con ciertos puntos, en primer lugar se deberá tener bien claro qué tipo de pruebas se van a aplicar y su correcto orden y diseño. Diseñar correctamente las pruebas de caja blanca y caja negra es un punto crucial. Antes de que las pruebas comiencen deberá estar aclarado cual será el punto aceptable de cobertura como así también cual será el método para medir los resultados. Las pruebas informales donde el resultado se aprecia muy sutilmente no deberán ser parte del plan de pruebas. El ingeniero de pruebas es el encargado de elaborar el plan de pruebas, el cual contiene los objetivos, los recursos, pasos a seguir para la realización de las pruebas, la estrategia de prueba, el resultado de las pruebas entre otros aspectos relevantes para la confección del plan.

1.15.1 Estructura del Plan de prueba.

Introducción.

Describe porqué el Plan de Pruebas fue desarrollado, cuales son sus objetivos fundamentales. Esto puede incluir requerimientos de documentación, definición de estrategias de prueba, identificación de recursos, estimación de plazos y proyección de entregables, además incluye el propósito, dejando de forma explícita el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos de un proceso de pruebas.

Pasos a seguir para la realización de las pruebas.

Este aspecto describe detalladamente qué hacer par llevar a cabo correctamente el proceso de aplicación de las pruebas, paso a paso se define cómo, con qué métodos y estrategias se realizan las pruebas al software, además de plantear que tipo de prueba se realizarán.

Requerimientos de Prueba.

El punto de requerimiento de prueba contiene una lista de todos los requerimientos ya sean de software o hardware. Cualquier requerimiento que no esté incluido en esta lista estará fuera del alcance de las pruebas. Esto libera la responsabilidad en caso de que existan problemas con la funcionalidad del sistema que se relaciona con un requisito no incluido en este listado. En el caso de los requerimientos funcionales todas las funciones deben ser probadas. Las pruebas de la interfaz de usuario, las estructuras de menú u otros elementos de diseño también deberían ser puestas en una lista o por lo menos ser referenciados hacia otro documento. Los requerimientos para probar el flujo de datos desde un componente a otro deben ser incluidos también ya que ellos formarán parte del Plan de Pruebas.

Estrategia de Prueba.

En esta sección se describen los objetivos de las pruebas que serán alcanzados para cada uno de los tipos de pruebas que formaran parte del plan de prueba, por ejemplo pruebas unitarias, integración, sistema, validación y verificación, estrés, configuración y/o de instalación, en este punto se pondrán todos los tipos de prueba que serán aplicadas al software que se está probando así como las técnicas a aplicar.

Herramientas.

Especifica los instrumentos o herramientas que serán empleados para la realización de las pruebas. En caso de que no se utilice ninguna herramienta automatizada pues se especifica de la misma forma.

Recursos.

Identificar los roles y las responsabilidades que serán requeridas para la ejecución del Plan de Pruebas, además se deben tener en cuenta los recursos de software y hardware, las herramientas de soporte y demás recursos que sean necesarios para dar paso a la ejecución de las pruebas.

Actividades de prueba.

En este punto se debe detallar las actividades que realizaran los probadores, especificando los días planificados para cada actividad y la fecha de comienzo y fin de las mismas.

Tareas de la etapa de prueba.

Dentro de cada actividad realizada existen una serie de pasos que debemos dejar bien documentados en el plan de prueba, o sea se especificará detalladamente qué pasos se deben seguir para dar cumplimiento a cada actividad.

Resultados de las pruebas.

Cuando el esfuerzo de prueba esté terminado, se documentan los resultados y las mediciones. Identificando cualquier discrepancia que pueda existir entre el plan de prueba propuesto y la puesta en práctica real describiendo adecuadamente cómo aquellas discrepancias fueron manejadas.

1.16 Herramientas usadas para realizar pruebas de software.

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.

TEST.

Es una unidad de pruebas automatizada y productos de análisis de código estándar, que trabaja sobre clases escritas en la plataforma Microsoft .NET, sin requerir que los desarrolladores realicen un caso de prueba. TEST mejorar 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. Para exponer los problemas de fiabilidad, TEST 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. Para exponer errores funcionales. Para asegurar 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.

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 eminencias en el campo, uno por desarrollar la 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 de que 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), etc. 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 ya la hemos explicado: se trata de desarrollar una unidad que se encargue de probar a otra unidad.

1.17 Conclusiones.

Es importante destacar que la calidad del software está estrechamente relacionada con las prueba de software, ya que es en esta actividad donde se detectan los errores y las inconsistencias que pueda tener el software, que atentan por supuesto con la calidad. Siendo esta una actividad tan importante como pudiera ser el diseño o la implementación, se podrá entonces tener un producto libre de defectos o por lo menos la menor cantidad.

Muchos elementos son fundamentales a la hora de llevar a cabo un proceso de prueba como por ejemplo la documentación del producto desarrollado, muchos le dan la mínima importancia a este aspecto sin darse cuenta que puede el software entonces perder su capacidad de ser reutilizado o modificado en caso de faltar alguno de los desarrolladores, el plan de prueba, las actividades de pruebas a realizar, los ítems a probar, los riesgos, los responsables de cada actividad así como los recursos utilizados deben estar previamente documentados y por ende bien especificados.

Se puede señalar por último que las empresas cubanas se encuentran en estos momentos en un estado muy básico en cuanto a lo que calidad y pruebas de software se refiere, ahora es que se están dando los primeros pasos en este sentido, de manera que las pruebas de software no están siendo correctamente aplicadas quizás por falta de preparación del personal, factor que influye considerablemente en el desarrollo de cualquier proceso.

CAPÍTULO 2: DISEÑO DE LAS PRUEBAS.

2.1 Introducción.

Una de las actividades de mayor importancia dentro de todo proceso de realización de las pruebas es la planeación ya que permite establecer una estrategia a seguir para el proyecto. En los procesos de pruebas esta etapa es vital ya que define el marco de trabajo sobre el cual se efectuará el proceso, sin embargo otro de los aspectos por los que la planeación resulta importante es la base histórica que genera a partir de la cual se establecen referentes para los futuros proyectos. Cada proceso de pruebas se inicia con un plan de pruebas, la cual es usado al final de cada proceso para realizar las comparaciones respectivas y establecer las diferencias que permiten identificar si los proyectos se están subestimando o sobreestimando. A continuación se describe el proceso de planeación además del proceso de diseño de pruebas que se propone para el software “Árbol Genealógico”.

2.2 Software Árbol Genealógico.

Este software ha sido diseñado para la representación gráfica de un árbol genealógico, siendo éste el resultado de un estudio de las funcionalidades de otros software existentes con el mismo objetivo pero que ninguno cumplía lo requisitos que necesitaban los genetistas de nuestro país.

El software tiene una interfaz amigable y fácil de manipular, no se requiere de grandes cursos acerca de cómo trabajar con él. El producto está disponible en español y brinda la posibilidad de representar familias gráficamente cumpliendo con los estándares internacionales para su representación.

Alguna de las funcionalidades que brinda son: crear un nuevo árbol, insertar individuos, guardar propiedades de algún individuo y poder verla en tiempo de ejecución, adicionar familiar a un individuo así como definir su parentesco con éste, adicionar relaciones conyugales existentes o antiguas, adicionar una enfermedad así como darle una simbología con varias opciones, adicionar exámenes y muestras, así como algún comentario de algún individuo, además permite guardar una imagen del árbol completo o sólo de una parte de éste, así como un resumen de los datos de cada uno de los individuos representados, se puede guardar y luego cargar en tiempo de ejecución, entre otras funcionalidades que nos brinda.

En el futuro se debe integrar a una plataforma con varios módulos donde se conecte a una base de datos y mediante ella interactúen todos los módulos como un sistema integrado.

2.3 Características a probar en el software.

Las pruebas de software varían en dependencia particularmente del tipo de producto que se desea probar, cada producto desarrollado tiene sus características propias debido a que algunos tienen mayor nivel de complejidad y debido a ello se prueban de manera diferente, pero generalmente el ingeniero de prueba a la hora de enfrentarse a las pruebas verifica que el software se ejecute sin errores, que sea entendible y fácil de usar para el usuario, que tenga toda la documentación requerida por el usuario y que cumpla con las normas internacionales de calidad. Se miden también atributos como: la fiabilidad, flexibilidad, robustez, tamaño, eficiencia, la modularidad, complejidad, portabilidad, usabilidad, reutilización.

En el producto Árbol Genealógico los principales atributos que se miden son facilidad de uso, robustez, comprensión y la complejidad ciclomática del programa, de manera que la aplicación cumpla con la mayoría de las especificaciones planteadas y exigidas por la empresa cliente.

2.4 Plan de prueba.

Introducción.

Propósito:

Se realiza un Plan de Pruebas para el sistema de software Árbol Genealógico, cuyos objetivos fundamentales son:

- ✓ Identificar los elementos que se van a probar.
- ✓ Describir la estrategia de pruebas que se va a seguir en el proceso de prueba.
- ✓ Identificar los recursos necesarios para llevar a cabo el proceso de prueba y estimar los esfuerzos que conlleva la realización de este proceso.
- ✓ Detallar los resultados que se obtienen de las actividades de prueba.

Ámbito:

En este Plan de Pruebas se describe las pruebas de unidad (caja blanca y caja negra) que se aplicarán al sistema software desarrollado, realizando las pruebas a todos los requisitos definidos en la especificación de requisitos y en el modelo de casos de uso, verificando además la correcta ejecución del producto partiendo de la revisión del código fuente.

Objetivo:

El objetivo fundamental que se persigue con la realización de estas pruebas es: la correcta ejecución del proyecto de software, teniendo en cuenta un conjunto de casos de pruebas que son definidos con anterioridad, con la primordial intención de detectar fallas o errores que puedan existir en el software y poder dar al cliente un mayor nivel de confiabilidad en el sistema desarrollado y por ende en los procesos que se desarrollarán a partir de la utilización de este producto.

Para la elaboración de los casos de pruebas y su posterior aplicación hay que tener en cuenta que un buen caso de prueba es aquel que tiene altísima probabilidad de mostrar un error que hasta entonces no se había descubierto y que por tanto los desarrolladores puedan corregirlo.

Pasos a seguir para la realización de las pruebas.

En el presente trabajo se realizan pruebas funcionales (Caja negra) y de unidad (Caja blanca) al software ArboGen en general. Con el principal objetivo de lograr la satisfactoria instalación del producto en el Centro de Genética Medica, se decidió aplicar pruebas funcionales (caja negra) al software en su conjunto, partiendo de las entradas y las salidas de datos del software, en este caso se elabora un plan de prueba específico el cual comprende todos los casos de prueba con las entradas y las salidas de cada funcionalidad, se realizaran las pruebas a todas las interfaces, diseñando así los casos de prueba de manera que las clases validas abarcaran la mayor cantidad de clases posibles a probar en un caso de prueba, la clases inválidas serán diseñadas en dependencia de cada clase válida seleccionada que así lo necesite , en este tipo de prueba es importante mostrar el resultado esperado en la ejecución de cada prueba en la sección que se esté probando, además de especificar la condición de entrada en cada caso, con esto el probador se percata si eran esos los resultados esperados y por tanto en caso contrario detectar el error que contenga el software.

Las pruebas de unidad (Caja blanca) se efectuaron a todas las clases del software, eligiendo siempre las que presentan condiciones más complejas, de esta forma se elaboró un plan de prueba específico, construyendo el grafo de flujo, analizando los posibles caminos básicos realizando las prueba a cada uno de los caminos para velar por su total funcionamiento y calculando luego la complejidad ciclomática del programa. Los casos de prueba se identificarán con la siguiente notación para ambos tipos de prueba: <Tipo de prueba>, Clase <nombre de la clase> Ejemplo: tipo de prueba (CB) Clase (Familia): **CBClaseFamilia.** Se tomaron las clases y los métodos más significativos o sea los que están más propensos a colapsar la aplicación. Todos los nombres se utilizaron en español pues es la notación utilizada en los ejecutables por los desarrolladores del sistema. En ambos tipos de prueba se

llega a un resultado analizando detalladamente los errores producidos siendo enviados estos a los desarrolladores para su posterior corrección.

Requerimientos de Prueba.

En la realización de las pruebas al software Árbol Genealógico se tuvo en cuenta que el objetivo fundamental de las pruebas es asegurar el trabajo apropiado de los requerimientos funcionales implícitos en el proyecto, se verifica el correcto procesamiento de los datos y la implementación adecuada de las reglas del negocio, que los resultados sean los esperados cuando se introduzcan los datos válidos, que se muestren mensajes de error apropiados cuando se usan datos inválidos. Para el comienzo del diseño y para la aplicación de las pruebas se esperó que el software cumpliera con la mayor cantidad de funcionalidades posibles o sea que el software se comporte de manera estable , ya que no tiene caso realizar las pruebas a las interfaces del software cuando aún reporta grandes fallos y no están las principales funcionalidades programadas y que la documentación no esté ni siquiera comenzada, por otra parte tampoco es recomendable esperar que el software y la documentación estén del todo terminadas ya que pueden existir errores gravísimos, los cuales influyen considerablemente en los costos del proyecto, y puede que ya sea demasiado tarde para remediarlos.

Se muestra una identificación de los elementos (requisitos funcionales y requisitos no funcionales) que son objetivos de las pruebas. Es decir, los elementos que vamos a probar.

Para las pruebas de caja negra se tiene en cuenta los requerimientos generales del sistema:

1. Establecer relaciones entre los individuos del árbol.
2. Desconectar a un individuo de sus relaciones.
3. Eliminar un individuo.
4. Permitir la consistencia de datos, mediante un fichero.
5. Exportar una imagen del árbol.
6. Exportar un reporte en forma de texto con los principales datos del árbol y cada individuo.
7. Mostrar los datos de los individuos en forma asequible.
8. Permitir aumentar y disminuir el tamaño de los individuos.
9. Enumerar según la generación los individuos del árbol.
10. Establecer simbología estándar para la representación de las enfermedades y otros aspectos relevantes del estudio.
11. Establecer una simbología adicional (con símbolos numéricos) para resaltar alguna característica especial en los individuos.

12. Permitir la definición de nuevos símbolos y símbolos adicionales.
13. Hacer una vista tabular con los principales datos de los individuos del árbol.
14. Definir enfermedades (consistencia).
15. Definir exámenes (consistencia).

Para las pruebas de caja blanca se tiene en cuenta lo siguiente:

- ✓ Verificar que se ejerciten por lo menos una vez todos los caminos independientes de cada método.
- ✓ Permitir obtener una medida de la complejidad lógica del diseño y usar esta medida como guía para la definición de un *conjunto básico*.
- ✓ Derivar los casos de prueba a partir de un conjunto dado de caminos independientes.

Estrategia de Prueba.

Se presenta el enfoque que se utiliza para probar el sistema ArboGen, de manera que se define cómo se realizarán las pruebas, o sea los tipos y las técnicas a utilizar.

Pruebas de Caja Negra.

El objetivo de estas pruebas es asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados, velando que se cumplan fielmente las reglas planteadas para el negocio. Este tipo de pruebas está basada en la técnica de caja negra, es decir, verificar la aplicación interaccionando a través de las interfaces de usuario y analizando los resultados.

Objetivo de la prueba	Asegurar la navegación correcta de la aplicación, la entrada y salida de los datos, su procesamiento y recuperación.
Técnicas	Se emplea la técnica de la partición equivalente ejecutando cada caso de uso y flujo del caso de uso con datos válidos e inválidos para verificar lo siguiente: ✓ Cuando se utilizan datos correctos se obtienen los resultados esperados. ✓ Cuando se utilizan datos incorrectos se obtienen los mensajes de error o advertencias adecuadas. ✓ Que cada regla de negocio se ha aplicado correctamente.

Pruebas de Caja Blanca.

Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que se ejercitan todos los caminos independientes de cada módulo, que se ejercitan todas las decisiones lógicas, que se ejecutan todos los bucles, y que se ejecutan las estructuras de datos internas.

Objetivo de la prueba	Asegurar la correcta ejecución de todos los caminos lógicos independientes, pudiendo obtener además la complejidad ciclomática del programa.
Técnicas	Se aplica la técnica del camino básico de la siguiente manera: ✓ A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado. ✓ Se calcula la complejidad ciclomática del grafo. ✓ Se determina un conjunto básico de caminos independientes. ✓ Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Herramientas.

Existen actualmente un sinnúmero de herramienta automatizadas para la ejecución de las pruebas de software, pudiendo citar algunas como JUnit, Rational Test, entre muchas otras, pero hasta hoy en las pruebas realizadas a los software en la Universidad de las Ciencias Informáticas no ha hecho uso de estas herramientas es por ello que no se utiliza específicamente ninguna para el desarrollo de estas pruebas.

Recursos.

Se plasman los recursos necesarios para realizar el proceso de prueba, sus principales responsabilidades y características.

Recursos de hardware:

Para realizar las pruebas fue necesario contar con algunos recursos de hardware dentro de los cuales podemos mencionar los siguientes:

- ✓ Microprocesador Pentium o superior.
- ✓ Al menos 128 MByte de memoria RAM.
- ✓ Mínimo de 2 GByte de Disco duro.

La ejecución de las pruebas se realizó desde una Personal Computer (PC) conectada a una red local, donde está instalado el sistema. Para diseñar, y ejecutar las pruebas fue necesario contar con un probador, así como también se necesitó un estimado de tiempo de 9 meses y 7 horas al día de tiempo de máquina. Cabe aclarar que en este tiempo estimado cae todo el tema de la investigación ya que era desconocido tanto el software como el procedimiento correcto a seguir para la ejecución de las pruebas.

Recurso	Cantidad	Nombre y Tipo
PC	1	Diseño de las pruebas y ejecución de las pruebas.

Recursos de software:

- ✓ Windows 95 o superior.
- ✓ Office 97 o superior.
- ✓ Microsoft Framework .NET 1.

Herramientas de soporte:

No se utiliza ninguna herramienta de soporte para la ejecución de estas pruebas.

Recursos humanos:

Rol	Mínimos recursos recomendados	Responsabilidades específicas.
Diseñador de prueba	1	Identificar y priorizar los casos de prueba. ✓ Generar el Plan de pruebas. ✓ Diseñar los Casos de prueba. ✓ Evaluar el resultado de las

		pruebas.
Probador (Tester)	1	Ejecutar las pruebas. ✓ Ejecutar pruebas. ✓ Recuperar los errores. ✓ Documentar los defectos.

Actividades de prueba.

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

Actividad	Esfuerzo (p/días)	Fecha de comienzo	Fecha de finalización
Planificación de la prueba	4	19/03/2007	23/03/2007
Diseño de la prueba	28	26/03/2007	23/04/2007
Ejecución de la prueba	7	23/04/2007	30/04/2007
Evaluación de la prueba	15	1/05/2007	15/05/2007

Tareas de la etapa de prueba.

Para la realización de cada una de las actividades de pruebas se siguen una serie de pasos que se le denominan tareas las cuales se mencionan a continuación.

Planificación de las pruebas:

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

Diseño de las pruebas:

- ✓ Desarrollo de las pruebas.
- ✓ Identificar y describir los casos de prueba.

Ejecución de las pruebas:

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

Evaluación de las pruebas:

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

2.5 Plan de prueba Específico.

Plan de prueba de Caja Blanca.

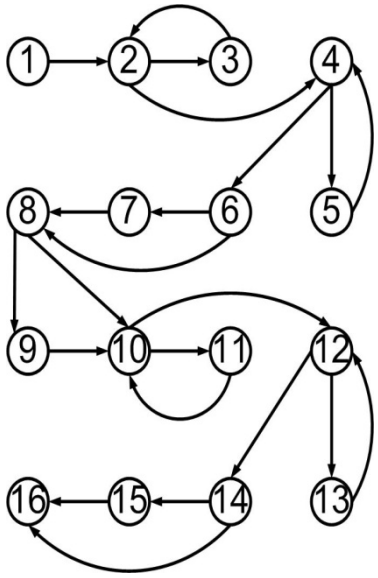
Para obtener los casos de prueba en el método de la Caja Blanca se utiliza la técnica del camino básico, la cual es muy efectiva para obtener todas las variantes en las que puede correr un procedimiento, comprobando que el programa en cada de una de sus variantes llegó al resultado esperado, se probó además la estabilidad requerida, se mostrarán los métodos más significativos, los demás se podrán encontrar en Anexo I.

2.5.1 CBClaseFamilia

Método PasarDatosMujerAMujer

```
private void PasarDatosMujerAMujer (Mujer destino, Mujer fuente )
  {
     Persona madre = fuente.Madre;
     Persona padre = fuente.Padre;                    1
     DesconectarDeLosPadres( fuente);
     for(int i=0; i<fuente.Hijos.Count; i++)  2
        fuente.Hijos[i].BorrarMadre();  3
     for(int i=0; i<fuente.Relaciones.Count; i++)  4
        fuente.Relaciones[i].BorrarRelacion(fuente); 5
```

```
destino.Nombre = fuente.Nombre;
destino.PrimerApellido=fuente.PrimerApellido;
destino.SegundoApellido= fuente.SegundoApellido;
destino.Muestras = fuente.Muestras;
destino.Identificador = fuente.Identificador;
destino.FechaDeNacimiento.AddHours( fuente.FechaDeNacimiento.Hour);
destino.FechaDeNacimiento.AddDays( fuente.FechaDeNacimiento.Day);
destino.FechaDeNacimiento.AddMonths( fuente.FechaDeNacimiento.Month);
destino.FechaDeNacimiento.AddYears( fuente.FechaDeNacimiento.Year);
destino.FechaDeMuerte.AddHours( fuente.FechaDeMuerte.Hour);
destino.FechaDeMuerte.AddDays( fuente.FechaDeMuerte.Day);                    5
destino.FechaDeMuerte.AddMonths( fuente.FechaDeMuerte.Month);
destino.FechaDeMuerte.AddYears( fuente.FechaDeMuerte.Year);
RemoverPersona(fuente);
AdicionarPersona(destino);
destino.Generacion = fuente.Generacion;
destino.Numero = fuente.Numero;
destino.Proposito = fuente.Proposito;
destino.Tipo_Gem = fuente.Tipo_Gem;
if (madre != null)   6
    AdicionarMadre(madre, destino);  7
if (padre != null)  8
   AdicionarPadre(padre, destino);  9
for(int i=0; i<fuente.Hijos.Count; i++)  10
    AdicionarMadre(destino, fuente.Hijos[i]);  11
for(int i=0; i<fuente.Relaciones.Count; i++)  12
    AdicionarEsposo(destino, fuente.Relaciones[i]);  13
 if (CambioDeEmbarazo!= null)  14
  {
      CambioDeEmbarazo(newEventAgrsFamiliaCambioEmbarazo((Persona)fuente,
(Persona)destino));  15
  }
}  16
```



Caminos básicos	Complejidad ciclomática
C1:1-2-4-6-8-10-12-14-16	V(G)= A-N+2
C2:1-2-4-5-4-6-8-10-12-14-16	V(G)= 22- 16 + 2 = 8 = R8
C3:1-2-4-6-7-8-10-12-14-15-16	
C4:1-2-4-5-4-6-8-9-10-12-14-16	
C5:1-2-4-6-7-8-9-10-12-14-15-16	
C6:1-2-3-2-4-6-7-8-10-11-10-12-13-12-14-16	
C7:1-2-3-2-4-5-4-6-7-8-9-10-11-10-12-13-12-14-15-16	
C8: 1-2-4-6-7-8-9-10-12-14-16	

Camino 1: 1-2-4-6-8-10-12-14-16

Caso de prueba: No se traspasan los datos.

Entrada: Mujer fuente (Hijos = 0, Relaciones = 0), Mujer destino (madre = null, padre = null, CambioDeEmbarazo = null, Hijos = 0, Relaciones = 0).

Resultado Esperado: Desconecta a la mujer del resto de la familia, no cumple con ninguna condición por tanto no se traspasan los datos de la mujer fuente a la mujer destino.

Condiciones: Exista mujer destino.

Camino 2: 1-2-4-5-4-6-8-10-12-14-16

Caso de prueba: Traspasar datos.

Entrada: Mujer fuente (Hijos = 0, Relaciones = 3, Nombre = Ana, PrimerApellido = Pérez, SegundoApellido = Lora, Muestras = orina, Identificador = 156735, FechaDeNacimiento = 24/05/1970, FechaDeMuerte = 3/11/2006, Generacion = segunda, Numero = 3, Proposito = true, Tipo_Gem = microRNA), Mujer destino (Hijos = 0, Relaciones = 0, CambioDeEmbarazo = null, madre = null, padre = null).

Resultado Esperado: Desconecta a la mujer del resto de la familia, se traspasan todos los datos de la mujer fuente a la mujer destino.

Condiciones: Exista mujer destino.

Camino 3: 1-2-4-6-7-8-10-12-14-15-16

Caso de prueba: Establecer relación con la madre y traspaso de embarazo.

Entrada: Mujer fuente (Hijos = 0, Relaciones = 0), Mujer destino (Hijos = 0, Relaciones = 0, CambioDeEmbarazo != null, madre = María, padre = null).

Resultado Esperado: Desconecta a la mujer del resto de la familia, le adiciona la relación con la madre a la mujer destino, se traspasa el embarazo.

Condiciones: Exista mujer destino.

Camino 4: 1-2-4-5-4-6-8-9-10-12-14-16

Caso de prueba: Traspasar datos y establecer relación con el padre.

Entrada: Mujer fuente (Hijos = 0, Relaciones = 3, Nombre = Ana, PrimerApellido = Pérez, SegundoApellido = Lora, Muestras = orina, Identificador = 156735, FechaDeNacimiento = 24/05/1970, FechaDeMuerte = 3/11/2006, Generacion = segunda, Numero = 3, Proposito = true, Tipo_Gem = microRNA), Mujer destino (Hijos = 0, Relaciones = 0, CambioDeEmbarazo = null, madre = null, padre = Pedro).

Resultado Esperado: Desconecta a la mujer del resto de la familia, se traspasan todos los datos de la mujer fuente a la mujer destino, le adiciona la relación con el padre a la mujer destino.

Condiciones: Exista mujer destino.

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

Caso de prueba: Establecer relación con los padres y traspaso de embarazo.

Entrada: Mujer fuente (Hijos = 0, Relaciones = 0), Mujer destino (Hijos = 0, Relaciones = 0, CambioDeEmbarazo != null, madre = María, padre = Pedro).

Resultado Esperado: Desconecta a la mujer del resto de la familia, adiciona la relación con el padre y con la madre a la mujer destino, traspasando el embarazo.

Condiciones: Exista mujer destino.

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

Caso de prueba: Establecer relaciones.

Entrada: Mujer fuente (Hijos = 3, Relaciones = 0), Mujer destino (Hijos = 3, Relaciones = 3, CambioDeEmbarazo = null, madre = María, padre = null).

Resultado Esperado: Desconecta a la mujer del resto de la familia, se le adiciona la relación con la madre y el esposo a la mujer destino.

Condiciones: Exista mujer destino.

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

Caso de prueba: Traspasar datos de mujer fuente a mujer destino.

Entrada: Mujer fuente (Hijos = 3, Relaciones = 3, Nombre = Ana, PrimerApellido = Pérez, SegundoApellido = Lora, Muestras = orina, Identificador = 156735, FechaDeNacimiento = 24/05/1970, FechaDeMuerte = 3/11/2006, Generacion = segunda, Numero = 3, Proposito = true, Tipo_Gem = microRNA), Mujer destino (Hijos = 3, Relaciones = 3, CambioDeEmbarazo != null, madre = María, padre = Pedro).

Resultado Esperado: Desconecta a la mujer del resto de la familia, se traspasan todos los datos de la mujer fuente a la mujer destino, adiciona la relación con el padre, con la madre y con el esposo a la mujer destino, se traspasa el embarazo.

Condiciones: Exista mujer destino.

Camino 8: 1-2-4-6-7-8-9-10-12-14-16

Caso de prueba: Establecer relación con los padres.

Exista mujer destino.

Entrada: Mujer fuente (Hijos = 0, Relaciones = 0), Mujer destino (Hijos = 0, Relaciones = 0, CambioDeEmbarazo = null, madre = María, padre = Pedro).

Resultado Esperado: Desconecta a la mujer del resto de la familia, adiciona la relación con el padre y con la madre a la mujer destino.

Condiciones: Exista mujer destino.

Para los demás métodos ver Anexo I.

2.5.2 CBVentanaAdicionarFamiliar

Método VentanaAdicionarFamiliar

```
public VentanaAdicionarFamiliar(Persona persona)
        {
            InitializeComponent();          ⎫ 1
            Tag = Acciones.NoAccion;        ⎭
            if (persona.Madre!= null)   2
                bMadre.Enabled = false;   3
            else
                bMadre.Enabled = true;   4
            if (persona.Padre!= null)   5
```

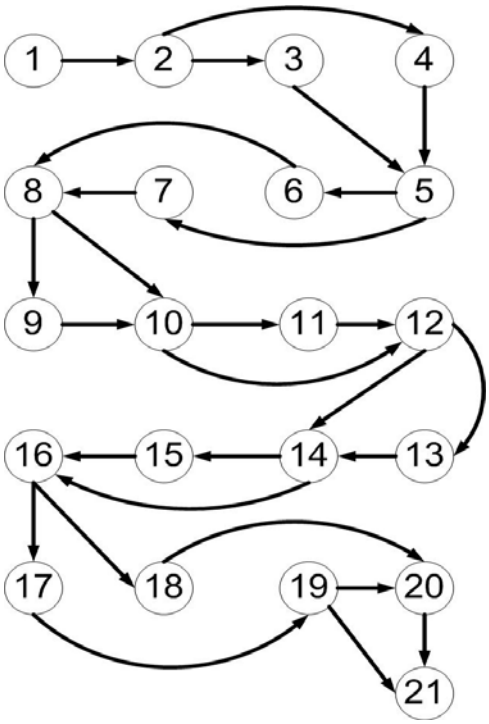
```
            bPadre.Enabled = false;   6
else
            bPadre.Enabled = true;   7
if (persona.Sexo == Sexos.Desconocido)    8
{
            bEsposa.Enabled = false;   9
            bEsposo.Enabled = false;
}
if (persona.Sexo == Sexos.Femenino)   10
            bEsposa.Enabled = false; 11

if (persona.Sexo == Sexos.Masculino) 12
            bEsposo.Enabled = false;   13
if (persona.Sexo == Sexos.Desconocido)  14
{
            bHijo.Enabled = false;
            bHija.Enabled = false;                  15
            bDescendiente.Enabled = false;
}
if ( (persona.Padre != null) && (persona.Madre != null) )   16
{

             bHermana.Enabled = true;
            bHermano.Enabled = true;          17
            bDesconocido.Enabled = true;
             bgemelo.Enab-led = true;

}
else
{
            bHermana.Enabled = false;
            bHermano.Enabled = false;          18
            bDesconocido.Enabled = false;
            bgemelo.Enabled = false;
```

```
        }
        if(persona.Grupo==true)  19
        {
                bdivorciar.Enabled=false;   20
                bGrupo.Enabled=false;
        }
}21
```



Caminos básicos	**Complejidad ciclomática**
C1:1-2-4-5-6-8-9-10-11-12-13-14-15-16-18-19-21	V(G)= A-N+2
C2:1-2-3-5-7-8-9-10-11-12-13-14-15-16-18-19-21	V(G)= 28 – 21 + 2 = 9 = R9
C3:1-2-3-5-6-8-10-11-12-14-16-18-19-21	
C4:1-2-3-5-6-8-10-12-13-14-16-18-19-21	
C5: 1-2-3-5-6-8-9-10-12-14-16-18-19-21	
C6: 1-2-3-5-6-8-10-12-14-15-16-18-19-21	
C7: 1-2-3-5-6-8-10-12-14-16-17-19-21	
C8: 1-2-3-5-6-8-10-12-14-16-18-19-21	
C9: 1-2-3-5-6-8-10-12-14-16-18-19-20-21	

Camino 1: 1-2-4-5-6-8-9-10-11-12-13-14-15-16-18-19-21

Caso de prueba: Activar madre.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = null, Sexo = Desconocido, Sexo = Femenino, Sexo = Masculino, Grupo = false

Resultado Esperado: Activa la relación de madre a la persona seleccionada.

Condiciones: Exista persona.

Camino 2: 1-2-3-5-7-8-9-10-11-12-13-14-15-16-18-19-21

Caso de prueba: Activar padre.

Entrada: persona (Nombre Luisa), Madre = null, Padre = Pedro, Sexo = Desconocido, Sexo = Femenino, Sexo = Masculino, Grupo = false

Resultado Esperado: Activa la relación de padre a la persona seleccionada.

Condiciones: Exista persona.

Camino 3: 1-2-3-5-6-8-10-11-12-14-16-18-19-21

Caso de prueba: No activa la opción de esposa.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = Pedro, Sexo = Femenino, Grupo = false

Resultado Esperado: Si el sexo de la persona es femenino no activa la opción de esposa.

Condiciones: Exista persona.

Camino 4: 1-2-3-5-6-8-10-12-13-14-16-18-19-21

Caso de prueba: No activa la opción de esposo.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = Pedro, Sexo = Masculino, Grupo = false

Resultado Esperado: Si el sexo de la persona es masculino no activa la opción de esposo.

Condiciones: Exista persona.

Camino 5: 1-2-3-5-6-8-9-10-12-14-16-18-19-21

Caso de prueba: No activa la opción de esposo ni esposa.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = null, Sexo = Desconocido, Grupo = false

Resultado Esperado: Si el sexo de la persona es desconocido no activa la opción de esposo ni de esposa.

Condiciones: Exista persona.

Camino 6: 1-2-3-5-6-8-10-12-14-15-16-18-19-21

Caso de prueba: No activa la opción de hijo, hija y descendiente.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = Pedro, Sexo = Desconocido, Sexo = Femenino, Sexo = Masculino, Grupo = false

Resultado Esperado: Si el sexo de la persona es desconocido no activa la opción de hijo, hija ni descendiente.

Condiciones: Exista persona.

Camino 7: 1-2-3-5-6-8-10-12-14-16-17-19-21

Caso de prueba: Activa la opción de hermana, hermano, desconocido y gemelo.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = Pedro, Sexo = Desconocido, Sexo = Femenino, Sexo = Masculino, Grupo = false

Resultado Esperado: Existe madre y padre por tanto se activa la opción de hermana, hermano, desconocido y gemelo.

Condiciones: Exista persona.

Camino 8: 1-2-3-5-6-8-10-12-14-16-18-19-21

Caso de prueba: No activa la opción de hermana, hermano, desconocido y gemelo.

Entrada: persona (Nombre Luisa), Madre = null, Padre = null, Sexo = Desconocido, Sexo = Femenino, Sexo = Masculino, Grupo = false

Resultado Esperado: No existe madre ni padre por tanto no se activa la opción de hermana, hermano, desconocido y gemelo.

Condiciones: Exista persona.

Camino 9: 1-2-3-5-6-8-10-12-14-16-18-19-20-21

Caso de prueba: No activa la opción de grupo y divorciar.

Entrada: persona (Nombre Luisa), Madre = Ana, Padre = Pedro, Sexo = Desconocido, Sexo = Femenino, Sexo = Masculino, Grupo = true

Resultado Esperado: No se activa la opción de grupo ni divorciar.

Condiciones: Exista persona.

Plan de prueba de Caja Negra.

Para la aplicación de las pruebas de Caja negra se utiliza la técnica de partición de equivalencia, siendo esta muy efectiva a la hora de realizar la pruebas sobre la interfaz del software probando la validez de cada entrada de datos o información al sistema de manera que es totalmente indiferente al comportamiento interno, estas pruebas pretenden demostrar que las funciones del software son operativas, cada entrada se acepta de forma adecuada y que a la vez que se produce una salida correcta la integridad de la información externa se mantiene. Un aspecto importante a medir cuando se llevan a cabo este tipo de pruebas es que se cumplan a cabalidad los requerimientos funcionales que fueron previstos en la etapa de análisis más específicamente en el flujo de trabajo de Requerimientos, tanto los requeridos por los clientes como los necesarios para el funcionamiento correcto de la aplicación, estos requerimientos se listan a continuación:

1. Establecer relaciones entre los individuos del árbol.
2. Desconectar a un individuo de sus relaciones.
3. Eliminar un individuo.
4. Permitir la consistencia de datos, mediante un fichero.
5. Exportar una imagen del árbol.
6. Exportar un reporte en forma de texto con los principales datos del árbol y cada individuo.
7. Mostrar los datos de los individuos en forma asequible.
8. Permitir aumentar y disminuir el tamaño de los individuos.
9. Enumerar según la generación los individuos del árbol.
10. Establecer simbología estándar para la representación de las enfermedades y otros aspectos relevantes del estudio.
11. Establecer una simbología adicional (con símbolos numéricos) para resaltar alguna característica especial en los individuos.
12. Permitir la definición de nuevos símbolos y símbolos adicionales.
13. Hacer una vista tabular con los principales datos de los individuos del árbol.
14. Definir enfermedades (consistencia).
15. Definir exámenes (consistencia).

2.5.3 CN MóduloGeneral

2.5.3.1 Interfaz Datos médicos

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Selecciona el botón A*dicionar*.		El sistema adiciona una muestra (muestra de sangre predefinida).	Muestra
	Selecciona el botón adicionar.	El sistema no adiciona la muestra.	
Selecciona el botón E*liminar.*		El sistema elimina la muestra seleccionada.	
Selecciona la muestra que se desea cambiar.		El sistema da la opción debe cambiar el tipo de muestra.	Datos de la Muestra
	Selecciona la muestra que se desea cambiar.	El sistema no cambie al tipo de muestra seleccionado.	

2.5.3.2 Interfaz Estado genético

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Selecciona la opción de símbolo.		El sistema debe poner al individuo el símbolo especificado ya sea afectado, portador, posible portador, etc.	Símbolo
	Selecciona la opción de símbolo.	El sistema no pone al individuo el símbolo especificado.	
Selecciona la opción de símbolo adicional.		El sistema debe poner al individuo el símbolo especificado ya sea +, -, *.	Símbolo Adicional
	Selecciona la opción de símbolo adicional.	El sistema no pone al individuo el símbolo especificado.	

2.5.3.3 Interfaz Familiar

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Introducir la cantidad de hijos		Al intentar el usuario introducir la cantidad de hijos, el sistema debe mostrar un mensaje de error impidiendo al usuario escribir.	Hijos
Introducir la madre.		Al intentar el usuario introducir la madre, el sistema no debe permitir la escritura.	Madre
Introducir el padre.		Al intentar el usuario introducir el padre, el sistema no debe permitir la escritura.	Padre

2.5.3.4 Interfaz Generales

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Selecciona la opción color de piel.		El sistema actualice el color de piel que se desea poner al individuo, ya sea blanca, negra o mestiza.	Color de piel
Selecciona la opción comentario.		El sistema permita introducir datos adicionales de interés sobre el individuo.	Comentarios
Selecciona el botón *Aceptar.*		El sistema guarda el comentario que se le hizo a ese individuo y si existen cambios son guardados igualmente.	
Selecciona el botón *Cancelar.*		El sistema cancela el comentario sin efectuar ningún cambio.	

Selecciona la opción Dirección: **Calle:** José Martí. **Municipio:** Mayarí. **Número:** 1532 **País:** Cuba **Provincia:** Holguín		El sistema permite introducir la dirección satisfactoriamente.	Dirección
	Selecciona la opción Dirección: **Calle:** José_"-58 Martí. **Municipio:** Mayarí. **Número:** 1532 **País:** Cuba **Provincia:** Holguín	El sistema debe mostrar un mensaje de error no permitiendo escribir cualquier tipo de caracter, debiendo especificar que sólo deben introducirse letras, puntos, comas y números.	
	Selecciona la opción Dirección: **Calle:** José Martí. **Municipio:** Mayarí_$58. **Número:** 1532 **País:** Cuba **Provincia:** Holguín	El sistema debe mostrar un mensaje de error no permitiendo escribir cualquier tipo de caracter, debiendo especificar que sólo deben introducirse letras, puntos y comas.	
	Selecciona la opción Dirección: **Calle:** José Martí. **Municipio:** Mayarí. **Número:** 1532() lateral. **País:** Cuba **Provincia:** Holguín	El sistema debe mostrar un mensaje de error no permitiendo escribir cualquier tipo de caracter, debiendo especificar que sólo deben introducirse números.	

	Selecciona la opción Dirección: **Calle:** José Martí. **Municipio:** Mayarí. **Número:** 1532 **País:** Cuba_45 **Provincia:** Holguín	El sistema debe mostrar un mensaje de error no permitiendo escribir cualquier tipo de caracter, debiendo especificar que sólo deben introducirse letras.	
	Selecciona la opción Dirección: **Calle:** José Martí. **Municipio:** Mayarí. **Número:** 1532 **País:** Cuba_45 **Provincia:** Holguín_54	El sistema debe mostrar un mensaje de error no permitiendo escribir cualquier tipo de caracter, debiendo especificar que sólo deben introducirse letras.	
Edad: 54.		El sistema introduce la edad satisfactoriamente.	Edad
	Edad: 54_uh	El sistema no permite introducir la edad satisfactoriamente, mostrando un mensaje de error impidiendo introducir valores erróneos.	
Selecciona la fecha de la muerte: antes de la actual.		El sistema permite escoger la fecha de la muerte (en caso de que sea fallecido) marcando con una ralla negra al individuo.	Fecha de Muerte
	Selecciona la fecha de la muerte: después de la actual.	El sistema permite escoger la fecha de la muerte no marca con una ralla negra al individuo, fecha incorrecta.	
Selecciona la fecha de nacimiento: antes de la actual.		El sistema permite escoger la fecha de nacimiento del individuo satisfactoriamente.	Fecha de Nacimiento

	Selecciona la fecha de nacimiento: después de la actual.	El sistema no permite escoger la fecha de nacimiento del individuo, fecha incorrecta.	
Selecciona la opción intento de suicidio.		El sistema permite escoger si el individuo ha cometido intento de suicidio o no.	Intento de suicidio
Nombre: Eduardo		El sistema debe introducir el nombre satisfactoriamente.	Nombre
	Nombre: Eduardo_45	El sistema no debe permitir introducir el nombre satisfactoriamente, existen caracteres no permitidos.	
Primer apellido: Calderín		El sistema debe introducir el apellido satisfactoriamente.	Primer apellido
	Primer apellido: Calderín_48	El sistema no debe permitir introducir el apellido satisfactoriamente, existen caracteres no permitidos.	
Segundo apellido: Camilo		El sistema debe introducir el apellido satisfactoriamente.	Segundo apellido
	Segundo apellido: Calderín_48	El sistema no debe permitir introducir el apellido satisfactoriamente, existen caracteres no permitidos.	

2.5.3.5 Interfaz Definición

Clases válidas	**Clases inválidas**	**Resultado Esperado**	**Condición de entrada**
Botón *Adicionar*		El sistema permite adicionar una nueva enfermedad que no esté definida con anterioridad.	Enfermedad

	Botón *Adicionar*	El sistema no permite adicionar enfermedades que tengan al mismo nombre, mostrando un mensaje aclarando que ya existe enfermedad con ese nombre.	
Botón *Eliminar*		El sistema permite eliminar la enfermedad que el usuario desee.	
Botón *Terminar*		Se cierra la ventana, guardando los cambios efectuados.	

2.5.3.6 Interfaz Información

Clases válidas	**Clases inválidas**	**Resultado Esperado**	**Condición de entrada**
Selecciona la opción propósito.		El sistema debe indicar si el individuo es el propósito del árbol o no, en caso que sea el propósito lo marca con una flecha.	Propósito

2.5.3.7 Interfaz Maternidad

Clases válidas	**Clases inválidas**	**Resultado Esperado**	**Condición de entrada**
Selecciona la opción embarazada.		El sistema debe activar la opción si el sexo de la persona es femenino y entonces permitir saber si la persona está o no embarazada.	Embarazada

Botón *Nuevo*		Permite agregar un nuevo símbolo al sistema, introduciendo caracterización del símbolo y la descripción del mismo.	Símbolo
Botón *Por Defecto*		Lista todos los símbolos que trae por defecto el sistema, sin incluir los símbolos adicionales que se agregan al sistema.	
Botón *Restablece*		Lista todos los símbolos que trae por defecto el sistema además de los que fueron agregados al sistema.	
Botón *Aceptar*		Guarda todos los cambios efectuados, o sea si fue agregado algún símbolo adicional.	
Botón *Cancelar*		Cierra la ventana sin guardar los cambios efectuados, o sea si fue agregado algún símbolo adicional no lo guarda.	
Botón *Eliminar*		Borra del sistema el símbolo adicional que fue agregado, no debe permitir eliminar los símbolos predefinidos.	
Botón *Adicionar*		El sistema permite agregar un nuevo examen.	Examen prenatal
Botón *Eliminar*		Borra del sistema el examen que fue agregado.	

2.5.3.8 Interfaz Archivo

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Abrir ArboGen		El sistema debe permitir abrir cualquier árbol.	Abrir
Guardar árbol		El sistema debe permitir guardar cualquier árbol, permitiendo poder verlo en otra ocasión.	Guardar
Abrir un nuevo árbol.		El sistema debe permitir abrir una nueva pantalla para crear un nuevo árbol.	Nuevo
Exportar texto		El sistema debe permitir salvar cualquier tipo de texto.	Exportar
Exportar árbol		El sistema debe permitir salvar cualquier árbol.	
Salir de la aplicación		El sistema debe cerrarse automáticamente.	Salir

2.5.3.9 Interfaz Vista

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Ampliar imagen		El sistema debe permitir ampliar la imagen del árbol a gusto del usuario.	Ampliar
Reducir imagen		El sistema debe permitir que se reduzca la imagen del árbol a gusto del usuario.	Reducir

2.5.3.10 Interfaz Datos de la creación

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Dibujado por: Rosa Pérez		El sistema debe permitir escribir el nombre de la persona que está confeccionando el árbol.	Datos de la creación.
	Dibujado por: Rosa Pérez_04	El sistema no debe permitir escribir el nombre de la persona que está confeccionando el árbol, mostrando un mensaje de error impidiendo la utilización de caracteres no autorizados.	
Selecciona la fecha de la creación: antes de la actual o la actual.		El sistema permite escoger la fecha de la creación del árbol satisfactoriamente.	
	Selecciona la fecha de la creación: después de la actual.	El sistema no permite escoger la fecha de la creación del árbol, fecha incorrecta.	
Selecciona última modificación: antes de la actual o la actual.		El sistema permite escoger la fecha de la última modificación satisfactoriamente.	
	Selecciona la última modificación: después de la actual.	El sistema no permite escoger la fecha de la última modificación, fecha incorrecta.	

2.5.3.11 Interfaz Datos de la familia.

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Seleccionar color		El sistema debe permitir seleccionar el color de la enfermedad con que se dibujaran los símbolos.	Color de la enfermedad
Seleccionar enfermedad		El sistema debe permitir seleccionar la enfermedad que afecta a la familia.	Enfermedad
Identificador: 123456		El sistema debe permitir colocar el identificador por el cual se va a reconocer la familia, debe permitir solo números.	Identificador
	Identificador: 123456_op	El sistema no debe permitir colocar el identificador por el cual se va a reconocer la familia, mostrando un mensaje de error pues tiene caracteres no autorizados y excede el número de dígitos.	
Nombre: Familia Pérez		El sistema debe permitir colocar el nombre que identifica a la familia.	Nombre
	Nombre: Familia Pérez_04	El sistema no debe permitir colocar el nombre que identifica a la familia, mostrando un mensaje de error pues tiene caracteres no autorizados.	

2.5.3.13 Interfaz Barra de herramientas

Clases válidas	Clases inválidas	Resultado Esperado	Condición de entrada
Botón *Femenino*		El sistema permite insertar un individuo de sexo femenino al árbol.	Femenino
Botón *Masculino*		El sistema permite insertar un individuo de sexo masculino al árbol.	Masculino
Botón *Desconocido*		El sistema permite insertar un individuo desconocido al árbol.	Desconocido

Para ver las descripciones de cada interfaz Ver Anexo II.

2.6 Conclusiones.

En el transcurso de este capítulo se ha detallado una breve descripción de en que consiste el software Árbol Genealógico para el cual se diseñaron las pruebas de caja blanca y caja negra con el objetivo de corregir los errores que pudiera tener la aplicación.

Para llevar a cabo el diseño y la aplicación de estas pruebas se tienen en cuenta una serie de aspectos que son abordados también en este capítulo como son:

- ✓ La confección del plan de prueba.
- ✓ Las características esenciales a ser probadas.
- ✓ La descripción general de software.

CAPÍTULO 3: RESULTADOS.

3.1 Introducción.

Luego de la aplicación de las pruebas se procede a la realización de una evaluación de los resultados obtenidos donde se exponen cada uno de los detalles de las pruebas así como los errores que fueron detectados. En este capítulo se exponen de manera clara y entendible un resumen de todos los fallos que se detectaron a la hora de aplicar las pruebas de caja blanca y caja negra al software Árbol Genealógico así como las recomendaciones que se hacen al respecto.

3.2 Análisis de los resultados.

Un requisito fundamental a la hora de realizar las pruebas es la documentación del software que debe tener claramente la descripción del análisis, el diseño, casos de uso con todas sus descripciones, dígase flujo de eventos tanto normal como alterno, son importantes también los resultados de la ejecución del software así como el manual de usuario siendo este un documento importantísimo a ser revisado ya que será la cara del producto ante el cliente. El hecho de que la documentación se encuentre completa da la posibilidad de tener un control estricto de todos los casos de prueba que se van a aplicar ya sea para las pruebas de caja blanca o caja negra.

Los resultados de las pruebas deben ser bien documentados de manera que los desarrolladores del software tengan una guía oficial por la cual regirse a la hora de corregir los errores de cada una de las partes del sistema a la que corresponda antes de que el proyecto pase a una fase de mayor complejidad donde la corrección de errores se hace más compleja y costosa. El orden y la organización a la hora de redactar el documento de los resultados juegan un papel importante, tributa pues a la calidad de nuestro trabajo, en el caso de las pruebas de Caja blanca por cada método, clase, módulo o sistema en general que sea revisado deben exponerse los resultados, de la misma forma para las pruebas de Caja negra deben especificarse bien los resultados de las pruebas de requerimientos y las pruebas de interfaz.

Teniendo en cuenta las particularidades del software Árbol Genealógico los resultados son presentados de la forma antes expuesta aunque existieron dificultades para la realización de estas pruebas las cuales serán mencionadas posteriormente:

✓ La no existencia de documentación que respalde el desarrollo del software dificultó en gran medida el desarrollo de las pruebas de Caja Negra, ya que no existe una guía por la cual pueda guiarse el probador.

✓ No están definidos los casos de uso del software a pesar de no estar el software dividido por módulos, siendo un producto fácil de manipular y entendible a los usuarios.

✓ El manual de usuario no está completo, faltan funcionalidades por ser colocadas y además de no estar muy bien explicado quedan algunas dudas de cómo trabajar con el software con total confianza.

Antes de comenzar a diseñar los casos de prueba para cualquier tipo de software deben tenerse en cuenta tanto los riesgos que se corren a la hora de ejecutar las pruebas como por ejemplo los errores más comunes que suelen ocurrir a la hora de implementar la aplicación, ya sea en la parte del código, la interfaz, la entrada de datos o los requerimientos, considerándose de esta forma un paso de adelanto para el ingeniero de prueba, quién tendría entonces que centrarse en encontrar otros errores no descubiertos antes.

Para ellos existen técnicas muy bien justificadas, una de ellas y muy efectiva a propósito, son las listas de defectos que son bien conocidas y vienen dadas por la experiencia de muchos probadores experimentados que las han publicado para el conocimiento de todos, son generadas teniendo en cuenta la complejidad de algunos métodos que suelen ser bastante comunes, teniendo en cuenta siempre las particularidades de cada producto en específico. Siempre podemos partir del hecho de que tener una idea previa de los errores que podamos encontrar en el software puede ser muy beneficioso para todo aquel personal encargado de realizar pruebas de software.

Es importante destacar que la ayuda del programador es fundamental a la hora de ejecutar por lo menos las pruebas de Caja blanca, su criterio siempre será válido, ellos saben dónde se pueden encontrar puntos débiles y complejos que estén más propensos a errores.

3.3 Pasos seguidos para reflejar los resultados.

Muchas pueden ser las estrategias a seguir a la hora de reflejar los resultados, en este caso especifico se describen de forma general partiendo del hecho de que no existen módulos en el proyecto y que se probó de esta manera, primeramente se exponen los resultados de las pruebas de Caja blanca y

seguidamente los de Caja negra, finalmente se realiza una comparación entre los resultados obtenidos y los esperados.

Los pasos que se tienen en cuenta para el análisis de los resultados son los siguientes:

✓ Se identifica el procedimiento o módulo a probar.

✓ Se establece un orden de jerarquía entre los miembros del equipo de prueba partiendo del responsable de la ejecución y demás ingenieros presentes en la pruebas, orientando a cada uno las tareas a cumplir y el nivel de responsabilidad para con el equipo.

✓ El ambiente de configuración en el que se van a realizar las pruebas, ya sea de software, hardware, etc.

✓ Se confecciona una tabla resumen, para las pruebas de caja blanca con el siguiente formato, donde la descripción debe ser explicada lo más claramente posible:

Error	Influencia del error	Observaciones

Error: Se especifican los errores que fueron detectados durante la etapa de pruebas.
Influencia del error: Marcar con una X si el error tiene una influencia alta en el proyecto (que provoca daños catastróficos).
Observaciones: Cualquier punto aclaratorio que se considere sea relevante o sugerencia que se desee hacer a los desarrolladores.

✓ Se confecciona una tabla resumen, para las pruebas de caja negra con el siguiente formato, donde la descripción debe ser explicada lo más claramente posible:

Elemento	No.	No conformidad	Aspecto correspondiente	Etapa de la detección

Elemento: Se especifica el tipo de error (ortográfico, de funcionalidad, etc.).

No.: Identificador del error (un número).
No conformidad: Se especifica la no conformidad que fue encontrada o sea el error detectado.
Aspecto correspondiente: Se describe detalladamente la no conformidad, especificando su localización y cualquier otro aspecto relevante.
Etapa de la detección: En que etapa del ciclo de vida del proyecto se detectó el error.

✓ Se registran los riesgos no previstos, realizando un análisis de ellos.

✓ Se desarrolla un resumen de todo el proceso de prueba incluyendo la relación de los errores que fueron detectados con mayor frecuencia.

3.4 Resultados de las pruebas de Caja negra.

Las pruebas se ejecutaron utilizando una PC con sistema operativo Windows XP, con al menos 64 Mbyte de RAM, el software fue instalado en la misma PC donde se ejecutaron las pruebas. Las pruebas fueron realizadas por un probador dentro del equipo de desarrollo, al no estar documentados los casos de uso, los casos de pruebas se diseñaron para las entradas y las salidas, se probó la funcionalidad de la interfaz y que cumpliera con los requerimientos, de esta forma se exponen los resultados.

Resultados del sistema.

Elemento	No.	No conformidad	Aspecto correspondiente	Etapa de la detección
Error ortográfico	1	A la palabra *Vacío* le falta la tilde.	En la aplicación en el menú de la parte derecha en el subtítulo de Estado genético.	Diseño y aplicación de la prueba.
Error ortográfico	2	A la palabra *Heterocigótico* le falta la tilde.	En la aplicación en el menú de la parte derecha en el subtítulo de Estado genético, Símbolo, al ser desplegado aparece la palabra heterocigótico.	Diseño y aplicación de la prueba.

Aplicación	3	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en el subtítulo de Dirección, Calle se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	4	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en el subtítulo de Dirección, Municipio se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	5	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en el subtítulo de Dirección, País se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	6	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en el subtítulo de Dirección, Provincia se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	7	Acepta cualquier fecha escogida.	En la aplicación en el menú de la parte derecha en Generales, Fecha de nacimiento, se puede escoger cualquier fecha después de la actual, cuando el sistema no debe permitir realizar esta acción.	Diseño y aplicación de la prueba.

Aplicación	8	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en Generales, Nombre, se puede escribir cualquier tipo de aplicación y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	9	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en Generales, Primer apellido, se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	10	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en Generales, Segundo apellido, se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Error ortográfico	11	Error ortográfico en la palabra *Caracterización.*	En definición, opción símbolo, para insertar nuevo símbolo hay que especificar la caracterización del símbolo y está mal escrita.	Diseño y aplicación de la prueba.
Error ortográfico	12	Errores ortográficos en la ventana de Redefinir símbolo.	En definición, opción símbolo, en la lista de todos los símbolos están mal escritas las palabras: vacío, heterocigótico. Además si todas las palabras están escritas con mayúsculas pues entonces se debe seguir ese patrón.	Diseño y aplicación de la prueba.

Error ortográfico	13	Error ortográfico en la ventana Redefinir símbolo.	Está mal escrita la palabra símbolos que da nombre a la ventana.	Diseño y aplicación de la prueba.
Aplicación	14	Acepta cualquier tipo de caracter.	En la aplicación en el menú de la parte derecha en Datos de la creación, Dibujado por, se puede escribir cualquier tipo de caracter y es aceptado por el sistema.	Diseño y aplicación de la prueba.
Aplicación	15	No muestra error	En la aplicación en el menú de la parte derecha en Datos de la creación, fecha de la creación al poner una fecha después de la fecha actual no muestra ningún error diciendo que la fecha no es válida, porque se supone que no esté creado aún el árbol.	Diseño y aplicación de la prueba.
Error ortográfico	16	Error ortográfico en el submenú de la derecha de aplicación en Datos de la creación.	En el submenú de la derecha de la aplicación en Datos de la creación esta mal escrita la palabra creación.	Diseño y aplicación de la prueba.
Aplicación	17	No muestra error	En la aplicación en el menú de la parte derecha en Datos de la creación, fecha de la última modificación al poner una fecha después de la fecha actual no muestra ningún error diciendo que la fecha no es válida, porque se supone que no este creado aún el árbol.	Diseño y aplicación de la prueba.

Error ortográfico	18	Error ortográfico en el submenú de la derecha de aplicación en Datos de la creación.	En el submenú de la derecha de la aplicación en Datos de la creación esta mal escrita la palabra última.	Diseño y aplicación de la prueba.
Aplicación	19	No muestra error	En la aplicación en el menú de la parte derecha en Datos de la familia, no muestra error cuando se introducen caracteres no autorizados como símbolos y números.	Diseño y aplicación de la prueba.

3.5 Resultados de las pruebas de Caja blanca.

Estas pruebas se ejecutaron utilizando una PC con sistema operativo Windows XP, con al menos 64 Mbyte de RAM, debido a la no existencia de documentación del software no se utilizó la herramienta de la Suite del Racional, *Racional Test*, para la realización de las pruebas, de manera que las pruebas fueron realizadas de forma manual probando directamente del *Microsoft Visual Studio 2005*. Los casos de pruebas que fueron diseñados por cada clase depurando cada uno de los procedimientos o métodos por separado, las pruebas fueron ejecutadas por un solo probador.

Resultados del sistema.

Error	Influencia del error	Observaciones
✓ Cuando se trata de actualizar algún elemento y que posteriormente se muestren sus datos, se genera un error que no es manipulado por el sistema.	X	✓ El código debe estar lo más comentado posible, esta acción facilita el trabajo de los ingenieros de prueba y además de que en el equipo de desarrollo pueden ocasionarse bajas.

✓ En ocasiones, no se pueden deshacer cambios realizados al sistema que son solicitados por el usuario, y se genera un error que no es manipulado por el sistema.	X	✓ En los métodos que están implementados existen fragmentos de código que están comentado por lo que se sugiere que si no van a ser utilizados pues que sean eliminados. ✓ Debe revisarse porque razón en ocasiones existen errores en el tiempo de ejecución de los procedimientos, ya que este aspecto atenta contra la estabilidad del mismo. ✓ El lenguaje en el que está desarrollado el proyecto debe ser estándar, no puede pasar que los mensajes de error estén por ejemplo en inglés y que en el proyecto estén en español. ✓ Revisar detalladamente los errores que se muestran a la hora de compilar el programa.
✓ Existe problemas a la hora de que algunas funciones que están implementadas se activan y desactivan cuando es debido o sea el tiempo de ejecución es aproximadamente de 20 a 25 min.	X	
✓ A la hora de que algún campo muestre los datos que contiene no lo hace en el tiempo de ejecución estipulado (2 a 3 min.) se tarda mucho y en ocasiones se produce pérdida de datos.	X	
✓ Los mensajes de error que se producen en la aplicación están en inglés y no son para nada explicativos, el usuario no sabe a que se debe el error.	X	
✓ Existen errores que no coinciden con el campo para el cual está validado que se muestre ese error.	X	
✓ Existen funciones que deben obligatoriamente pedir confirmación para su ejecución por ejemplo eliminar, y sin embargo no sucede así.	X	

✓ Existen errores que muestra el compilador sin embargo no tiene aparentemente ninguna influencia en la correcta ejecución del proyecto.	X	

3.6 Resultados del sistema en general.

En sentido general luego de analizar cada uno de los resultados obtenidos en la ejecución de las pruebas de Caja blanca y Caja negra se puede resumir diciendo que los defectos detectados tienen gran importancia ya que pueden conllevar a que el sistema un algún momento no funcione o emita datos erróneos atentando contra la confiabilidad del software, errores como por ejemplo de procedimiento que pueden generar fallas en el sistema mostrando salidas no esperadas por los usuarios propiciando esto que el sistema se comporte de forma inestable, otros defectos que fueron encontrados mientras se ejecutaron las pruebas no fueron pues tan relevantes debido que se produjeron errores dentro del código fuente que no tributaron a la ocurrencia de fallas en la aplicación, al probar la funcionalidad a la cual respondía el procedimiento ésta no sufría ninguna alteración.

Los errores de interfaz producidos no todos son de gran relevancia, aunque hay que tener cuidado con algunos como las faltas de ortografía, que de cierta forma no conllevan a un error catastrófico que provoquen fallos en la ejecución del sistema pero sin embargo atentan contra la estética y la calidad del software desarrollado. Por otro lado el manual de usuario debiera ser más específico, existen funcionalidades que no son abordadas, pasa de la misma forma con la ayuda del software ya que éstas son la únicas posibilidades que pueden guiar al usuario en la utilización del mismo. La carencia de mensajes de error cuando sea necesario es un aspecto importante a considerar puesto que pueden introducirse datos o información errónea al sistema.

3.7 Comparación entre resultados.

El software Árbol Genealógico en la actualidad se encuentra instalado en el centro de Genética Médica por tal razón se esperaba tuviera muy pocos errores, no se puede decir que ninguno porque no hay software alguno que carezca de defectos, sin embargo la práctica nos reafirma una vez más que el ciclo de vida del software es inviolable aunque muchos se empeñen en dar más importancia a una fase que a otra, sin darse cuenta que todas juegan un papel importantísimo en este proceso, la etapa de prueba por ejemplo comienza desde la fase de inicio llevando a cabo un control estricto de cada

cambio que se produzca en el software para tratar de evitar males mayores, sin embargo muchos hacen caso omiso de este detalle y por eso llegan a perder entonces el prestigio que tiene la institución para con los clientes.

Realizando entonces una breve comparación entre los errores que se esperaba estuvieran presentes en el proyecto y los que realmente fueron encontrados, se puede decir que el software cumple con cada una de las funcionalidades previstas, sin embargo se producen errores dentro del código que no eran esperados, siendo entregados estos para ser revisados para la puesta en marcha de posteriores versiones. Las pruebas de Caja blanca en sentido general fueron bastante satisfactorias ya que no se pronosticaron grandes errores, cumpliendo así con los pronósticos.

En las pruebas de caja negra se espera que el software cumpliera con los requerimientos especificados obteniendo resultados satisfactorios en este sentido pero no se esperaba sin embargo que existieran errores ortográficos que pueden parecer insignificantes pero no lo son, incluso errores de no reconocimiento de entradas inválidas que si pueden de algún modo llegar a desestabilizar el sistema, en estos momentos luego de haber entregado los resultados obtenidos en las pruebas a los desarrolladores del proyecto, se encuentra entonces en la fase de reparación y corrección de errores incluyendo también las fallas que no fueron previstas con anterioridad.

3.8 Conclusiones.

Con la realización de este capítulo se da por concluido todo el proceso de prueba que se ha desarrollado de una manera entendible y fácil de manejar, fueron de esta forma documentados todos y cada uno de los defectos ya sea los que fueron previstos que ocurrieran como los que se produjeron en el transcurso de todo el proceso.

Estadísticamente podemos concluir diciendo que el 90 % de los resultados obtenidos fueron previstos con anterioridad por el equipo de trabajo y el 10% surgieron durante el transcurso de las pruebas. Luego de haber terminado de probar el software Árbol Genealógico, se puede afirmar que no cumple con las normas de calidad por las cuales se rige la Universidad para realizar la entrega del producto al cliente podemos citar entonces:

- ✓ El software no cuenta con documentación alguna.
- ✓ No fue revisado por el grupo de calidad de la facultad antes de ser desplegado, lo que trajo consigo que haya sido entregado al cliente con defectos.

Naturalmente el estudio realizado servirá para que los desarrolladores lo tengan en cuenta a la hora de confeccionar una versión superior y además les servirá de experiencia para el mejoramiento de su trabajo profesional.

CONCLUSIONES.

El vertiginoso desarrollo de la industria del *software,* y la necesidad existente de que los productos sean más confiables y consistentes hace que crezca cada vez más la exigencia por parte de los clientes y usuarios, los sistemas desarrollados deben cumplir con las normativas además de que en ellos deben estar bien aplicadas las técnicas de ingeniería de software y por ende bien definida y aplicada la etapa de prueba que es donde se dictamina si el software puede pasar a manos de los cliente o no, teniendo en cuenta que cumpla con los niveles de calidad requeridos.

Luego de planificarse y aplicarse posteriormente las pruebas al producto ArboGen se llegaron a las siguientes conclusiones:

✓ En la industria del *software* existe una gran competencia y por ende entre las propias empresas que están inmersas en este desarrollo tecnológico por ello tiene gran influencia la calidad de los productos y por supuesto que influye considerablemente en la aceptación que puedan tener éstos por parte de los usuarios.

✓ Se logró una exitosa comunicación entre el equipo de desarrollo del software y los probadores compartiéndose así conocimientos validos para ambos.

✓ Se lograron aplicar correctamente las técnicas de pruebas siguiendo una estrategia fiable para su aplicación y los pasos establecidos.

✓ Se desarrollaron los casos de pruebas de calidad utilizando las técnicas de Caja Blanca y Caja Negra, las cuales validaron el diseño de prueba propuesto para el producto.

✓ Las pruebas se realizaron con un alto nivel de responsabilidad y por ende de calidad a todo el producto comprobando posteriormente los resultados que realmente se obtuvieron con los pronosticados.

✓ A raíz de toda la investigación realizada se logró elaborar una adecuada documentación de todo el proceso de prueba que servirá sin lugar a dudas para la corrección de los errores así como para su utilización en otros proyectos de bioinformática.

✓ El producto ArboGen luego de haber pasado por todo este proceso poseerá mayor calidad y confiabilidad.

✓ Se ha cumplido con el objetivo propuesto al inicio del trabajo de diploma, se diseñaron y aplicaron correctamente las pruebas de *software* al producto Árbol Genealógico, logrando con la posterior eliminación de los errores mayor calidad.

✓ Las pruebas que tienen mayor aplicabilidad para medir la calidad del software son: las de Caja Blanca, Caja Negra, Seguridad y Sistema.

✓ El cliente recibirá un producto sólido que cumple con los requisitos previstos y además en la UCI se desarrolla por primera vez la aplicación de pruebas de *software* de Bioinformática.

RECOMENDACIONES.

Luego de haber concluido con todo el proceso de prueba y de comprender de cuan importante es la detección de errores a tiempo, se recomienda:

✓ Que los desarrolladores del sistema corrijan los errores detectados lo más pronto posible.

✓ Que las pruebas realizadas no se queden en esta sola iteración, la continuidad de las pruebas es fundamental hasta que el producto quede libre de defectos.

✓ La realización de otras pruebas al producto, como por ejemplo pruebas de sistema (rendimiento, usabilidad, seguridad) e integración.

✓ Utilizar este trabajo de diploma como bibliografía para posteriores trabajos de diplomas, además de que puede servir como complemento para las asignaturas de Ingeniería de Software y Administración de Empresas, que abordan temas de calidad, en la Universidad de Ciencias Informáticas.

✓ Que los desarrolladores tomen conciencia de cuan importante es la etapa de prueba y apliquen de esta forma las técnicas y estrategias de prueba que aquí se explican y las apliquen en sus proyectos.

✓ Hacer llegar a todos los proyectos este trabajo investigativo con el fin de que conozcan qué tipo de documentación es fundamental para el desarrollo de cualquier proyecto.

REFERENCIAS BIBILOGRÁFICAS.

1. **Bedini G, Alejandro.** *Calidad Tradicional y de Software.*

2. **Pressman, Roger S.** *Ingenieria del software.Un enfoque práctico.* Cuarta. s.l. : McGraw Hill, 1998.

3. **Laitinen, Mauri, Fayad, Mohamed and Ward, Robert.** *Software Engineering in the Small. Comunications of the ACM.* 2000.

4. **Sastre, Benjamín del.** Portal Universi Argentina. [En línea] 2006.

5. **Febles, Ailin.** *Modelo de Referencia para la Gestión de Configuración en la pequeña y mediana empresa de software.* Ciudad de la Habana : s.n., 2001. Tesis doctoral.

6. **Hall, Tracy, Rainer, Austen and Baddoo, Nathan.** *Implementing Software Process Improvement: An Empirical Study. Software Process:Improvement and Practice.* 2002.

7. **Kulpa, Margaret and Johnson, Kent.** *Interpreting the CMMI: A Process Improvement Approach.* 2003.

8. **McFeeley, Robert.** *IDEAL: A Users Guide for Software Process Improvement,Handbook.* Pittsburgh : Software Engineering Institute, Carnegei Mellon University, 1996.

9. **Di Mónaco, Silvia y Vitali, Silvio.** *El Modelo IDEAL para implementar CMMI.* [PowerPoint] s.l. : Centro de Calidad en Tecnologias de la información, 2005.

10. **Calvo Manzano, J.A.** Experiences in the Application of Software Process Improvement in SMES. s.l. : Software Quality Journal, 2002. págs. 261-273.

11. **Straub, Pablo.** *El costo de la calidad.* 2006.

12. *Estimación del coste de la calidad del software a través de la simulación del proceso de desarrollo.* **Ruiz, Mercedes y Ramos, Isabel.** 1, Revista Colombiana de computación, Vol. II, págs. 75-87.

13. **Guzmán, Adolfo.** *Medición de la calidad del Software.* s.l. : Centro de Investigación en Computación.

14. **Jacobson, I., Booch, G. y Rumbaugh, J.** *Proceso Unificado de Desarrollo de Software.* Vol. I.

15. **Patricio, Letelier.** *Pruebas del Software.* Valencia : Departamento de Sistemas Informáticos y Computación.

BIBILOGRAFÍA.

1. *Juan Manuel. Cueva,* http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF.

2. Un enfoque actual sobre al calidad del software, *Oscar M. Fernández Carrasco, Delba García León y Alfa Beltrán Benavides,* http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.

3. Calidad en ingeniería del software, dmi.uib.es/~bbuades/calidad/index.htm

4. Calidad del software: gestión eficaz de los procesos de desarrollo de software para satisfacer la calidad, http://www.fundacion.unavarra.es/CursosdeVerano/Calidad_del_Software.pdf

5. Gestión de al calidad del software, http://www-306.ibm.com/software/info/ecatalog/es_ES/rational/SW730.html

6. Modelos de calidad de software y software libre, *Ernesto Quiñones A.***,** http://www.egsoft.net/presentas/modelos_de_calidad_y_software_libre.pdf

7. Control de la calidad del software, http://www.sdl.com/es/services/services-sdl-language-services/services-sdl-software-qa.htm.

8. Estimación del coste de la calidad del software a través de la simulación del proceso de desarrollo, http://www.unab.edu.co/editorialunab/revistas/rcc/pdfs/r21_art6_c.pdf.

9. Calidad y testeo de software, http://www.iti.upv.es/about_iti/lines/calidad_st.

10. Mitos, creencias y supersticiones sobre la calidad del software y su enseñanza, Adolfo Guzmán, http://www.cic.ipn.mx/aguzman/papers/165%20mitos,%20creencias%20y%20superst%20sobre%20la%20calidad%20del%20software.pdf

11. Pruebas de software, http://lsi.ugr.es/~ig1/docis/pruso.pdf

12. Pruebas de software terminado, http://gbtcr.chileforge.cl/info_web/node139.html

13. Aplicación practica del diseño de pruebas de software a nivel de programación, http://www.willydev.net/descargas/oguzman-diseno_pruebas.pdf

14. Pruebas de software, http://lml.ls.fi.upm.es/ftp/ed2/0203/Apuntes/pruebas.ppt

15. Pruebas de software, http://www.greensqa.com/archivos/Servicio%20Pruebas%20de%20Software.pdf

16. Pruebas del software, http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf

17. http://www.seccperu.org/?q=node/389

18. Bedini G, Alejandro. *Calidad Tradicional y de Software.*

19. Pressman, Roger S. *Ingenieria del software.Un enfoque práctico.* Cuarta. s.l. : McGraw Hill, 1998.

20. Laitinen, Mauri, Fayad, Mohamed and Ward, Robert. *Software Engineering in the Small. Comunications of the ACM.* 2000.

21. Sastre, Benjamín del. Portal Universi Argentina. [En línea] 2006.

22. Febles, Ailin. *Modelo de Referencia para la Gestión de Configuración en la pequeña y mediana empresa de software.* Ciudad de la Habana : s.n., 2001. Tesis doctoral.

23. Hall, Tracy, Rainer, Austen and Baddoo, Nathan. *Implementing Software Process Improvement: An Empirical Study. Software Process:Improvement and Practice.* 2002.

24. Kulpa, Margaret and Johnson, Kent. *Interpreting the CMMI: A Process Improvement Approach.* 2003.

25. McFeeley, Robert. *IDEAL: A Users Guide for Software Process Improvement,Handbook.* Pittsburgh : Software Engineering Institute, Carnegei Mellon University, 1996.

26. Di Mónaco, Silvia y Vitali, Silvio. *El Modelo IDEAL para implementar CMMI.* [PowerPoint] s.l. : Centro de Calidad en Tecnologias de la información, 2005.

27. Calvo Manzano, J.A. Experiences in the Application of Software Process Improvement in SMES. s.l. : Software Quality Journal, 2002. págs. 261-273.

28. Straub, Pablo. *El costo de la calidad.* 2006.

29. *Estimación del coste de la calidad del software a través de la simulación del proceso de desarrollo.* **Ruiz, Mercedes y Ramos, Isabel.** 1, Revista Colombiana de computación, Vol. II, págs. 75-87.

30. Guzmán, Adolfo. *Medición de la calidad del Software.* s.l. : Centro de Investigación en Computación.

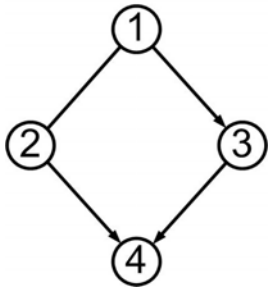
31. Jacobson, I., Booch, G. y Rumbaugh, J. *Proceso Unificado de Desarrollo de Software.* Vol. I.

32. Patricio, Letelier. *Pruebas del Software.* Valencia : Departamento de Sistemas Informáticos y Computación.

ANEXO I

Método DarmeElIdenti

```
public void DarmeElIdenti(Persona persona)
  {
        if(listaDegeneraciones.Count==0)  1
         {
                ArrayList cantidad=new ArrayList();
                cantidad.Add(cantidad.Count+1);
                listaDegeneraciones.Add(cantidad);     2
                persona.Generacion=1;
                persona.Numero=1;
         }
        else
         {
                ArrayList cantidad=(listaDegeneraciones[0] as ArrayList);
                cantidad.Add(cantidad.Count+1);
                persona.Generacion=1;                                  3
                persona.Numero=cantidad.Count;
         }
}  4
```



Caminos básicos	**Complejidad**
C1:1-2-4	**ciclomática**
C2:1-3-4	V(G)= A-N+2
	V(G)= 4 - 4 + 2 = 2 = R2

Camino 1:1-2-4

Caso de prueba: Identificador del primer individuo.

Entrada: persona (Nombre = Ana), Generacion=1, Numero=1

Resultado Esperado: Se comprueba que la lista de generaciones esté vacía llamando al método contar de la clase listaDegeneraciones, se adiciona la persona, devolviendo el identificador del individuo en el árbol en la primera posición.

Condiciones: Que exista persona.

Camino 2:1-3-4

Caso de prueba: Identificador del individuo.

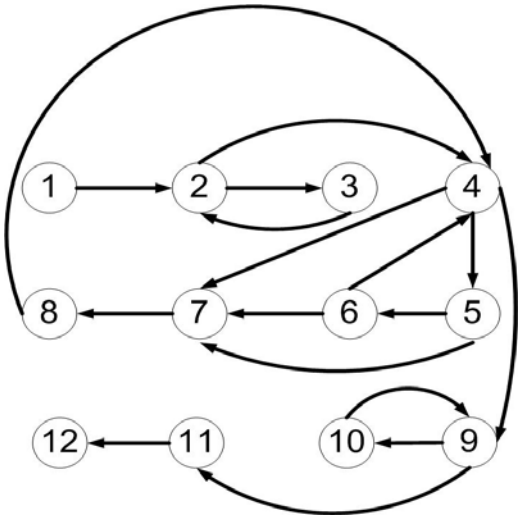
Entrada: persona (Nombre = Ana), Generacion=1, Numero= 3

Resultado Esperado: Se comprueba que la lista de generaciones no esté vacía, devuelve el identificador que tiene el individuo en el árbol en la posición que se encuentre.

Condiciones: Que exista persona.

Método Remover

```
public void Remover (Persona persona)
        {
            DesconectarDeLosPadres (persona);  1
            for (int i=0; i < persona.Relaciones.Count; i++)   2
                persona.Relaciones[i].BorrarRelacion (persona);  3
            for (int i=0; i < persona.Hijos.Count; i++)   4
                {
                  if (persona.Sexo == Sexos.Masculino)  5
                        persona.Hijos[i].Padre = null;  6
                  if (persona.Sexo == Sexos.Femenino)  7
                        persona.Hijos[i].Madre = null;  8
                }
           for (int i=0; i< persona.Gemelos.Count; i++)   9
           {
            persona.BorrarGemelo (persona.Gemelos[i]);  10
           }
            personas.Remove (persona);  11
        } 12
```



Caminos básicos	**Complejidad ciclomática**
C1:1-2-4-9-11-12	V(G)= A-N+2
C2:1-2-4-9-10-9-11-12	V(G)= 17- 12 + 2 = 8= R7
C3:1-2-4-5-6-4-9-11-12	
C4:1-2-4-7-8-4-9-11-12	
C5:1-2-4-5-6-4-7-8-4-9-11-12	
C6:1-2-3-2-4-9-11-12	
C7:1-2-3-2-4-5-6-4-7-8-4-9-10-9-11-12	

Camino 1: 1-2-4-9-11-12

Caso de prueba: Eliminar.

Entrada: persona (Nombre = Juan), Relaciones = 0, Hijos = 0, Gemelos = 0

Resultado Esperado: Desconecta la persona de los padres, se elimina la persona del árbol.

Condiciones: Exista persona.

Camino 2: 1-2-4-9-10-9-11-12

Caso de prueba: Desconectar del gemelo.

Entrada: persona (Nombre = Juan), Relaciones = 0, Hijos = 0, Gemelos = 1

Resultado Esperado: Desconecta la persona de los padres y de su gemelo, se elimina la persona del árbol.

Condiciones: Exista persona.

Camino 3: 1-2-4-5-6-4-9-11-12

Caso de prueba: Desconectar relaciones.

Entrada: persona (Nombre = Juan), Relaciones = 0, Hijos = 2, Sexo = masculino, Gemelos = 0

Resultado Esperado: Desconecta la relación con los padres, se elimina la relación que exista con los hijos de sexo masculino, se elimina la persona del árbol.

Condiciones: Exista persona.

Camino 4: 1-2-4-7-8-4-9-11-12
Caso de prueba: Desconectar.
Entrada: persona (Nombre = Juan), Relaciones = 0, Hijos = 2, Sexo = femenino, Gemelos = 0
Resultado Esperado: Desconecta la relación con los padres, se elimina la relación que exista con los hijos de sexo femenino, se elimina la persona del árbol.
Condiciones: Exista persona.

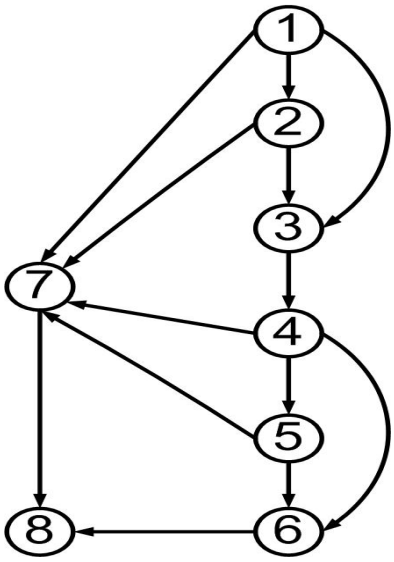
Camino 5: 1-2-4-5-6-4-7-8-4-9-11-12
Caso de prueba: Remover persona.
Entrada: persona (Nombre = Juan), Relaciones = 0, Hijos = 2, Sexo = masculino, Sexo = femenino, Gemelos = 0
Resultado Esperado: Desconecta la relación con los padres, se elimina la relación que exista con los hijos de cualquier sexo, se elimina la persona del árbol.
Condiciones: Exista persona.

Camino 6: 1-2-3-2-4-9-11-12
Caso de prueba: Borrar relaciones.
Entrada: persona (Nombre = Juan), Relaciones = 5, Hijos = 0, Gemelos = 0
Resultado Esperado: Desconecta todas las relaciones del individuo, lo elimina del árbol.
Condiciones: Exista persona.

Camino 7: 1-2-3-2-4-5-6-4-7-8-4-9-10-9-11-12
Caso de prueba: Remover.
Entrada: persona (Nombre = Juan), Relaciones = 5, Hijos = 2, Sexo = masculino, Sexo = femenino, Gemelos = 1
Resultado Esperado: Desconecta la relación con los padres y todas las relaciones que tenga el individuo, se elimina la relación que exista con los hijos de cualquier sexo y de su gemelo, elimina la persona del árbol.
Condiciones: Exista persona.

Método PuedeSerEsposoDe

```
public bool PuedeSerEsposoDe(Persona esposa, Persona esposo)
                {
if( (esposa.Sexo!=Sexos.Femenino) 1 || (esposo.Sexo!=Sexos.Masculino) ) 2
                        return false;  3
if( EsDescendienteDe(esposa,esposo) 4 || EsDescendienteDe(esposo,esposa) )  5
                        return false;   6
                    return true;   7
                }  8
```



Caminos básicos	**Complejidad**
C1:1-7-8	**ciclomática**
C2:1-2-7-8	V(G)= A-N+2
C3:1-2-3-4-6-8	V(G)= 13 - 8 + 2 = 7 = R7
C4:1-2-3-4-7-8	
C5:1-3-4-5-6-8	
C6:1-2-3-4-5-7-8	
C7:1-2-3-4-5-6-8	

Camino 1:1-7-8

Caso de prueba: Existe relación comparando sexo de la esposa.

Entrada: esposa (Nombre = Anabel), Sexo = femenino

Resultado Esperado: Si el individuo es esposa se comprueba el sexo, si es femenino se define que puede existir relación.

Condiciones: Exista persona para establecer relación.

Camino 2:1-2-7-8

Caso de prueba: Existe relación comparando sexo de la esposa y del esposo.

Entrada: esposa (Nombre = Anabel), esposo (Nombre = Mario), Sexo esposa = femenino, Sexo esposo = masculino

Resultado Esperado: Si el individuo es esposa se comprueba el sexo, verificando que sea femenino, si el individuo es esposo se comprueba el sexo, verificando que sea masculino y luego se define que puede existir relación.
Condiciones: Exista persona para establecer relación.

Camino 3:1-2-3-4-6-8
Caso de prueba: Existe relación comparando sexo de la esposa, del esposo y si esposa es descendiente de esposo.
Entrada: esposa (Nombre = Anabel), esposo (Nombre = Mario), Sexo esposa = masculino, Sexo esposo = femenino
Resultado Esperado: Si el individuo es esposa se comprueba el sexo, si es masculino se define que no puede haber relación entre esas personas o si el individuo es esposo se comprueba el sexo, si es femenino se define que no puede haber relación entre esas personas, si esposa es decendiente de esposo no puede haber relación de pareja entre esas personas.
Condiciones: Exista persona para establecer relación.

Camino 4:1-2-3-4-7-8
Caso de prueba: Existe relación comparando sexo de la esposa, del esposo y si esposa es descendiente de esposo.
Entrada: esposa (Nombre = Anabel), esposo (Nombre = Mario), Sexo esposa = masculino, Sexo esposo = femenino
Resultado Esperado: Si el individuo es esposa se comprueba el sexo, si es masculino se define que no puede haber relación entre esas personas o si el individuo es esposo se comprueba el sexo, si es femenino se define que no puede haber relación entre esas personas, si esposa no es decendiente de esposo puede haber relación de pareja entre esas personas.
Condiciones: Exista persona para establecer relación.

Camino 5: 1-3-4-5-6-8
Caso de prueba: No existe relación comparando sexo de la esposa, y si esposo y esposa son descendientes.
Entrada: esposa (Nombre = Anabel), esposo (Nombre = Mario), Sexo esposa = masculino
Resultado Esperado: Si el individuo es esposa se comprueba el sexo, si es masculino se define que no puede haber relación entre esas personas, si esposa es decendiente de esposo o si esposo es descendiente de esposa no puede existir relacion esntre esos individuos.

Condiciones: Exista persona para establecer relación.

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

Caso de prueba: No existe relación comparando sexo de la esposa y del esposo, y si la esposa es descendiente o si el esposo es descendiente.

Entrada: esposa (Nombre = Anabel), esposo (Nombre = Mario), Sexo esposa = masculino, Sexo esposo = femenino

Resultado Esperado: Si el individuo es esposa se comprueba el sexo, si es masculino se define que no puede haber relación entre esas personas o si el individuo es esposo se comprueba el sexo, si es femenino se define que no puede haber relación entre esas personas, si esposa no es decendiente de esposo o si esposo no es descendiente de esposa se define que puede existir relación entre esos individuos.

Condiciones: Exista persona para establecer relación.

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

Caso de prueba: No existe relación comparando sexo de la esposa y del esposo y si la esposa es descendiente o si el esposo es descendiente.

Entrada: esposa (Nombre = Anabel), esposo (Nombre = Mario), Sexo esposa = masculino, Sexo esposo = femenino

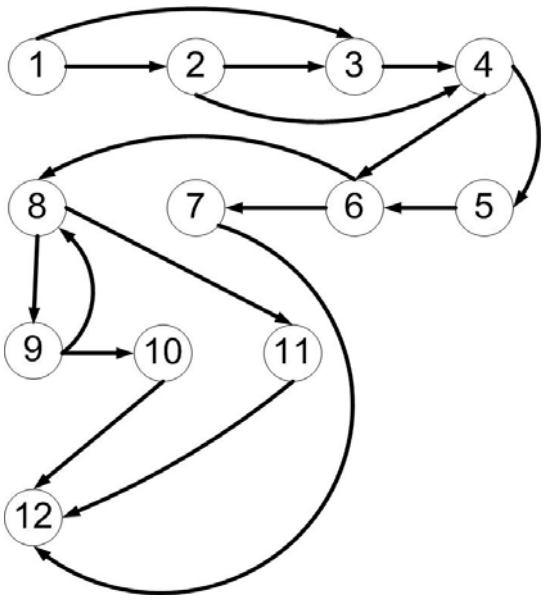
Resultado Esperado: Si el individuo es esposa se comprueba el sexo, si es masculino se define que no puede haber relación entre esas personas o si el individuo es esposo se comprueba el sexo, si es femenino se define que no puede haber relación entre esas personas, si esposa es decendiente de esposo o si esposo es descendiente de esposa se define que no puede existir relación entre esos individuos.

Condiciones: Exista persona para establecer relación.

Método EsDescendienteDe

```
public bool EsDescendienteDe(Persona descendiente, Persona antecesor)
  {
    if ( ( BuscarPersona(descendiente) == -1) 1 || (BuscarPersona(antecesor) == -1) )  2
        throw new Exception("Descendiente o antecesor no pertenece a la familia");  3
    if (descendiente == antecesor)   4
                MessageBox.Show("Antecesor y Descendiente son la misma persona");  5
    if (EsHijoDe(descendiente, antecesor))   6
```

```
                    return true;   7
else
    {
        for(int i=0; i < antecesor.CantidadDeHijos; i++)   8
                    if (EsDescendienteDe(descendiente, antecesor.Hijos[i]))   9
                                return true;   10
        return false;   11
                    }  12
            }
```



Caminos básicos	Complejidad ciclomática
C1:1-3-4-5-6-7-12	V(G)= A-N+2
C2:1-2-4-5-6-7-12	V(G)= 17- 12 + 2 = 7 = R7
C3:1-2-4-6-8-9-8-11-12	
C4:1-2-4-6-8-9-10-12	
C5:1-2-4-6-8-11-12	
C6:1-2-3-4-6-7-12	
C7:1-2-3-4-5-6-8-9-10-12	

Camino 1: 1-3-4-5-6-7-12

Caso de prueba: Es hijo.

Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Juan).

Resultado Esperado: El descendiente o el antecesor no pertenecen a la familia además coincide que el antecesor y el descendiente son la misma persona, pero es descendiente de su antecesor entonces se cumple que el individuo es el hijo.

Condiciones: Que exista individuo descendiente.

Camino 2: 1-2-4-5-6-7-12

Caso de prueba: Pertenecen a la familia.

Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Juan).

Resultado Esperado: Se cumple que ambos individuos pertenecen a la familia, coincide que son la misma persona, pero es descendiente de su antecesor entonces se cumple que el individuo es el hijo.

Condiciones: Que exista individuo descendiente.

Camino 3: 1-2-4-6-8-9-8-11-12

Caso de prueba: No es hijo de ese antecesor.

Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Pedro), CantidadDeHijos antecesor= 0

Resultado Esperado: Recorre la lista de hijos que tiene una persona, o sea del antecesor, luego se verifica que esa persona no es descendiente de alguno de ellos y por tanto noexiste relación de descendencia.

Condiciones: Que existan individuo descendiente.

Camino 4: 1-2-4-6-8-9-10-12

Caso de prueba: Antecesor de un solo hijo.

Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Pedro), CantidadDeHijos antecesor = 1

Resultado Esperado: Recorre una sola vez la lista de hijos que tiene una persona, o sea del antecesor, luego se verifica que esa persona es descendiente de alguno de ellos y por tanto existe relación de descendencia.

Condiciones: Que exista individuo descendiente.

Camino 5: 1-2-4-6-8-11-12

Caso de prueba: No es descendiente.

Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Pedro), CantidadDeHijos antecesor = 0

Resultado Esperado: No se cumple que es descendiente de su antecesor por tanto no es hijo.

Condiciones: No exista descendiente.

Camino 6: 1-2-3-4-6-7-12

Caso de prueba: Es descendiente.

Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Pedro)

Resultado Esperado: Descendiente o antecesor no pertenecen a la familia, pero si es descendiente del antecesor entonces es el hijo.

Condiciones: Que exista individuo descendiente.

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

Caso de prueba: Buscar descendiente

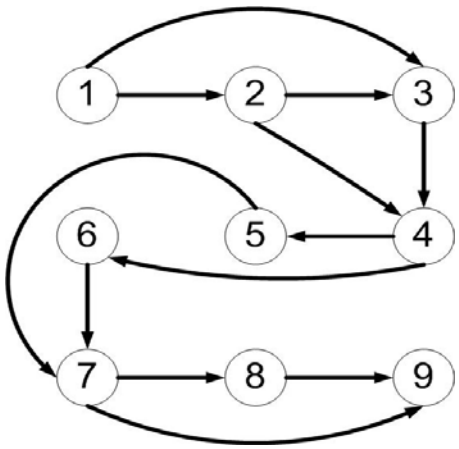
Entrada: descendiente (Nombre = Juan), antecesor (Nombre = Pedro), CantidadDeHijos antecesor = 1

Resultado Esperado: Si el descendiente o el antecesor no pertenecen a la familia y coincide con que son la misma persona el antecesor que el descendiente, devuelve que la persona no es descendiente, pero recorre una sola vez la lista de hijos que tiene esa persona, o sea del antecesor, luego se verifica que esa persona es descendiente de alguno de ellos y por tanto existe relación de descendencia.

Condiciones: Que exista individuo descendiente.

AdicionarPadre

```
public void AdicionarPadre(Persona padre, Persona hijo)
        {
        if ( ( BuscarPersona(padre) == -1) 1 || (BuscarPersona(hijo) == -1) )  2
                throw new Exception("Padre o hijo no pertenece a la familia");   3
                if (hijo.Padre != null)   4
                        throw new Exception("Ya tiene un padre");   5
                hijo.Padre = padre;
                padre.AdicionarHijo(hijo);   6
                 if (hijo.Madre!= null)  7
                        AdicionarEsposo(hijo.Madre, padre);   8
        }  9
```



Caminos básicos	Complejidad ciclomática
C1:1-2-3-4-5-7-9	V(G)= A-N+2
C2:1-3-4-5-7-9	V(G)= 12- 9 + 2 = 5 = R5
C3:1-2-4-5-7-9	
C4:1-2-4-6-7-8-9	
C5:1-2-4-6-7-9	

Camino 1: 1-2-3-4-5-7-9
Caso de prueba: No es padre.
Entrada: Padre = Alberto, hijo = Pedro, Madre = null
Resultado Esperado: Padre o hijo no pertenecen a la familia, verifica además que el hijo ya tiene padre por tanto no se adiciona la relación.
Condiciones: Exista hijo.

Camino 2: 1-3-4-5-7-9
Caso de prueba: No adiciona el padre.
Entrada: Padre = Alberto, hijo = Pedro**,** Madre = null
Resultado Esperado: El padre no pertenece a al familia además el hijo ya tiene un padre por tanto no se adiciona la relación de padre a ese hijo.
Condiciones: Exista hijo.

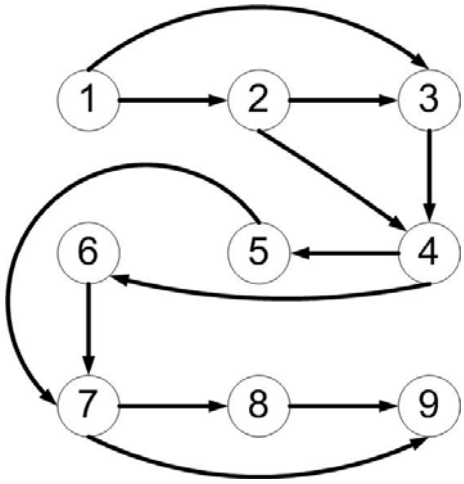
Camino 3: 1-2-4-5-7-9
Caso de prueba: No relaciona al hijo con ese padre.
Entrada: Padre = Alberto, hijo = Pedro, Madre = null
Resultado Esperado: Padre e hijo pertenecen a la familia pero el hijo ya tiene un padre por tanto no se adiciona la relación de padre a ese hijo.
Condiciones: Exista hijo.

Camino 4: 1-2-4-6-7-8-9
Caso de prueba: Adiciona relación con la esposa.
Entrada: Padre = Alberto, hijo = Pedro, Madre = Ana
Resultado Esperado: Padre e hijo pertenecen a la familia, se adiciona la relación de padre con ese hijo y luego se establece la relación con la esposa al padre.
Condiciones: Exista un hijo.

Camino 5: 1-2-4-6-7-9
Caso de prueba: No adiciona relación con la esposa.
Entrada: Padre = Alberto, hijo = Pedro, Madre = Ana
Resultado Esperado: Padre e hijo pertenecen a la familia, se adiciona la relación de padre con ese hijo y luego no se establece la relación con la esposa al padre.
Condiciones: Exista un hijo.

Método AdicionarMadre

```
public void AdicionarMadre(Persona madre, Persona hijo)
        {
          if ( ( BuscarPersona(madre) == -1) 1 || (BuscarPersona(hijo) == -1) )  2
              throw new Exception("Madre o hijo no pertenece a la familia");   3
              if (hijo.Madre != null)   4
                 throw new Exception("Ya tiene una madre");  5
              hijo.Madre = madre;                            6
              madre.AdicionarHijo(hijo);
              if (hijo.Padre!= null)   7
                  AdicionarEsposo(madre, hijo.Padre);   8
      } 9
```



Caminos básicos	**Complejidad**
C1:1-2-3-4-5-7-9	**ciclomática**
C2:1-3-4-5-7-9	V(G)= A-N+2
C3:1-2-4-5-7-9	V(G)= 12- 9 + 2 = 5 =
C4:1-2-4-6-7-8-9	R5
C5:1-2-4-6-7-9	

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

Caso de prueba: No es madre.

Entrada: Madre = Ana, hijo = Pedro, Padre = null

Resultado Esperado: Madre o hijo no pertenecen a la familia, verifica además que el hijo ya tiene madre por tanto no se adiciona la relación.

Condiciones: Exista hijo.

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

Caso de prueba: No adiciona la madre.

Entrada: Madre = Ana, hijo = Pedro, Padre = null

Resultado Esperado: La madre no pertenece a la familia además el hijo ya tiene una madre por tanto no se adiciona la relación de madre a ese hijo.

Condiciones: Exista hijo.

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

Caso de prueba: No relaciona al hijo con esa madre.

Entrada: Madre = Ana, hijo = Pedro, Padre = null

Resultado Esperado: Madre e hijo pertenecen a la familia pero el hijo ya tiene una madre por tanto no se adiciona la relación de madre a ese hijo.

Condiciones: Exista hijo.

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

Caso de prueba: Adiciona relación con el esposo.

Entrada: Madre = Ana, hijo = Pedro, Padre = Alberto

Resultado Esperado: Madre e hijo pertenecen a la familia, se adiciona la relación de madre con ese hijo y luego se establece la relación con el esposo a la madre.

Condiciones: Exista un hijo.

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

Caso de prueba: No adiciona relación con la esposa.

Entrada: Madre = Ana, hijo = Pedro, Padre = Alberto

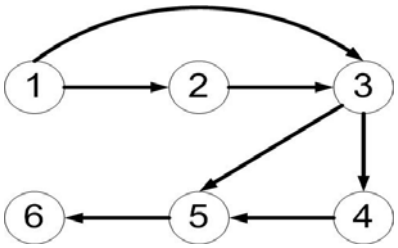
Resultado Esperado: Madre e hijo pertenecen a la familia, se adiciona la relación de madre con ese hijo y luego no se establece la relación con el esposo a la madre.

Condiciones: Exista un hijo.

Método DesconectarDeLosPadres

```
public void DesconectarDeLosPadres(Persona persona)
                {
                        if (persona.Madre != null)   1
                                persona.Madre.BorrarHijo(persona);   2
                        if (persona.Padre != null)   3
```

```
                persona.Padre.BorrarHijo(persona);   4
            persona.Madre = null;  ┐ 5
            persona.Padre = null;  ┘
        }  6
```



Caminos básicos	Complejidad iclomática
C1:1-2-3-5-6	V(G)= A-N+2
C2:1-3-4-5-6	V(G)= 7 - 6 + 2 = 3 = R3
C3:1-2-3-4-5-6	

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

Caso de prueba: Desconectar relación de madre.

Entrada: persona (Nombre = Bertha), Madre = Ana, Padre = null

Resultado Esperado: Si existe un hijo conectado con la madre, el árbol genealógico se actualiza borrando la conexión, desaparecen la conexion existente entre madre-hijo.

Condiciones: Exista hijo.

Camino 2: 1-3-4-5-6

Caso de prueba: Desconectar relación de padre.

Entrada: persona (Nombre = Bertha), Madre = null, Padre =Pedro

Resultado Esperado: Si existe un hijo conectado con el padre, el árbol genealógico se actualiza borrando la conexión, desaparecen la conexion existente entre padre-hijo.

Condiciones: Exista hijo.

Camino 3: 1-2-3-4-5-6

Caso de prueba: Desconectar relación con los padres.

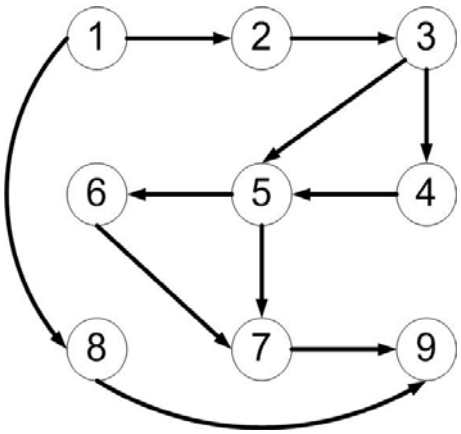
Entrada: persona (Nombre = Bertha), Madre = Ana, Padre =Pedro

Resultado Esperado: Si existe un hijo conectado con el padre y con la madre, el árbol genealógico se actualiza borrando la conexión, desaparecen la conexion existente entre padre-hijo y madre-hijo.

Condiciones: Exista hijo.

Método AdicionarHermano

```
public void AdicionarHermano(Persona nuevoHermano, Persona persona)
  {
    if ( (BuscarPersona(persona) > -1) && (BuscarPersona(nuevoHermano) > -1) )  1
      {
        if (PuedeSerHermanoDe(nuevoHermano, persona))   2
            {
              if (persona.Padre!= null)   3
                 AdicionarPadre( persona.Padre, nuevoHermano);   4
              if (persona.Madre!= null)   5
                 AdicionarMadre(persona.Madre, nuevoHermano);   6
            }  7
      }
    else throw new Exception("Elemento no perteneciente a la familia");   8
  }  9
```



Caminos básicos	**Complejidad ciclomática**
C1:1-8-9	V(G)= A-N+2
C2:1-2-3-4-5-7-9	V(G)= 11- 9 + 2 = 4 = R4
C3:1-2-3-5-6-7-9	
C4:1-2-3-4-5-6-7-9	

Camino 1: 1-8-9

Caso de prueba: No pertenece a la familia.

Entrada: nuevo hermano (Nombre = Pedro), Padre = Alberto, Madre = Ana

Resultado Esperado: Devuelve que las personas no pertenecen a la familia.

Condiciones: Existan personas.

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

Caso de prueba: Adicionar al nuevo hermano la relación con el padre.

Entrada: nuevo hermano (Nombre = Pedro), Padre = Alberto, Madre = null

Resultado Esperado: Si la persona es un nuevo hermano se establece la relación de hermano entre esos individuos y además se añade la relación con el padre a ese nuevo hermano adicionado.

Condiciones: Exista nuevo hermano.

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

Caso de prueba: Adicionar al nuevo hermano la relación con la madre.

Entrada: nuevo hermano (Nombre = Pedro), Padre = null, Madre = Ana

Resultado Esperado: Si la persona es un nuevo hermano se establece la relación de hermano entre esos individuos y además se añade la relación con la madre a ese nuevo hermano adicionado.

Condiciones: Exista nuevo hermano.

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

Caso de prueba: Adicionar hermano.

Entrada: nuevo hermano (Nombre = Pedro), Padre = Pedro, Madre = Ana

Resultado Esperado: Si la persona es un nuevo hermano se establece la relación de hermano entre esos individuos y además se añade la relación con la madre y el padre a ese nuevo hermano adicionado.

Condiciones: Exista nuevo hermano.

ANEXO II

CPR 1: Datos médicos

Descripción.

Permite conocer la cantidad de pruebas que se ha tomado el individuo, conociendo además el tipo de muestra.

Flujo Central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario dibuja el árbol en el área destinada a la confección del árbol.
- El usuario selecciona un individuo.
- Se selecciona en el menú de la derecha la opción *Datos médicos*.
- El usuario llena los datos de la muestra, para que sean guardados por el sistema.

Condiciones de ejecución.

- Existan muestras realizadas al individuo para ser introducidas en el sistema.
- Exista un individuo seleccionado.

CPR 2: Estado genético.

Descripción.

Permite identificar a cada individuo en dependencia del símbolo que le corresponda.

Flujo Central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario dibuja el árbol en el área destinada a la confección del árbol.
- El usuario selecciona un individuo.
- Se selecciona en el menú de la derecha la opción *Estado genético*.
- El usuario puede escoger el símbolo que tiene el individuo o no escoger ninguno.

Condiciones de ejecución.

- Exista un individuo seleccionado.

CPR 3: Familiares.

Descripción.

Permite obtener información de quiénes son los familiares del individuo, ya sea si tiene hijos, el nombre de la madre y el nombre del padre.

Flujo Central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario dibuja el árbol en el área destinada a la confección del árbol.
- El usuario selecciona un individuo.
- Se selecciona en el menú de la derecha la opción *Familiares.*
- El usuario puede ver el nombre y la cantidad de hijo del individuo además de ver el nombre de la madre y el padre.
- El sistema guarda esta información.

Condiciones de ejecución.

- Exista un individuo seleccionado.

CPR 4: Generales.

Descripción.

Permite obtener información general del individuo como fecha de nacimiento, color de piel, dirección, edad, nombre, apellidos, etc.

Flujo Central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario dibuja el árbol en el área destinada a la confección del árbol.
- El usuario selecciona un individuo.
- Se selecciona en el menú de la derecha la opción *Generales.*
- El usuario llena todos los campos especificados.
- El sistema guarda esta información.

Condiciones de ejecución.

- Exista un individuo seleccionado.

CPR 5: Información.

Descripción.

Permite identificar si el individuo es el propósito por el cual se decide crear el árbol.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario dibuja el árbol en el área destinada a la confección del árbol.
- El usuario selecciona un individuo.
- Se selecciona en el menú de la derecha la opción *Información.*
- El usuario tiene la posibilidad de llenar los campos especificados, no es necesario que sean todos.

- El sistema guarda esta información.

Condiciones de ejecución.

- Exista un individuo seleccionado.

CPR 6: Maternidad.

Descripción.

Si el individuo es de sexo femenino permite decir si está embarazada o no, si es de sexo masculino la opción se desactiva.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario dibuja el árbol en el área destinada a la confección del árbol.
- El usuario selecciona un individuo.
- Se selecciona en el menú de la derecha la opción *Maternidad.*
- El usuario escoge la opción que corresponda.
- El sistema guarda la información.

Condiciones de ejecución.

- Exista un individuo seleccionado.

CPR 7: Definición (Enfermedades).

Descripción.

Permite ingresar al sistema las enfermedades para su posterior identificación para cada individuo.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Definición.*
- Selecciona *Enfermedades.*
- Ingresa las enfermedades.
- Oprime el botón Aceptar.
- El sistema guarda los datos ingresados.

Flujo alterno.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Definición.*
- Selecciona *Enfermedades.*
- Ingresa las enfermedades.

- Oprime el botón *Cancelar*.
- El sistema cierra la ventana y no modifica la información.

Condiciones de ejecución.

- Exista enfermedad que agregar o introducir al sistema.

CPR 7: Definición (Símbolo).

Descripción.

Permite adicionar un símbolo nuevo al sistema.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Definición.*
- Selecciona *Símbolo.*
- El usuario tiene la opción de agregar un nuevo símbolo seleccionando el botón *Nuevo.*
- Oprime el botón *Aceptar*.
- El sistema guarda los datos ingresados.

Flujo alterno.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Definición.*
- Selecciona *Símbolo.*
- El usuario tiene la opción de agregar un nuevo símbolo seleccionando el botón *Nuevo.*
- Oprime el botón *Cancelar.*
- El sistema cierra la ventana y no modifica la información.

Condiciones de ejecución.

- Exista símbolo que agregar o introducir al sistema.

CPR 8: Definición (Examen prenatal).

Descripción.

Permite ingresar al sistema un nuevo examen prenatal o eliminarlo en caso que error.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Definición.*
- Selecciona *Examen prenatal.*
- El usuario puede introducir un examen oprimiendo el botón *Adicionar.*

- El sistema guarda la información introducida cuando de oprime el botón *Terminar*.

Flujo alterno.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Definición.*
- Selecciona *Examen prenatal.*
- El usuario puede eliminar un examen oprimiendo el botón *Eliminar.*
- El usuario oprime el botón *Terminar* y se guarda la información modificada.

Condiciones de ejecución.

- Exista un examen prenatal que agregar o introducir al sistema.

CPR 9: Archivo (Abrir)

Descripción.

Permite abrir un árbol que ya esté creado con anterioridad y que esté guardado en cualquier parte del ordenador.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Archivo.*
- Selecciona la opción *Abrir*.
- El usuario puede abrir cualquier árbol.
- El sistema muestra el árbol seleccionado.

Condiciones de ejecución.

- Exista un árbol para abrir.

CPR 10: Archivo (Guardar)

Descripción.

Permite guardar el árbol creado así como cualquier cambio que sea efectuado en el transcurso de la confección del mismo.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Archivo.*
- Selecciona la opción *Guardar*.
- El sistema guarda los cambios efectuados.

Condiciones de ejecución.

- Exista un árbol para guardar o alguno al cual se le hayan realizado cambios.

CPR 11: Archivo (Nuevo)

Descripción.

Permite abrir un nuevo editor para dibujar un nuevo árbol.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Archivo.*
- Selecciona la opción *Nuevo*.
- El usuario puede dibujar un nuevo árbol.

Condiciones de ejecución.

- Exista un nuevo árbol para crear.

CPR 12: Archivo (Exportar)

Descripción.

Permite exportar desde otra parte ya sea texto o un árbol que ya esté creado.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Archivo.*
- Selecciona la opción *Exportar*.
- El usuario selecciona la opción *Texto*.
- El sistema exporta el texto seleccionado hacia el árbol.

Flujo alterno

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Archivo.*
- Selecciona la opción *Exportar*.
- El usuario selecciona la opción *Árbol*.
- El sistema exporta el árbol seleccionado.

Condiciones de ejecución.

- Exista un árbol para ser exportado.

CPR 13: Archivo (Salir)

Descripción.

Se cierra el sistema automáticamente.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Archivo.*
- Selecciona la opción *Salir*.
- El sistema se cierra.

CPR 14: Vista (Ampliar)

Descripción.

Permite ampliar la imagen del árbol cuanto se desee, o sea al tamaño que escoja el usuario.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Vista.*
- Selecciona la opción *Ampliar.*
- El sistema aumenta la imagen del árbol.

Condiciones de ejecución.

- Exista un árbol confeccionado para ser ampliado.

CPR 15: Vista (Reducir)

Descripción.

Permite reducir la imagen del árbol cuanto se desee, o sea al tamaño que escoja el usuario.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede a la barra de herramientas y selecciona la opción *Vista.*
- Selecciona la opción *Reducir.*
- El sistema comprime la imagen del árbol.

Condiciones de ejecución.

- Exista un árbol confeccionado para ser reducido.

CPR 16: Datos de la creación.

Descripción.

Permite dejar registrado los datos de la persona que confecciona el árbol.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede al menú de la derecha de la aplicación.
- El usuario introduce sus datos personales en la sección *Datos de la creación*.
- El usuario introduce su nombre y apellidos.
- El usuario introduce la fecha de la creación del árbol.
- El usuario introduce la fecha de la última modificación realizada en el árbol.

Condiciones de ejecución.

- Exista una persona destinada a realizar los árboles.

CPR 17: Datos de la familia.

Descripción.

Permite dejar registrado los datos de la familia a la que se le va a confeccionar el árbol.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario accede al menú de la derecha de la aplicación.
- El usuario introduce los datos de la familia en la sección *Datos de la familia*.
- El usuario introduce el color de la enfermedad con que se dibujaran los símbolos.
- El usuario introduce la enfermedad que afecta a la familia.
- El usuario introduce el número que identifica a la familia.
- El usuario introduce el nombre que identifica a la familia.

Condiciones de ejecución.

- Exista una familia que vaya a ser representada en un árbol.

CPR 18: Barra de herramientas (Botón Femenino).

Descripción.

Permite acceder de forma rápida a varias funcionalidades del sistema como por ejemplo crear los individuos.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.

- El usuario selecciona el botón *Femenino* que se encuentra en la barra de herramientas.
- Lo marca y pinta el individuo en el área de dibujo.

Condiciones de ejecución.

- Exista una familia que vaya a ser representada en un árbol.

CPR 19: Barra de herramientas (Botón Masculino).

Descripción.

Permite acceder de forma rápida a varias funcionalidades del sistema como por ejemplo crear los individuos.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario selecciona el botón *Masculino* que se encuentra en la barra de herramientas.
- Lo marca y pinta el individuo en el área de dibujo.

Condiciones de ejecución.

- Exista una familia que vaya a ser representada en un árbol.

CPR 20: Barra de herramientas (Botón Desconocido).

Descripción.

Permite acceder de forma rápida a varias funcionalidades del sistema como por ejemplo crear los individuos.

Flujo central.

- El sistema muestra la interfaz principal de la aplicación.
- El usuario selecciona el botón *Desconocido* que se encuentra en la barra de herramientas.
- Lo marca y pinta el individuo en el área de dibujo.

Condiciones de ejecución.

- Exista una familia que vaya a ser representada en un árbol.

ANEXO DE FIGURAS.

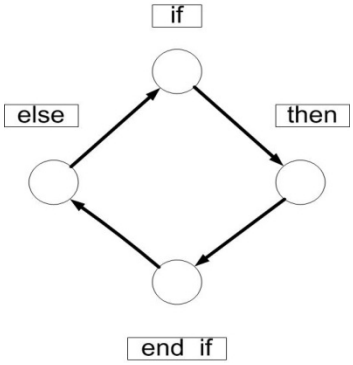


Fig.1.1 Notación del grafo de flujo para la Sentencia **If.**

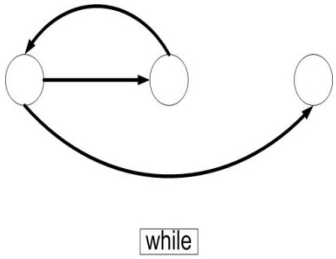


Fig.1.2 Notación del grafo de flujo para la Sentencia **While.**

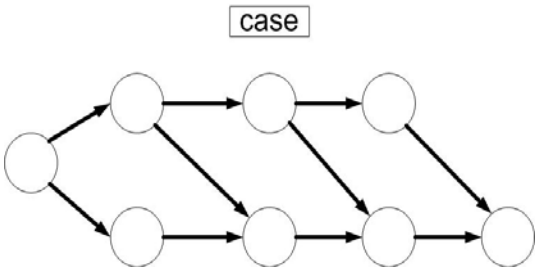


Fig.1.3 Notación del grafo de flujo para la Sentencia **Case.**

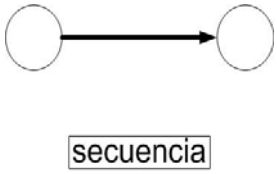


Fig.1.4 Notación del grafo de flujo para la Sentencia **Secuencia.**

GLOSARIO DE TÉRMINOS.

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. Véase trabajador, actividad.

Atributo: Característica de un archivo o carpeta.

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

Caso de uso**:** Un casos de uso es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas.

CMM: El CMM para software (CAPABILITY MATURITY MODEL - Modelo de Capacidad de Madurez) es el modelo del SEI (Software Engineering Institute) que las organizaciones pueden usar para determinar sus capacidades para desarrollar o mantener software, basado en un proceso de mejoramiento continuo en la organización.

Código: Se refiere a las instrucciones contenidas en un programa, y entendibles por el ordenador. Las aplicaciones normales pueden tener miles de líneas de código que es necesario mantener y actualizar.

Cortafuego: Máquina situada entre la red de una empresa e Internet, dedicada a mantener fuera del perímetro a los usuarios no autorizados.

Defecto: Error, falta o fallo.

Ergonomía: Estudio de datos biológicos y tecnológicos aplicados a problemas de mutua adaptación entre el hombre y la máquina.

Error: Acción humana que produce una falta.

Estándar: Es cualquier tecnología aprobada por un comité formalizado.

Fallo: Manifestación de una falta.

Falta: Algo que está mal en un producto (modelo, código, documento, 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.

Herramienta: Subprograma o módulo encargado de funciones específicas y afines entre sí para realizar una tarea. Una aplicación o programa puede contar con múltiples herramientas a su disposición. Por ejemplo, el corrector ortográfico puede ser una herramienta en una aplicación para redactar documentos, pero no es una aplicación en sí misma.

***IEEE*:** (Institute of Electrical and Electronics Engineers). Asociación de profesionales norteamericanos que aporta criterios de estandarización de dispositivos eléctricos y electrónicos.

Interfaz: Una colección de operaciones que se usan para especificar el servicio de una clase o de un componente. Circuito electrónico que gobierna la conexión entre el hombre y el hardware, y los ayuda a intercambiar información de manera confiable.

ISO: (International Organization for Standardization). ISO establece o fija estándares internacionales. Se ocupa de todos los campos, excepto la electricidad y la electrónica, que se rigen por la International Electrotechnical Commission (IEC).

Metodología: Es un conjunto de procedimientos, técnicas, instrumentos y documentos que ayudan a los analistas y programadores en sus esfuerzos para obtener un nuevo sistema informativo.

Modelo: Representación de la realidad por abstracción.

Portabilidad: Característica de ciertos programas que les permite ser utilizados en distintos ordenadores sin que precisen modificaciones de importancia.

***Procedimiento*:** Dentro de una aplicación, se denomina procedimiento al conjunto de instrucciones, controles, etc. que hacen posible la resolución de una cuestión específica. La impresión es un procedimiento, como lo es la incorporación de una imagen a un texto predeterminado, etc.

Proceso: Un proceso es una instancia de un programa. Actualmente los sistemas multitarea soportan la ejecución de múltiples procesos, dando la apariencia de que pueden correr simultáneamente (de forma concurrente). De hecho, sólo un proceso puede estar siendo ejecutado al mismo tiempo por el CPU (excepto los CPU con múltiples procesadores). Los procesos son creados, destruidos y comunicados entre sí por el sistema operativo. En Windows se pueden ver los procesos en ejecución desde el Administrador de Tareas.

Script: Pequeños programas incrustados en las páginas que nos permiten definir interactividades de cualquier tipo.

Seguridad: Seguridad es la cualidad de seguro, es decir, de estar libre y exentos de todo daño, peligro o riesgo.

Sistema Informático: Tiene por finalidad exclusiva y excluyente el almacenamiento, el procesamiento, la recuperación y la difusión de la información contenida en documentos de cualquier especie. Conjunto u ordenación de elementos organizados para llevar a cabo algún método, procedimiento o control mediante el proceso de información.

***Software*:** Software es un término genérico que designa al conjunto de programas de distinto tipo (sistema operativo y aplicaciones diversas) que hacen posible operar con el ordenador.

Técnica: Conjunto de saberes prácticos o procedimientos para obtener un resultado. Requiere de destreza manual e intelectual, y generalmente con el uso de herramientas. Las técnicas se transmiten

de generación en generación. La técnica nace en la imaginación y luego se llevan a la concreción, siempre de forma empírica. En cambio la tecnología surge de forma científica, reflexiva y con ayuda de las técnicas.

Tecnología: Abarca un conjunto de técnicas, conocimientos y procesos para el diseño y construcción de objetos para satisfacer necesidades humanas. La tecnología puede referirse a objetos que usa la humanidad (como máquinas, utensilios, hardware), pero también abarca sistemas, métodos de organización y técnicas, se basa en aportes científicos.

Usabilidad: La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y de ser atractivo para el usuario, en condiciones específicas de uso. La usabilidad también hacer referencia al grado de facilidad con que una aplicación, sitio Web, periférico o sistema, se adapta a sus usuarios. La usabilidad puede estar relacionada incluso a la Ergonomía, y a la potabilidad de un dispositivo.

Usuario: Persona que utiliza el software.