Universidad de las Ciencias Informáticas

Facultad 2

´´Propuesta de estrategia de pruebas de software para el Centro de Informatización de la Seguridad Ciudadana (ISEC)´´.

Trabajo de Diploma para optar por el Título de

INGENIERO EN CIENCIAS INFORMATICAS

Autores:

Leidy Laura Sánchez González.

Amaray Aranyos Bravo.

Tutor:

Ing.Norbelis Leyva Montero.

Co-Tutor:

MSc.Violena Hernández Aguilar.

¨Año del 52 Aniversario de la Revolución¨

Ciudad de la Habana, Cuba, 2010.

La calidad nunca es un accidente; siempre es el resultado de un esfuerzo de la inteligencia.

John Ruskin.

Declaración de Autoría

Por este medio declaramos que somos los únicos autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmamos la presente a los_ días del mes de _del año _.

_____ _____

Leidy Laura Sánchez González. Amaray Aranyos Bravo.

_____

Ing.Norbelis Leyva Montero

Dedicatoria…

Este trabajo va dedicado especialmente para las personitas más maravillosas del mundo y las más importantes en mi vida, mis padres: Marta y Lauro por ser tan especiales conmigo, por luchar a mi lado en cada momento de mi vida, por estar siempre ahí cuando más los necesité, por darme tanto amor y cariño, a ellos les debo todo, por lograr hacer de mí una persona de bien y una profesional, por eso ni dándoles mi vida entera los recompensaré jamás

A mi hermanito querido, a mis adorados abuelos a toda mi maravillosa familia y a mis grandes amigos que tanto me han apoyado y han influido en la realización de este sueño hecho realidad.

A todos ustedes va dedicado este trabajo. Gracias.

Leidy Laura Sánchez González

Este trabajo se lo dedico a tres personas que han sido TODO en mi vida, que me han dado tanto cariño y confianza que no sabría cómo recompensarlos. A mi mamá Sara y mis abuelos Amparo y Pedro les dedico el esfuerzo de 5 años y todos mis logros. Mi abuelo ha sido mi padre, mi amigo y quisiera que Dios le diera muchos años de vida para poderle agradecer. Mi abuela me ha dado el carácter y la fuerza para desafiar las adversidades. Mi mamá me ha mimado y consentido, pero a la vez ha posibilitado que todos estos años fuera de mi hogar fueran una experiencia única y que mis deseos de superarme profesionalmente eclipsaran los miedos e inseguridades. A mi abuelito del alma que me querido incondicionalmente. A todos ellos les dedico MI VIDA.

Amaray Aranyos Bravo

Agradecimientos

Cuando son tantas las personas involucradas en tu sueño y la gratitud hacia ellos es tan grande se te quedan cortas las palabras para reconocer a todos los que han tenido que ver con que suceda este momento tan especial en mi vida.

A mí mamita adorada por ser tan linda conmigo, por apoyarme en todo, a mi papi por ser un ejemplo, una guía, por querer que cada día que pasa aprenda más, a él le debo el estar hoy aquí. Gracias a ambos por confiar en mí.

A mi hermosa familia por preocuparse tanto, a mis abuelos esos que en todo me consienten y me ayudan, a mi hermanito adorado por ser tan bueno y especial.

A todos mis amigos esos que estuvieron desde el inicio y sé que sin ellos hubiera sido un poco más difícil, a la Yari por estar siempre ahí, a Reyni a Willo por ser tan buenos amigos, a Angel porque todo el tiempo que estuvo a mi lado me ayudo muchísimo en los años más difíciles de esta escuela y sin el creo que nada hubiera sido igual. A los amigos que llegaron después pero no por eso dejan de ser importantes a Leli eres una de las personas más maravillosas que he conocido, a Misly a Yunnier, a Danny a Aylín, a todos los quiero muchísimo.

A mí compañera de tesis, Amaray no sabes que feliz me siento de haber compartido este trabajo contigo eres la mejor compañera que alguien pueda tener, me siento muy orgullosa de haberte escogido y haber visto en ti no solo el talento que tienes sino también la gran amiga que puedes ser.

A los profesores que más que eso fueron esas personas mayores que siempre estuvieron ahí para aconsejarme y que los considero los amigos adultos que todos tenemos, la Yure y Abel muchas gracias.

A todos mis compañero de grupo que fueron participes de muchas alegrías y tristezas todas son grandes personas a las que admiro y siempre recordaré.

A mi tutora Norbelis que aunque muchas veces no estaba de acuerdo con las cosas que me decías siempre pude ver en ti alguien que se toma las cosas muy en serio y que de verdad le importaba, gracias por la paciencia.

A Dios y a San Lázaro por iluminarme el camino y que todo resultara más fácil.

En fin doy las gracias a todas las personas que estuvieron a mi lado a lo largo de este sueño, hoy hecho realidad.

Leidy Laura

Agradecimientos

A mi mamá y mis abuelos les agradezco el apoyo y la confianza que depositaron en mí.
A mis tías Aimé y Magalis por hacerme sentir dentro de la familia como una persona digna de imitar.
A mi tío Papo por hacerme sentir alguna vez como su sobrina preferida. A Benitico, Delia y Gilberto.
Mi mayor agradecimiento a Yunnier por estar conmigo en todo momento desde que entré en la Universidad, por ser mi amigo, mi confidente y el amor que todos deseamos encontrar.
A Caridad, Yudermis, Martha, Pablo, Yuni, Maricela, Oria, a todos ellos que me hicieron sentir como en casa los años que estuve fuera.
No podría dejar de agradecer a mi compañera de tesis Leidy Laura que estuvo a mi lado en los momentos difíciles durante el desarrollo de este trabajo y supo verle el lado bueno a todo.
A mi amiga de muchos años Anileny con la que he compartido momentos y otros muy divertidos.
A todos los amigos que fueron motivos de muchas alegrías: Leydi Katlen, Anierys, Sam, Erick y todos los de mi actual grupo.
A esos amigos que no pudieron continuar estudios con nosotros: Yasnahi, Yulaine, Pablo, Danay, Aylen.
A mi tutora Norbelis por tener tanta paciencia y preocuparse tanto por mí.
A todos lo que significan mucho para mis más sinceros agradecimientos.

Amaray

Resumen

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia, esta es medible y varía de un sistema a otro.

Una disciplina importante en el proceso de desarrollo de los Sistemas de Software es la relativa a Pruebas, orientada a encontrar fallas en el sistema y con su erradicación contribuir a garantizar la calidad del software.

La presente investigación se enmarca en la propuesta de una Estrategia de Prueba para el Centro de Informatización de la Seguridad Ciudadana ISEC de la Facultad 2, compuesta por niveles, tipos, técnicas y herramientas de pruebas que tributan a garantizar las características significativas del software en desarrollo.

Para obtener un resultado adecuado, se realizó un análisis de la metodología de desarrollo utilizada en los proyectos que pertenecen al centro, así como la NC ISO/IEC 9126-1: 2005 para poseer conocimiento profundo de las características de calidad que debe tener todo software. La validación de la Estrategia de Pruebas se realizó a través del Método Experto, por este medio se comprobó que la estrategia propuesta tuvo un índice de aceptación elevado por parte de los expertos seleccionados.

Palabras Claves: Características de calidad de software, Proceso de Pruebas, Estrategia de pruebas, Validación.

Índice

Introducción

La informática es una ciencia que le ha permitido al hombre realizar innumerables actividades y ha facilitado en gran medida la gestión y organización de su trabajo, convirtiéndose así en parte del sustrato tecnológico del proceso de globalización en el cual está inmerso el mundo. Por tanto se impone la necesidad de brindarle a las nuevas generaciones una cultura tecnológica resultado de la actual dependencia de los sistemas computacionales, donde alguna falla puede ocasionar catástrofes económicas (Errores en sistemas transaccionales de los bancos) o humanas (Fallas en los sistemas de control aéreo), por citar algunas. La informática está en la vida de todos, gran parte de la sociedad se ha desarrollado al amparo de las nuevas tecnologías y debe su éxito en gran parte a esta ciencia.

El desarrollo del mercado mundial, el incremento de la competencia, de las producciones, sumado a la presión de la sociedad y de las administraciones en consumir productos de calidad trae consigo el empeño de las empresas en proporcionar mayor satisfacción a los consumidores. Esta evolución ayuda a comprender de dónde proviene la necesidad de ofrecer una mayor calidad del producto o servicio que se proporciona al cliente. Las organizaciones se encuentran en una situación donde deben idear estrategias que las pongan en ventaja con sus competidores. Todo esto ha permitido el desarrollo de productos con alta calidad, productos que sus características inherentes cumplen con requisitos establecidos y que su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema; de esta manera, se pretende minimizar los problemas relacionados al desarrollo de sistemas. La calidad no se ha convertido únicamente en uno de los requisitos esenciales del producto sino que en la actualidad es un factor estratégico clave del que dependen la mayor parte de las organizaciones, no sólo para mantener su posición en el mercado sino incluso para asegurar su supervivencia.

El software durante su desarrollo transita por un proceso que está compuesto por una serie de etapas, que se necesitan ejecutar para generar un producto. En la etapa de prueba es donde se verifica que el software realice correctamente las tareas indicadas en la especificación del problema, garantizando la calidad exigida por el cliente.

Cuba en su afán por insertase en el mercado de la informática ha creado la Universidad de las Ciencias Informáticas (UCI), un centro de estudios universitarios, con el propósito de contribuir a la informatización

de los procesos de la sociedad cubana y desarrollar la industria del software. Alcanzar la máxima calidad de sus productos es uno de los principales objetivos para introducir al país en dicho mercado, siguiendo un conjunto organizado de procedimientos definidos y entrelazados armónicamente, que permitan alcanzar las metas trazadas.

En la UCI, el Laboratorio Industrial de Pruebas de Software (LIPS) perteneciente a la dirección CALISOFT es el encargado de certificar la conformidad de los productos desarrollados en la Universidad. A raíz de las pruebas efectuadas por el LIPS y las Revisiones y Auditorías realizadas en el 2009 por el grupo de CALISOFT encargado de estas funciones, se detectaron un conjunto de No Conformidades como resultado del ineficiente proceso de prueba aplicado por los proyectos productivos en la UCI; estas se concentran siguiendo un orden decreciente en: gestión de los requisitos, establecimiento de la gestión de la configuración, definición del proyecto, definición de la arquitectura de software, gestión de pruebas y completamiento del expediente de proyecto.

Los proyectos que se desarrollan en el Centro de Informatización de la Seguridad Ciudadana (ISEC) de la Facultad 2, no están exentos de dicho escenario ya que estas No Conformidades están generalizadas en mayor o menor grado en todos los proyectos que se ejecutan en la Universidad; lo que evidencia una inadecuada aplicación del proceso de prueba al software durante su proceso de desarrollo, al no contar con una estrategia de pruebas que les permita guiar el proceso de pruebas, generando así deficiencias que inciden de forma negativa al no cumplir con los requisitos del producto estipulados con el cliente.

Teniendo en cuenta lo antes expuesto, el **Problema Científico** queda formulado de la siguiente manera: ¿Cómo lograr un proceso de prueba de software que se adecúe a las características distintivas de los productos desarrollados en los proyectos productivos del Centro de Informatización de la Seguridad Ciudadana?

En la presente investigación se tiene como **Objeto de Estudio**: El proceso de pruebas de software.

Centrando el **Campo de Acción** en**:** El proceso de pruebas de software para los productos del ISEC.

Para darle solución al problema antes planteado se ha trazado como **Objetivo General**: Proponer una estrategia de pruebas adecuada a las características significativas del software desarrollado en el Centro de Informatización de la Seguridad Ciudadana de la Facultad 2.

Como **Idea a Defender** que**:** Con la propuesta de la estrategia de pruebas de software que permita aportar la confianza adecuada en que el producto satisfacerá los requisitos dados de calidad, se garantizará la identificación y evaluación de los defectos que puedan afectar a los proyectos del ISEC de forma temprana, propiciando con esto la entrega del software en tiempo y con la calidad requerida.

Para el cumplimiento de los objetivos trazados se proponen las siguientes **Tareas Científicas:**

- ✓ Análisis de la norma NC ISO/IEC 9126-1: 2005 para conocer las características de calidad que debe cumplir el software.
- ✓ Entrevista con los expertos de los proyectos que conforman el ISEC para definir las características de calidad con más peso en dichos proyectos.
- ✓ Investigación del proceso de pruebas de software, tipos, técnicas, y herramientas para un mayor conocimiento sobre el tema.
- ✓ Análisis de herramientas de software libre para pruebas con el objetivo de automatizar las pruebas propuestas como parte de la estrategia.

Para lograr un mejor desarrollo de esta investigación se emplearon los siguientes **Métodos Científicos**:

Métodos teóricos:

- ✓ Método analítico – sintético: Se utilizó para que toda la información obtenida durante la investigación, quedara organizada y sintetizada para finalmente darle una estructura adecuada, permitiendo obtener los elementos más significativos que se relacionan con el tema a desarrollar.
- ✓ Inductivo-Deductivo: Se utilizó para el planteamiento del objetivo, la idea a defender y la extracción de las ideas fundamentales.

Métodos Empíricos:

- ✓ Entrevista: Se emplea para obtener las características de los productos del ISEC y tener conocimiento de su proceso de desarrollo.
- ✓ Observación: Se utiliza para conocer y analizar el proceso de pruebas de software en la Facultad 2 y para revisar las evidencias arrojadas en las entrevistas realizadas.
- ✓ Encuesta: Se utiliza para conocer el impacto de la propuesta brindada para mejorar el proceso de pruebas de software.

El documento está organizado en tres capítulos, a continuación se brinda una breve descripción:

Capítulo 1: **"Fundamentación teórica".** Se ilustran conceptos relacionados con la calidad del software y la gestión de la misma así como el proceso de pruebas de software, se analizan las tendencias en el área a nivel mundial y en el país para profundizar en cuanto a los diferentes tipos de pruebas existentes, sus características y procesos que las componen.

Capítulo 2: **"Propuesta de la estrategia".** Se presenta la propuesta de estrategia de pruebas para los productos del ISEC haciendo un análisis minucioso de las técnicas, herramientas y tipos de pruebas, que permiten preservar las características de calidad identificadas como las de mayor importancia para el software de dicho centro.

Capítulo 3: **"Validación de la propuesta".** Para validar la estrategia propuesta se utilizará el Método de Experto, ya que no se tienen los antecedentes necesarios para poder realizar una comparación entre los resultados obtenidos anteriormente y los actuales aplicando la estrategia. Se mostrarán los resultados de las opiniones de expertos con experiencia en el área de calidad específicamente en pruebas, donde se evidenciará la calidad e impacto del trabajo de diploma.

Capítulo 1 Fundamentación Teórica

1 Introducción

En este capítulo se abordan conceptos relacionados con la calidad del software y la gestión de la misma, con el objetivo de procurar una perspectiva profunda sobre el contenido a tratar en el presente Trabajo de Diploma. Se realizará un estudio de la NC ISO/IEC 9126-1: 2005, y un análisis de la metodología de desarrollo aplicada en el ISEC, haciendo énfasis en el flujo de trabajo de pruebas, así como los tipos de pruebas que se le efectúan a los software durante su ciclo de vida.

1.1. Calidad

"*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"[1]*

La Organización Internacional para la Estandarización (ISO) define calidad como "*Conjunto de propiedades y de características de un producto o servicio, que le confieren aptitud para satisfacer necesidades explícitas o implícitas*".[2]

Desarrollar un producto llevando a cabo un serio proceso de calidad ayuda a mejorar la imagen de la empresa, brindando apoyo al marketing de la misma, generando valor añadido y favoreciendo el espíritu de equipo, todo esto hace que exista un crecimiento sostenido basado en la excelencia que permite hacer inversiones sin riesgos.

La UCI se rige por una norma que define un conjunto de características que todo software debe tener, para obtener la calidad requerida, la NC ISO/IEC 9126-1: 2005 en ella se describe un modelo en dos partes, para la calidad de los productos de software calidad interna y externa y calidad de uso. Esta norma cuenta con seis características mencionadas y explicadas a continuación, cada una de ellas con un conjunto propio de sub características que las complementan.

- ✓ **Funcionalidad:** La capacidad del software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas cuando el software se usa bajo las condiciones especificadas.

- **Idoneidad:** Capacidad del software para mantener un conjunto apropiado de funciones para las tareas y los objetivos del usuario especificados.
- **Precisión:** Capacidad del software para proporcionar efectos o resultados correctos o convenidos con el grado de exactitud necesario.
- **Interoperabilidad:** Capacidad del producto de software para interactuar recíprocamente con uno o más sistemas especificados.
- **Seguridad:** Capacidad del producto de software para proteger información y los datos, para que personas o sistemas desautorizados no puedan leer o pueden modificar los mismos, y las personas o sistemas autorizados tenga el acceso a ellos.
- **Conformidad con la funcionalidad:** Capacidad del software para adherirse a las normas que se le apliquen, convenciones, regulaciones, leyes y las prescripciones similares relativas a la funcionalidad.

✓ **Confiabilidad:** La capacidad del software para mantener su nivel de ejecución cuando se usa bajo las condiciones especificadas.

- **Madurez:** Capacidad del producto de software de evitar un fallo total como resultado de haberse producido un fallo del software.
- **Tolerancia ante fallos:** Capacidad del producto de software de mantener un nivel de ejecución o desempeño especificado en caso de fallos del software o de infracción de su interfaz especificada.
- **Recuperabilidad:** Capacidad del producto de software de restablecer un nivel de ejecución especificado y recuperar los datos directamente afectados en caso de fallo total.
- **Conformidad con la confiabilidad:** Capacidad del producto de software para adherirse a las normas que se le apliquen, convenciones, regulaciones, leyes y las prescripciones similares relativas a la confiabilidad.

✓ **Usabilidad:** La capacidad del software de ser comprendido, aprendido, utilizado y de ser amigable para el usuario, cuando se emplee bajo las condiciones especificadas.

- **Comprensibilidad:** Capacidad del producto de software para permitirle al usuario entender si el software es idóneo, y cómo puede usarse para las tareas y condiciones de uso particulares.

- **Cognoscibilidad:** Capacidad del producto del software para permitirle al usuario aprender su aplicación.
- **Operabilidad:** Capacidad del producto del software para permitirle al usuario operarlo y controlarlo.
- **Atracción:** Capacidad del producto del software de ser atractivo o amigable para el usuario.
- **Conformidad con la usabilidad:** Capacidad del producto de software para adherirse a las normas, convenciones, guías de estilo o regulaciones relativas a la usabilidad.

✓ **Eficiencia:** La capacidad del software para proporcionar la requerida ejecución, en relación con la cantidad de recursos usados, bajo las condiciones declaradas.

- **Rendimiento:** Capacidad del producto de software para proporcionar apropiados tiempos de respuesta y procesamiento, así como tasas de producción de resultados, al realizar su función bajo condiciones establecidas.
- **Utilización de recursos:** Capacidad del producto de software para utilizar la cantidad y el tipo apropiado de recursos cuando el software realiza su función bajo las condiciones establecidas.
- **Conformidad de la eficiencia:** Capacidad del producto de software de adherirse a las normas o convenciones que se relacionan con la eficiencia.

✓ **Mantenibilidad:** La capacidad del software de ser modificado. Las modificaciones pueden incluir las correcciones, mejoras o adaptación del software a los cambios en el ambiente, y en los requerimientos y las especificaciones funcionales.

- **Diagnosticabilidad:** Capacidad del producto del software de ser objeto de un diagnóstico para detectar deficiencias o causas de los fallos totales en el software, o para identificar las partes que van a ser modificadas.
- **Flexibilidad:** Capacidad del producto del software para permitir la aplicación de una modificación especificada.
- **Estabilidad:** Capacidad del producto de software para minimizar los efectos inesperados de las modificaciones realizadas al software.

- **Contrastabilidad:** Capacidad del producto del software para permitir la validación de un software modificado.
- **Conformidad de la mantenibilidad:** Capacidad del producto de software para adherirse a las normas o convenciones que se relacionan con la mantenibilidad.

✓ **Portabilidad:** La capacidad del software de ser transferido de un ambiente a otro.

- **Adaptabilidad:** Capacidad del producto de software de ser adaptado a los ambientes especificados sin aplicar acciones o medios de otra manera que aquellos suministrados con el propósito de que el software cumpla sus fines.
- **Instalabilidad:** Capacidad del producto de software de ser instalado en un ambiente especificado.
- **Coexistencia:** Capacidad del producto de software de coexistir con otro software independiente en un ambiente común y compartir los recursos comunes.
- **Remplazabilidad:** Capacidad del producto de software de ser usado en lugar de otro producto de software especificado para los mismos fines y en el mismo ambiente.
- **Conformidad con la portabilidad:** Capacidad del producto de software de adherirse a las normas o convenciones relativas a la portabilidad. [3]

Hoy en día la calidad es un término que preocupa a las empresas productoras de software y que debe tenerse en cuenta en todas las etapas del desarrollo del mismo. Independientemente del tipo de producto que se esté desarrollando la calidad es fundamental para lograr la satisfacción de las necesidades y expectativas del cliente, por eso es de suma importancia contar con una estrategia para garantizar que todos los productos tengan las características de calidad que sean necesarias según el tipo de software y así encaminar a mejorar la gestión de la calidad que debe garantizar el cumplimiento de las características del tipo de software en desarrollo.

Los proyectos que conforman el ISEC tienen características que los distinguen de los demás centros, el objetivo fundamental de cada uno de ellos es informatizar la seguridad ciudadana, por lo que el software se enfoca a gestionar datos referentes a dicha seguridad. Además de ser en su mayoría aplicaciones web y de utilizar igualmente como lenguaje de programación Java, gestor de base de datos Oracle y de ser un

software hecho a la medida, donde el cliente demanda un servicio, y el equipo de desarrollo se encarga de darle cumplimiento a las especificaciones requeridas por dicho cliente.

A los expertos de calidad de los proyectos productivos del ISEC se le realizó una entrevista (Consultar Anexo 1) obteniéndose como resultado que las características que plantea la NC ISO/IEC 9126-1: 2005 son de suma importancia pero específicamente para este tipo de software las que más peso tienen y las que le dan cumplimiento a las características propias del centro son las de Funcionalidad, Eficiencia, Confiabilidad y Usabilidad, según se evidencia en la siguiente gráfica.

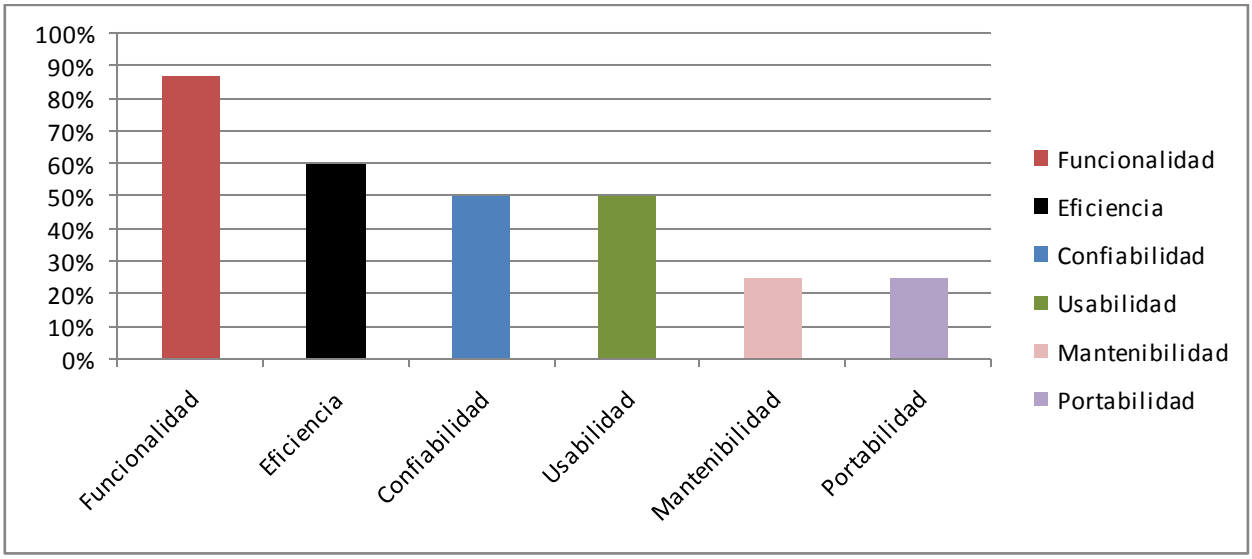


Fig. 1.1 Características presentes en el software del ISEC.

En el eje horizontal se encuentran las características que define la NC ISO/IEC 9126-1: 2005 y en el vertical la importancia en porciento que le dan los proyectos del ISEC a dichas características. A continuación se ofrece una breve descripción de estas:

Funcionalidad: Es un elemento común en todos los software del ISEC, debe ser considerado además como el aspecto más importante a garantizar, desde el inicio (procesos, requisitos, modelación) hasta la implantación y mantenimiento futuro del software. Este tipo de software comúnmente manejan información muy sensible para la seguridad de un país: antecedentes penales, registro delictivo, vida carcelaria, planes de seguridad ciudadana, operativos policiales, entre otros. De aquí entonces la importancia de garantizar, entre muchos otros aspectos: seguridad y acceso a la base de datos desde el propio gestor de bases de datos, seguridad y acceso a los servidores, servicios, redes que soportan la gestión de la

información y seguridad desde el propio diseño/programación del software de manera que se puedan evitar accesos no autorizados a la información, ya sea mediante ataques externos o debido a fallas en el propio sistema. El objetivo final debe ser garantizar la seguridad de la información almacenada por el sistema, y en este sentido se deben tener en cuenta todas las aristas del vasto mundo de la informática. Dentro de la funcionalidad las subcaracterísticas con mayor importancia son la seguridad como eje principal en el producto, la precisión ya que este tipo de software presta servicios en tiempo real.

Eficiencia: Es un aspecto muy importante, basándose en la necesidad de este tipo de sistemas que al brindar servicio en tiempo real se vuelve imprescindible trabajar aprovechando al máximo y de manera muy rápida los recursos disponibles para dar servicio a la mayor cantidad de usuarios posibles. Como subcaracterísticas de la eficiencia se puede destacar el rendimiento como el eslabón principal para un gran número de software de seguridad ciudadana debido a la necesidad de una rápida respuesta al cliente.

Confiabilidad: Para este tipo de software es de vital importancia, ya que los datos que maneja son muy delicados y no todo tipo de personas puede tener acceso a ellas. Dentro de la confiabilidad resaltar subcaracterísticas como la tolerancia ante fallos y recuperabilidad ya que es importante para ante cualquier falla del sistema este pueda recuperarse en el menor tiempo posible a fin de restablecer el servicio que presta.

Usabilidad: Es una característica propia de cada producto que hace que se considere como un aspecto fundamental para el cliente al que representa, ya que son proyectos con una amplia gama de procesos, por lo que todas las funcionalidades que están presentes en el mismo deben ser fáciles de operar para que el trabajo de los usuarios que lo manejan no sea engorroso y ayude a mejorar las labores diarias y así lograr el objetivo de su informatización. Dentro de sus subcaracterísticas podemos encontrar la operabilidad y la comprensibilidad para el usuario que interactúe con el sistema.

1.2. Gestión de la Calidad

Cuando se va a llevar a cabo un efectivo sistema de gestión se establece la política y los objetivos para la consecución de estos, se hace uso de un conjunto completo de procesos que conlleva a lograr la gestión de calidad que no es más que un grupo de actividades de la función general de la dirección que determinan la política de la calidad, los objetivos, las responsabilidades, y se implantan por medios tales

como la planificación de la calidad, mejora de la calidad, el control de la calidad, y el aseguramiento de la calidad dentro del marco del sistema de calidad.[4]

1.2.1. Planificación de la Calidad

El liderazgo en calidad requiere que los bienes, servicios y procesos internos satisfagan a los clientes. La planificación de la calidad es el proceso que asegura que dichos elementos cumplen con las expectativas de los clientes. Proporciona un enfoque participativo y estructurado para planificar nuevos productos, servicios y procesos, involucrando a todos los grupos con un papel significativo en el desarrollo y la entrega, de forma que todos participan conjuntamente como un equipo y no como una secuencia de expertos individuales. [4]

1.2.2. Mejora de la Calidad

La mejora de la calidad es parte de la gestión orientada a mejorar su eficacia y eficiencia, es un proceso para cumplir con la política de la calidad y con los objetivos de esta. Muchas de las organizaciones no suelen adquirir un hábito de constancia en la mejoría de sus productos y servicios, esto atrae muchas deficiencias en cada uno de sus procesos, lo ideal es que se planteen una buena práctica de mejora para que de esta manera tengan competitividad con las demás empresas y sobre todo puedan permanecer en el mercado. La importancia que logra tener esta técnica es que a través de su aplicación se contribuye a mejorar las debilidades y hacer que la organización se fortalezca. Con la mejora continua en las organizaciones se logra a que se desarrollen sus procesos de una manera más productiva y eficiente para así reducir costos y poder ofrecer un producto o servicio de calidad. [5]

1.2.3. Control de la Calidad

Conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requerimientos relativos a la calidad del producto o servicio, centradas en mantener bajo control el proceso de desarrollo y eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

El control de la calidad del software está centrado en dos objetivos fundamentales:

–Mantener bajo control un proceso.

–Eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

En general, se puede decir que el control de la calidad del software son las actividades para evaluar la calidad de los productos desarrollados.[5]

1.2.4. Aseguramiento de la Calidad

El Aseguramiento de la Calidad nace como una evolución natural del Control de Calidad, que resultaba limitado y poco eficaz para prevenir la aparición de defectos. Es un conjunto de acciones planificadas y sistemáticas implantadas dentro del sistema de la calidad, para proporcionar la confianza adecuada de que una entidad cumplirá los requisitos de calidad. Las actuaciones en materia de calidad deben de ir enfocados a garantizar que el producto o servicio cumplan los requisitos del cliente, además de que sea diseñado, realizado y entregado en los plazos establecidos, reciba el mantenimiento o servicio postventa adecuado, y todo ello al menor precio posible. Ello implica que la estrategia de la calidad debe ir dirigida, más a la prevención de los problemas que a la detección y solución una vez producidos.

El aseguramiento de calidad del software se diseña para cada aplicación antes de comenzar a desarrollarla. Hay quienes prefieren decir garantía de calidad en vez de aseguramiento. La garantía, puede confundir con garantía de productos, mientras que el aseguramiento pretende dar confianza en que el producto tiene calidad. [5]

1.3. Metodologías de desarrollo

Los estándares o metodologías definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software. Si no se sigue una metodología que guíe el camino de forma correcta siempre habrá incertidumbre de si lo que se desarrolla es lo que se quiere lograr. Todas las metodologías y herramientas tienen un único fin producir software de gran calidad.
Hoy en día existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Estas han demostrado ser efectivas y necesarias en un gran número de proyectos, sobre todo aquellos de gran tamaño (respecto a tiempo y recursos).

Los proyectos que conforman el ISEC utilizan como guía de desarrollo para sus productos el Rational Unified Process (RUP), la investigación estará fuertemente relacionada con esta metodología.

1.3.1. Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo (*del inglés Rational Unified Process* [RUP]) ayuda a planificar, diseñar, implementar, ejecutar y evaluar pruebas que verifiquen estas cualidades. El aseguramiento de la calidad es parte del proceso de desarrollo y no la responsabilidad de un grupo independiente, este flujo de trabajo es el encargado de evaluar la calidad del producto que estamos desarrollando, pero no para aceptar o rechazar el producto al final del proceso de desarrollo, sino que debe ir integrado en todo el ciclo de vida.[6]

El ciclo de vida del software describe el desarrollo de este desde la fase inicial hasta la fase final, todas son importantes, se requieren para guiar y validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo y asegurar de que los métodos utilizados son los apropiados que conllevaran al éxito deseado.

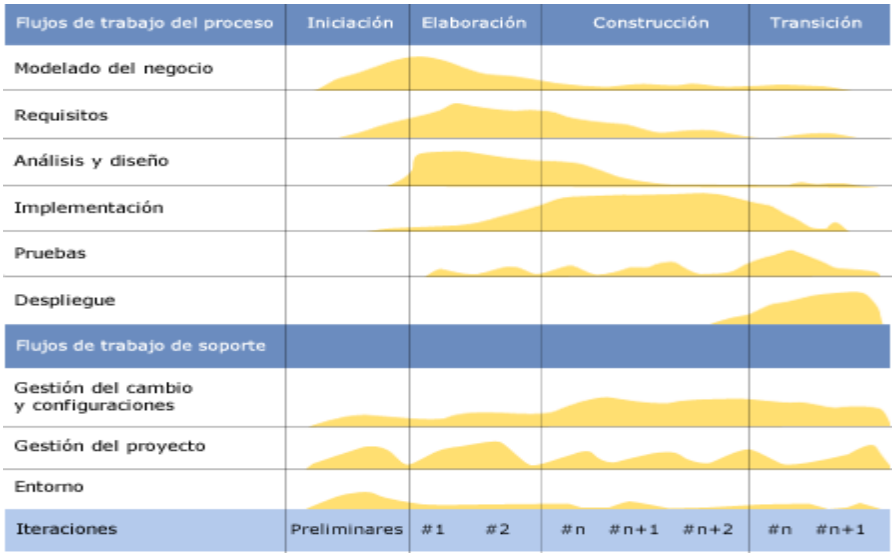


Fig. 1.1 Vista general de RUP

El flujo de trabajo de prueba está presente con un mayor o menor grado de profundidad en todas las fases del proceso de desarrollo de software para garantizar desde el comienzo la calidad del producto.

1.3.1.1. Fases

Fase de Inicio

El grupo de prueba se pone al corriente de las características del sistema propuesto, considera que pruebas requerirá y va desarrollando algunos planes provisionales de prueba. En esta etapa no se realiza un trabajo significativo de pruebas ya que es una fase exploratoria. Se puede generar un modelo de pruebas aunque no es necesario.

Fase de Elaboración

Probar los componentes ejecutables que se han implementado y que deben corresponderse con la arquitectura básica de la aplicación.

Fase de Construcción

Desarrollar los casos de prueba y procedimientos de prueba para hacerlos.

Fase de Transición

En esta fase se buscan pequeñas deficiencias que pasaron desapercibidas durante la fase de construcción y que pueden ser corregidas en el marco de la línea base de la arquitectura existente. El producto en su entorno real de operación es probado por usuarios reales.

Para cumplir estos objetivos RUP propone una serie de trabajadores y artefactos, entre los trabajadores que toman parte en este flujo de trabajo se encuentran: Administrador de Prueba, Analista de Prueba, Diseñador de prueba y Probador. Estos trabajadores interactúan con una serie de artefactos.[7]

1.3.1.2. Roles y responsabilidades

Administrador de Prueba

Es el responsable del éxito de la prueba, este rol involucra, planificación, administración de recursos y resolución de problemas que impiden las pruebas.

Analista de Pruebas

Es el responsable de identificar y definir las pruebas requeridas, monitorear el progreso de la prueba y el resultado en cada ciclo de prueba y evaluando la calidad total experimentada como un resultado de las actividades de prueba. Este rol lleva la responsabilidad para representar apropiadamente las necesidades de los stakeholders que no tienen representación regular y directa en el proyecto.

Diseñador de prueba

Es el responsable de definir el método de prueba y asegurar su implementación exitosa. El rol incluye identificar técnicas apropiadas, herramientas e instrucciones para implementar las pruebas necesarias y encauzar los recursos correspondientes para las pruebas.

Probador

Es el responsable durante las actividades principales de las pruebas, el cual incluye la conducción de las pruebas necesarias y el registro del resultado de la prueba. [8]

1.3.1.3. Artefactos

Modelo de Pruebas

El artefacto modelo de prueba describe principalmente cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. Puede describir también cómo se han probado aspectos específicos del sistema, por ejemplo, si la interfaz de usuario del sistema cumple con su objetivo y describe el cumplimiento de los requisitos funcionales y no funcionales del sistema. Es una colección de casos de pruebas, procedimientos de prueba y componentes de prueba.[8]

Caso de prueba

El diseño de casos de prueba es uno de los pasos más importantes al realizar una estrategia de pruebas, ya que estos constituyen la fuente para evaluar los resultados del software. Especifican una forma de probar el sistema, incluyendo la entrada o resultado con que se va a probar y las condiciones bajo las que ha de probarse. Una de las principales características que deben presentar los casos de prueba es su carácter abarcador, es decir, deben ser completos y estar adaptados al producto, pues deben ofrecernos la posibilidad de encontrar la mayor cantidad de errores posibles en un tiempo y costo no muy elevados.[8]

Procedimiento de Prueba

Un procedimiento de prueba especifica cómo realizar uno o varios casos de pruebas o partes de éstos. Un procedimiento de prueba puede ser una instrucción para un individuo sobre cómo realizar un caso de

prueba manualmente, o una especificación de cómo interactuar manualmente con una herramienta de automatización de pruebas, para crear componentes ejecutables de prueba.[8]

Componente de Prueba

Un componente de prueba automatiza uno o varios procedimientos de prueba o partes de ellos. Estos pueden ser desarrollados utilizando un lenguaje de guiones o un lenguaje de programación, o con una herramienta de automatización de pruebas. Los componentes de pruebas se utilizan también para probar los componentes en el modelo de implementación, proporcionando entradas de pruebas, controlando y motorizando la ejecución de los componentes a probar e informando de los resultados de las pruebas. Los componentes de pruebas pueden ser implementados usando tecnología de objetos.[8]

Plan de prueba

La construcción de un buen Plan de Pruebas es el principal factor de éxito para la puesta en práctica de una estrategia de pruebas que permita entregar un software de mejor nivel. Es el artefacto que permite trazar el tipo de prueba que se le va a aplicar al producto, cuyo propósito es dejar de forma explícita el alcance, el enfoque, los recursos requeridos, el calendario y los responsables del proceso de pruebas. Durante el desarrollo del software se diseña el plan de prueba con el objetivo de asegurar que todos los requisitos tanto funcionales como de rendimiento, se satisfagan.[8]

Defecto

Un defecto es un síntoma de un fallo en el software o un de un problema descubierto en una revisión, el cual puede ser utilizado para localizar cualquier cosa que los desarrolladores necesiten registrar como síntoma de problema en el sistema.[8]

Evaluación de Prueba

Es una evaluación de los resultados de los esfuerzos de prueba, tales como la cobertura del caso de prueba, de código y el estado de los defectos. La evaluación es realizada por los diseñadores quienes comparan los resultados obtenidos con los objetivos trazados en el plan de prueba. Durante la evaluación

de las pruebas, se realizan métricas que permiten determinar el nivel de calidad del software y qué cantidad de pruebas se deben realizar.[8]

1.4. Evaluaciones

A medida que se generan artefactos se revisan y corrigen, la documentación generada debe ser revisada para eliminar errores relacionados tanto con la ortografía y redacción como con elementos relacionados con la metodología seguida y la aplicación debe ser probada para eliminar errores lógicos y de implementación, por tal razón es necesario realizar dos tipos de evaluaciones: evaluaciones estáticas y evaluaciones dinámicas.

Evaluación Estática: Busca faltas sobre el sistema en reposo. Estudian los distintos modelos que componen el sistema de software buscando posibles faltas en los mismos. Así pues, estas técnicas se pueden aplicar, tanto a requisitos como a modelos de análisis, diseño y código es decir a toda la documentación asociada a la aplicación.[9]

Evaluación Dinámica: Genera entradas al sistema con el objetivo de detectar fallos, cuando el sistema ejecuta dichas entradas. Los fallos se observan cuando se detectan incongruencias entre la salida esperada y la salida real. La aplicación de técnicas dinámicas es también conocida como pruebas de software o testing y se aplican generalmente sobre código, puesto que es, hoy por hoy, el único producto ejecutable del desarrollo.[9]

Estos dos tipos de evaluaciones comprenden los elementos que aparecen en la figura 6 y que se explicarán en detalle en los siguientes epígrafes.[10]

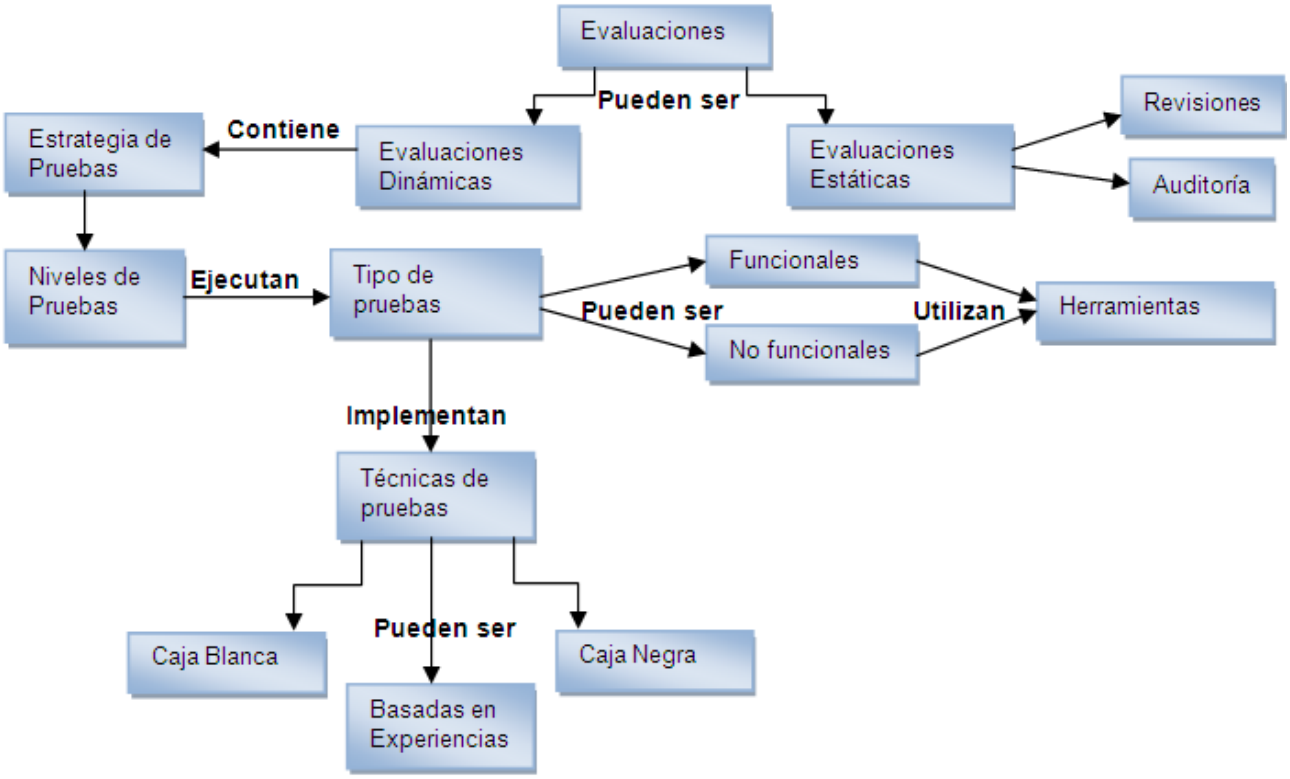


Fig. 1.2 Elementos que conforman la estrategia de evaluación de software

1.4.1. Evaluaciones Estáticas

Las técnicas de Evaluación estática de artefactos del desarrollo se las conoce de modo genérico por Revisiones. Las revisiones pretenden detectar manualmente defectos en cualquier producto del desarrollo. Manualmente quiere decir que el producto en cuestión (sea requisito, diseño, código, etc.) es analizado mediante la lectura del mismo, sin ejecutarlo.

Existen varios tipos de revisiones, dependiendo de qué se busca y cómo se analiza ese producto.

- ✓ **Revisiones formales o Inspecciones:** En las Revisiones Formales, los participantes son responsables de la fiabilidad de la evaluación, y generan un informe que refleja el acto de la revisión, estas revisiones se complementan con las pruebas (evaluaciones dinámicas).

- ✓ **Auditorías:** Las auditorías contrastan los artefactos generados durante el desarrollo con estándares, generales o de la organización. Típicamente pretenden comprobar formatos de documentos, inclusión de toda la información necesaria, etc. Es decir, no se tratan de comprobaciones técnicas, sino de gestión o administración del proyecto. [10]

1.4.2. Evaluaciones Dinámicas

La aplicación de técnicas de evaluación dinámicas se le denomina también pruebas del software. Todas comienzan definiendo los niveles, que establecen el alcance de las mismas y quienes las desarrollan. Una vez definida hacia dónde va dirigida la prueba y cuánto abarca, se debe definir qué cualidad o característica será comprobada y para ello se utilizan los tipos de pruebas a realizar. Para implementar cada tipo de prueba se hace uso de las técnicas y los mecanismos de pruebas.
Se define evaluaciones como: la determinación sistemática de hasta qué grado una entidad (empresa, software.) cumple los requisitos especificados. [2]

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que permiten el excelente desempeño de un programa, así como también la mejor publicidad que una empresa dedicada a la producción de software pueda tener. Las técnicas para encontrar problemas en un programa son extensamente variadas y van desde el uso del ingenio por parte del personal de prueba hasta herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad. Pueden intencionalmente esforzar al programa y producir errores en las respuestas para determinar si los sucesos ocurren cuando no tendrían que ocurrir o cuando los hechos no suceden cuando deberían suceder.

Para que la prueba se pueda llevar a cabo es necesario especificar las condiciones de entrada y de ejecución de la misma y los resultados esperados, que se registran en el caso de prueba. Todo lo anterior se recoge en lo que se ha denominado estrategia de pruebas que comprende:

1. Niveles de Prueba.
2. Tipos de Prueba.
3. Técnicas de pruebas.
4. Herramientas. [10]

Dado que la estrategia de pruebas, objetivo de la investigación, solo está presente en las evaluaciones dinámica se enmarcará la investigación en esta.

1.4.2.1. Niveles de Pruebas

Los niveles de prueba aplicables en las evaluaciones dinámicas se basan en dos criterios:

1. ¿Sobre qué elemento del sistema actúan las pruebas?
2. ¿Qué stakeholders desarrollan las pruebas?

Siguiendo el criterio 1 los niveles de pruebas se agrupan en:

- Unidad (sobre las clases)
- De integración (sobre los componentes del sistema)
- Sistema (sobre el sistema como un todo)

Siguiendo el criterio 2 los niveles de pruebas se agrupan en:

- De desarrollador
- Independiente
- De aceptación[9]

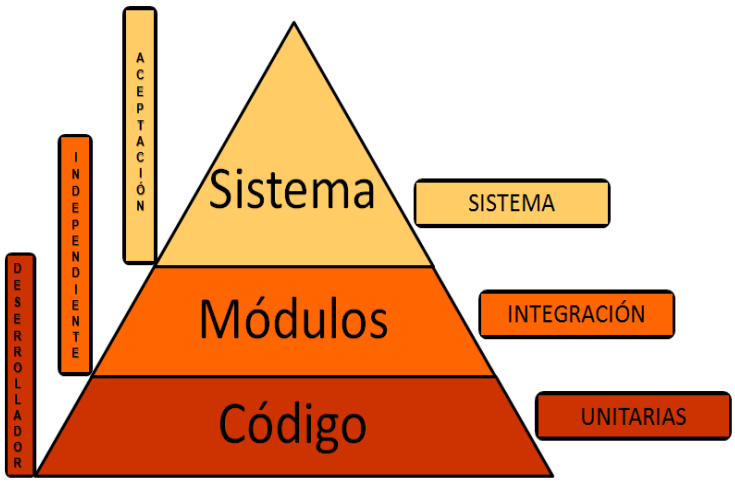


Fig. 1.3 Niveles de Pruebas de Software.

De los niveles descritos a continuación la investigación se enfocará para la estrategia que se desea proponer en el criterio 1.

- ✓ **Prueba de Unidad**

Es la prueba enfocada a los elementos probables 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. Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente.

Prueba de unidad en el contexto Orientado a Objetos (OO)

En lugar de módulos individuales, una menor unidad a probar es la clase u objeto encapsulado. Una clase puede contener un cierto número de operaciones, y una operación particular puede existir como parte de un número de clases diferentes. Esta prueba de clases para el software OO es equivalente a la prueba de unidad para el software convencional.[9]

La prueba de clases para software OO está dirigida por las operaciones encapsuladas por la clase y el estado del comportamiento de la clase. No se puede probar una operación aisladamente sino como parte de una clase.

- ✓ **Prueba de Integración**

Es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un requisito. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Estas pruebas descubren errores en las especificaciones de las interfaces de los paquetes. Esta prueba debe ser responsabilidad de desarrolladores y de independientes, sin solaparse. Es el proceso de combinar y probar múltiples componentes juntos. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Se llama integración incremental cuando el programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y corregir, es más probable que se puedan probar completamente las interfaces y se pueda aplicar un enfoque de prueba sistemática. Hay dos estrategias de integración incremental:[9]

Integración Descendente (Top-Down)

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

Integración Ascendente (Bottom-Up)

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

En el caso de integrar varios módulos y de encontrar un error en el momento de integrarlos, se tiene que hacer una Prueba de Regresión**.**

Las pruebas de integración se realizan una vez terminadas las pruebas unitarias, donde se probarán todos las CU que se implementaron durante la iteración. Durante la iteración siguiente se probarán en conjunto los resultados de la iteración-1 y la iteración en cuestión.

Prueba de Regresión

Cada vez que se añade un nuevo módulo como parte de una prueba de integración, el software cambia. Estos cambios pueden causar problemas con funciones que antes trabajaban perfectamente. La prueba de regresión es la actividad que ayuda a asegurar que los cambios no introducen un comportamiento no deseado o errores adicionales. El conjunto de pruebas de regresión contiene tres clases diferentes de casos de prueba:

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

No es práctico ni eficiente volver a ejecutar cada prueba de cada función del programa después de un cambio. Existen dos estrategias diferentes para pruebas de integración en sistemas OO:

Prueba basada en hilos:

Integra el conjunto de clases necesario para responder a una entrada o evento del sistema. Cada hilo se integra y prueba individualmente. Se aplica la prueba de regresión para asegurar que no ocurren efectos colaterales.

Prueba basada en uso:

Comienza la construcción del sistema probando aquellas clases (llamadas independientes) que usan muy pocas de las clases servidor. Luego se comprueban la próxima capa de clases, llamadas clases dependientes, que usan las clases independientes. Esta secuencia de capas de clases dependientes continúa hasta construir el sistema por completo.

Estas pruebas son realizadas cada vez que se produzca un cambio en la aplicación. Cada vez que se realiza una nueva iteración, implementándose nuevos casos de uso (CU) se realiza las pruebas de regresión. Una vez concluido el sistema o durante la fase de mantenimiento se realizan estas 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. En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos bien formados de comportamiento de caso de uso son implementados.

Para asistir en la determinación de casos de prueba de sistema, el ejecutor de la prueba debe basarse en los casos de uso que forman parte del modelo de análisis. El caso de uso brinda un escenario que posee una alta probabilidad con errores encubiertos en los requisitos de interacción del cliente. Los métodos convencionales de prueba de caja negra, pueden usarse para dirigir estas pruebas.[9]

- ✓ **Prueba de Desarrollador**

Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas pruebas han sido consideradas solo para la prueba de unidad, aunque en la actualidad en algunos casos pueden ejecutar pruebas de integración. Se recomienda que estas pruebas 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 stakeholders, que son pruebas basadas en las necesidades y preocupaciones de estos.

Prueba de liberación

Es una variante de prueba independiente definida por CALISOFT implementada en la Universidad de las Ciencias Informáticas. Su objetivo es revisar exhaustivamente un software y declararlo en estado óptimo para su entrega posterior al cliente y servir como antesala a las pruebas de aceptación.[10]

- ✓ **Prueba de Aceptación**

La prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Durante su desarrollo se verifica 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. El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario. La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual define las verificaciones a realizar y los casos de prueba asociados. Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta también los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.[10]

1.4.2.2. Tipos de Pruebas

Las pruebas se realizan con el objetivo de comprobar el cumplimiento de los requisitos funcionales y no funcionales del sistema. En dependencia del requisito que se pruebe coincidirá con el tipo de prueba a realizar, de aquí que los tipos de pruebas se clasifican en:

- ✓ Función.
- ✓ Rendimiento
- ✓ Resistencia
- ✓ Instalación.
- ✓ Validación.
- ✓ Estructura.
- ✓ Stress.
- ✓ Benchmark.
- ✓ Robustez
- ✓ Seguridad
- ✓ Usabilidad
- ✓ Recuperación.
- ✓ Volumen.
- ✓ Contención.
- ✓ Carga
- ✓ Configuración

Función: Es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático.

Rendimiento: Determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones.

Resistencia: Determinan hasta dónde puede soportar el programa determinadas condiciones extremas.

Robustez: Determinan la capacidad del programa para soportar entradas incorrectas.

Seguridad: Se determinan los niveles de permiso de usuarios, las operaciones de acceso al sistema y acceso a datos.

Usabilidad: Se determina la calidad de la experiencia de un usuario en la forma en la que éste interactúa con el sistema, se considera la facilidad de uso y el grado de satisfacción del usuario.

Validación: Son las pruebas realizadas sobre un software completamente integrado para evaluar el cumplimiento con los requisitos especificados.

Recuperación: Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.

Volumen: Prueba centrada en verificar las habilidades de los objetos de prueba para manejar grandes cantidades de datos, tanto en entrada como en salida, o residente en la base de datos. Puede incluir un

procedimiento que indique el uso de consultas que devuelvan todo el contenido de la base de datos, o cuando la cantidad de datos de entrada excede a la cantidad establecida de cada campo.

Estructura: Enfocada a la valoración a la adherencia a su diseño y formación. Este tipo de prueba es hecho a las aplicaciones Web asegurando que todos los enlaces están conectados, el contenido deseado es mostrado y no hay contenido huérfano.

Stress: Enfocada a evaluar cómo el sistema responde bajo condiciones anormales. (Extrema sobrecarga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponible).

Performance profile: Enfocadas a monitorear el tiempo en flujo de ejecución, acceso a datos, en llamada a funciones y sistema para identificar y direccional los cuellos de botellas y los procesos ineficientes.

Benchmark: Compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.

Contención: Enfocada a la validación de las habilidades del elemento a probar para manejar aceptablemente la demanda de múltiples actores sobre un mismo recurso (registro de recursos, memoria)

Carga: Usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante. La variación en carga es simular la carga de trabajo promedio y con picos que ocurre dentro de tolerancias operacionales normales.

Configuración: Enfocada a asegurar que funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema.

Instalación: Enfocada a asegurar la instalación en diferentes configuraciones de hardware y software bajo diferentes condiciones (insuficiente espacio en disco.) [10]

1.4.2.3. Técnicas de Pruebas

- ✓ **Pruebas Basadas en Experiencia**

Son aquellas pruebas que requieren de mayores competencias en el probador y están mayormente enfocadas a las experiencias obtenidas de pruebas anteriores.

- Adivinación de Errores
- Ataques al sistema
- Exploratorias

✓ **Pruebas de Caja Blanca**

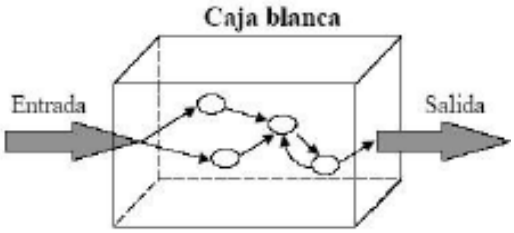


Fig. 1.4 Pruebas de Caja Blanca.

Las pruebas de caja blanca comprueban los caminos lógicos del software proponiendo casos de prueba que se ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coinciden con el esperado o mencionado.
Requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Mediante los métodos de prueba de la caja blanca, el ingeniero de software puede obtener casos de prueba que garanticen que:

1. Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
2. Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
3. Ejecuten todos los bucles en sus límites y con sus límites operacionales.
4. Se ejerciten las estructuras internas de datos para asegurar su validez.

La prueba de la caja blanca, a primera vista, podría parecer impracticable puesto que no es posible aplicarla exhaustivamente para grandes sistemas, sin embargo no se debe desechar ya que se puede elegir y ejercitar una serie de caminos lógicos importantes, que invoquen además las estructuras de datos más importantes para comprobar su validez. Se pueden combinar ambos métodos para llegar a un método que valide la interfaz del software y asegure selectivamente que el funcionamiento interno del software es correcto.[10]

Mecanismos de caja blanca

- La prueba del camino básico

- La prueba de condición
- La prueba de flujo de datos
- La prueba de bucles.[10]

La prueba del camino básico: Esta prueba permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

- 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.
- Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

➢ **Notación de Grafo de Flujo**

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo. Cada nodo del grafo corresponde a una o más sentencias de código fuente. Todo segmento de código de cualquier programa se puede traducir a un Grafo de Flujo. Para construir el grafo se debe tener en cuenta la notación para las instrucciones. Figura 2.1 y Figura 2.2.

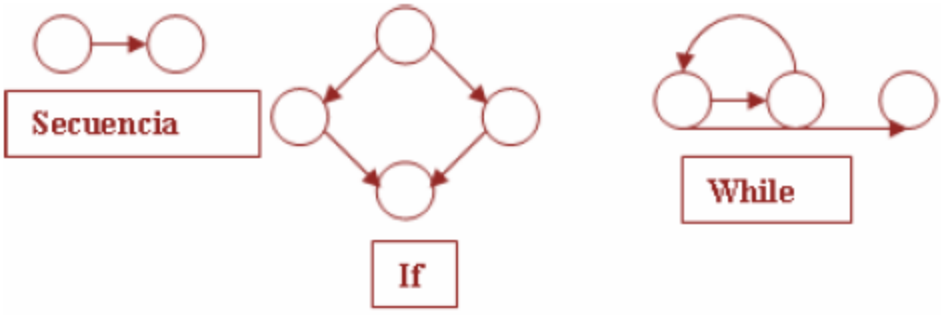


Fig. 1.5 Notación de grafos de flujos para las instrucciones: Secuenciales, if, while.

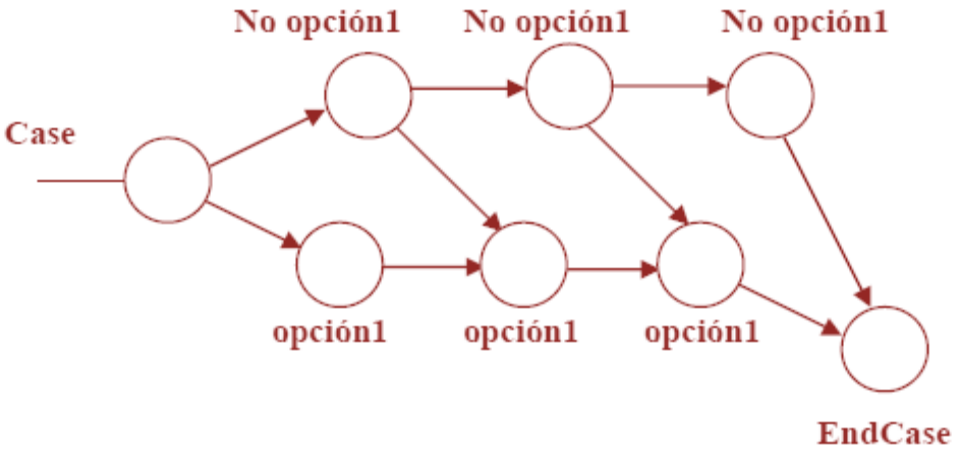


Fig. 1.6 Notación de grafos de flujos para la instrucción Case.

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

Los componentes son:

- **Nodo**

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

- **Aristas**

Las flechas del grafo se denominan aristas y representan el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

- **Regiones**

Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

Un ejemplo de representación de Grafo de Flujo es el mostrado en la Figura 2.3 en el cual aparecen sus componentes:

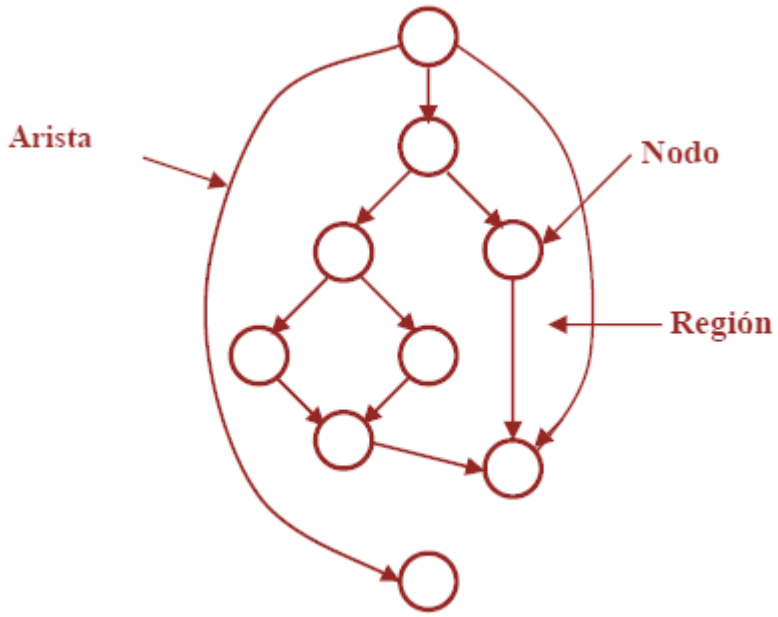
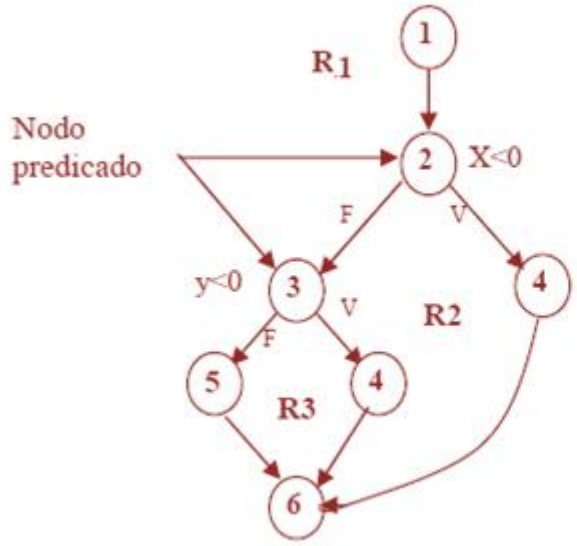


Fig. 1.7 Características de los grafos.

Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo. Cuando en un diseño se encuentran condiciones compuestas (uno o más operadores AND, NAND, NOR lógicos en una sentencia condicional), la generación del grafo de flujo se hace un poco más complicada.[8]

Ejemplo de cómo elaborar un Grafo de Flujo.

Si en un segmento de código se tiene una sentencia IF *a* OR *b* THEN entonces se crea un nodo aparte para cada una de las condiciones(*a* o *b*); cada nodo que contiene una condición se denomina nodo predicado y está caracterizado porque dos o más aristas emergen de él, Figura 2.4.




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

Fig. 1.8 Representación de un Grafo de Flujos.

- **Complejidad Ciclomática**

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente.

Ejemplo: En la Figura 1.4 un ejemplo de camino independiente sería:

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

Camino 2: 1-2-4-6.

Si se diseñan pruebas que fuercen el recorrido de esos caminos, se garantiza que se ejecute al menos una vez cada sentencia del programa y que cada condición se ejecute en sus variantes verdadera y falsa.

Se debe tener en cuenta que de un mismo diseño procedimental se pueden derivar varios conjuntos básicos.[8]

Hay tres formas fundamentales de calcular la complejidad:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

La complejidad ciclomática, *V (G)*, se define como:

V (G) = A - N + 2

Donde: *A* es el número de aristas del grafo y *N* es el número de nodos.

2. La complejidad ciclomática, *V (G)*, también se define como:

***V (G) = P* + 1**

Donde: *P* es el número de nodos predicado contenido en el grafo *G*.

Ejemplo: Teniendo en cuenta la Figura 1.4:

1. El grafo de flujo tiene tres regiones.

2. *V (G)* = 8 aristas - 7 nodos + 2 = 3.

3. *V (G)* = 2 nodos predicados + 1 = 3.

Por tanto la complejidad ciclomática del grafo de flujo de la Figura 2.4 es 3.

- **Derivación de casos de prueba**

Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino. Un ejemplo de derivación de casos de pruebas basándose en el grafo de la Figura 2.4 sería de la siguiente manera:

Casos de prueba para cada camino.

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

Escoger algún *X* y *Y* tales que cumpla *X >= 0* AND *Y >= 0.*

X = 10 AND Y = 20.

Camino 2: 1-2-4-6.

Escoger algún *X* tal que se cumpla *X < 0.*

X = -15.

Luego de confeccionar los casos de prueba se ejecutan cada uno de estos y se comparan los resultados con los esperados. Una vez terminados todos los casos de prueba, se estará seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez. Es importante considerar que algunos caminos no se pueden probar de forma aislada. O sea, la combinación de datos requeridos para recorrer el camino no se puede obtener con el flujo normal del programa. En tales casos, estos caminos se prueban como parte de otra prueba de camino.

La prueba de condición: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.

La prueba de flujo de datos: Se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

La prueba de bucles: Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

- ✓ **Pruebas de Caja Negra**

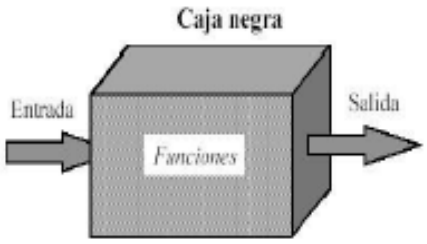


Fig. 1.9 Pruebas de Caja Negra.

También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una "Caja Negra" cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. Para seleccionar el conjunto de entradas y salidas sobre las que trabajar, hay que tener en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en nuestro sistema, y como consecuencia producen una serie de salidas que revelan la presencia

de defectos. Para confeccionar los casos de prueba de Caja Negra existen distintos mecanismos. Algunos de ellos son:

- Particiones de Equivalencia.
- Análisis de Valores Límite.
- Métodos Basados en Grafos.
- Guiada por Casos de Prueba.[10]

Particiones de Equivalencia: Según Pressman es un método de prueba que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Esta se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada.

Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

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

- ✓ Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.
- ✓ Si una entrada requiere un valor concreto, aparecen 3 clases de equivalencia: por debajo, en y por encima del rango.
- ✓ Si una entrada requiere un valor de entre los de un conjunto, aparecen 2 clases de equivalencia: en el conjunto o fuera de él.
- ✓ Si una entrada es booleana, hay 2 clases: si o no.

Los mismos criterios se aplican a las salidas esperadas: hay que intentar generar resultados en todas y cada una de las clases. Aplicando estas directrices se ejecutan casos de pruebas para cada elemento de datos del campo de entrada a desarrollar. Los casos se seleccionan de forma que ejerciten el mayor número de atributos de cada clase de equivalencia a la vez. Para aplicar esta técnica de prueba se tienen en cuenta los siguientes pasos:

Primeramente se deben identificar las clases de equivalencia lo cual se hace tomando cada condición de entrada y aplicándole las directrices antes expuestas, Figura 1.10.

Condición externa	Clases de equivalencia válidas	Clases de equivalencia Inválidas
Condición de entrada	Clases válidas para esa condición de entrada	Clases inválidas para esa condición de entrada

Tabla 1.1 Características de los Grafos.

Para definir las clases de equivalencia hace falta tener en cuenta un conjunto de reglas:

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

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

- ✓ Escribir un nuevo caso de cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.
- ✓ Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de pruebas.

Con la aplicación de esa técnica se obtiene un conjunto de pruebas que reduce el número de casos de pruebas y nos dicen algo sobre la presencia o ausencia de errores. A menudo se plantea que las pruebas a los software nunca terminan, simplemente se transfiere del desarrollador al cliente. Cada vez que el

cliente usa el programa está llevando a cabo una prueba. Aplicando el diseño de casos de pruebas al software en cuestión se puede conseguir una prueba más completa y descubrir y corregir el mayor número de errores antes de que comiencen las “pruebas del cliente”.

Análisis de Valores Límite: El análisis de valores límite (AVL) según Pressman es una técnica de diseño de casos de prueba que complementa la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.

Métodos Basados en Grafos: El primer paso en la prueba de caja negra es entender los objetos que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. Dicho de otra manera, la prueba del software empieza creando un grafo de objetos importantes y sus relaciones, y después diseñando una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir los errores.

Guiada por Casos de Prueba: Verifican las especificaciones funcionales y no consideran la estructura interna del programa. Se realiza sin el conocimiento interno del producto. No validan funciones ocultas (por ejemplo funciones implementadas pero no descritas en las especificaciones funcionales del diseño) por tanto los errores asociados a ellas no serán encontrados. [11]

1.4.3. Herramientas para pruebas de software automáticas

Las herramientas para pruebas ayudan a los equipos de desarrollo a investigar los errores de software, verifican la funcionalidad de los sistemas y se aseguran de que el software que se desarrolla es seguro y confiable, las herramientas a continuación explicadas son el resultado de una investigación realizada anteriormente, lo que facilitó la selección de las más acordes según las características del centro así como los tipos de pruebas que serán propuestos para darle cumplimiento a estas, tratando de que se utilice la menor cantidad de herramientas para de esta manera facilitarle el trabajo a los probadores.

A continuación se resume las herramientas estudiadas, todas se adquieren de forma gratuita y son de software libre.

TestNG

Es una herramienta de ejecución (y reporte de resultado) de pruebas unitarias. Esta herramienta ofrece una serie de opciones de configuración que permiten un control exhaustivo sobre la ejecución de las pruebas. Ofrece cierto orden en la ejecución de las pruebas, primero la carga de contenidos, recuperación, seguido de actualización de los mismos y por último borrado. Estas operaciones claramente requieren orden en su ejecución. [12]

PushToTest TestMaker

Es una plataforma que su funcionamiento se basa en la utilización de los diferentes test de unidad para rehusarlos como Pruebas de Integración, Funcionales, Carga, Estrés, Volumen, Regresión y posee un servicio de vigilancia que se encarga de monitorear el funcionamiento de la aplicación web cada un tiempo determinado por el usuario. Posee una gran variedad de gráficas para la visualización de los resultados de los test, así como un monitoreo del funcionamiento de los recursos (RED, CPU, Memoria) de los nodos que realizan la prueba. Las mismas pueden ser repartidas en diferentes nodos de ejecución y así lograr un escenario más real donde probar las aplicaciones web. También puede ser usado en conjunto de otros lenguajes muy utilizados en la actualidad como son: Java.NET, Jython, Groovy, PHP, Ruby para la construcción de los casos de pruebas. Además soporta Servicio Orientado a Arquitectura (SOA), AJAX, Web 2.0, y servicios REST usando sus protocolos nativos (HTTP, HTTPS, SOAP, XML-RPC, y los protocolos de correo, SMTP, POP e IMAP). Los datos para la realización de las pruebas pueden ser obtenidos a través de bases de datos o archivos de texto.

Esta herramienta cuenta con asistentes (wizards), como el soapUI 2.0.2 que es usado para la construcción de pruebas funcionales y de carga a los distintos servicios web que hayan sido creados para ser usados por las aplicaciones web. Además cuenta con grabadores (recorders), como el TestGen4Web o el Selenium, que facilitan la construcción de los diferentes casos pruebas, aún cuando no se cuenten con ninguna experiencia como programadores o aseguradores de calidad.[13]

Selenium

Es una herramienta de software libre para pruebas de aplicaciones Web. Las pruebas de Selenium se ejecutan directamente en un navegador y facilitan las pruebas de compatibilidad en navegadores, también como pruebas funcionales de aceptación de aplicaciones Web. Esta herramienta posee un ambiente de desarrollo llamado Selenium IDE, este facilita el registro de pruebas de aceptación y su depuración.[14]

JMeter

Es una herramienta de software libre que ofrece la posibilidad de realizar pruebas de carga sobre diferentes aspectos de una aplicación desarrollada en Java. JMeter es fácilmente configurable y permite conocer los tiempos de respuesta experimentados por una aplicación cuando se tiene un número N de usuarios y el número real de transacciones procesadas por unidad de tiempo. Los resultados pueden ser visualizados en diferentes formatos lo cual facilita las labores de análisis para el tester.[15]

Nessus

Nessus, una herramienta para escanear vulnerabilidades en un host o en una red de hosts, es un programa de escaneo de vulnerabilidades en diversos sistemas operativos. Consiste en nessusd, el daemon Nessus, que realiza el escaneo en el sistema objetivo, y nessus, el cliente (basado en consola o gráfico) que muestra el avance y reporte de los escaneos. Desde consola puede ser programado para hacer escaneos programados con cron.[16]

Shadow Security Scanner

Es una poderosa herramienta de control, que analiza el sistema íntegramente en busca de errores, puede ejecutar más de 4.000 tipos de auditorías diferentes, tiene un excelente desempeño analizando servicios TCP/IP, HTTPS, FTP, UDP, Registro, Windows Media Service, así como cualquier otro de los que soporta, es compatible con Windows, pero también con otros sistemas operativos (UNIX, Linux, Solaris, HP, CISCO). La aplicación permite guardar los resultados altamente detallados de las auditorías en formato RTF, PDF, HTML, CHM y XML. [17]

Brutus.A

Es un programa que permite averiguar contraseñas por el método de la fuerza bruta, es decir, probando secuencialmente cada una de las diferentes posibilidades existentes, hasta dar con la correcta, soporta múltiples protocolos de autentificación de usuarios, como por ejemplo POP3, HTTP, FTP, SMB. No implica ningún riesgo por sí mismo, pero puede ser empleado para obtener contraseñas de forma ilícita.[18]

OpenSTA

No se trata de una herramienta específica, pero si una colección de herramientas que usando en una arquitectura distribuida basada en CORBA, realiza pruebas a aplicaciones webs. Se requiere conocimiento de HTTP y de la aplicación en la que se está trabajando, por lo que no es sencillo su uso.

No cumple ninguna metodología de testing, sino que es un sistema flexible para realizar testings y obtener datos.[7]

1.5. Estrategia de pruebas de software

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

Es la línea guía del equipo de pruebas de un proyecto de desarrollo de software, son escritas, en un documento, que será entregado formalmente al líder del proyecto, para su posterior revisión y aprobación.

Se debe tener claridad en que dicho documento no debe ser visto como un entregable más, éste debe ser visto como un artefacto especialmente creado para reunir las ideas más representativas del proceso de pruebas que se llevará a cabo.

La estrategia de pruebas busca que el producto de software que se está construyendo o modificando, reúna los requisitos de lógica del negocio que el cliente ha pedido realizar mediante el debido contrato de desarrollo de software.

Hoy en día los desarrolladores se ven con la necesidad de aplicar estrategias para el desarrollo de software, ya que se vuelven más complejos, por lo tanto toda su implementación se torna de manera más complicada, por ello se han ingeniado estrategias para simplificar de alguna forma las tareas al momento de desarrollar software amplios, como técnicas y estrategias que involucran realizar pruebas, para tener software de calidad.[15]

Todas las estrategias de prueba tienen las siguientes características:

- ✓ Para realizar pruebas efectivas un equipo de software debe efectuar revisiones técnicas formales y efectivas.
- ✓ La prueba comienza al nivel de componentes y trabaja "hacia fuera", hacia la integración de todo el sistema de cómputo.
- ✓ Diferentes técnicas de prueba son apropiadas en diferentes momentos.

- ✓ La prueba la dirige el desarrollador del software y (en el caso de proyectos grandes) un grupo independiente de pruebas.
- ✓ La prueba y la depuración son actividades diferentes, pero la segunda debe incluirse en la primera. [19]

1.5.1. Componentes fundamentales de la Estrategia de Pruebas de Software

A través de la investigación se definió que la estrategia propuesta contara con los elementos que se mencionan a continuación.

- ✓ Recursos.
- ✓ Diseño de Pruebas

  -Niveles de prueba

  -Tipos.
  - Objetivo Específico.

  -Técnica de prueba
  - Mecanismos de prueba

  -Herramientas.

  -Roles y responsabilidades.

  -Duración.

  -Criterios de éxito y culminación.
- ✓ Riesgos.
- ✓ Artefactos

1.6. Conclusiones

En el presente capítulo se abordaron:

- Las características que conforman la NC ISO/IEC 9126-1: 2005.

- Los conceptos de calidad, gestión y los procesos que lo abarcan en la esfera de desarrollo de software, las pruebas de software, la estrategia de pruebas y la metodología RUP y cómo definen sus fases, roles y artefactos.
- Los elementos que conforman la estrategia de evaluación de software y sus características de forma general.

Capítulo 2 Propuesta de la Estrategia de Pruebas

2 Introducción

En este Capítulo se realiza un análisis de las características de calidad en los software del Centro de Informatización de la Seguridad Ciudadana, lo que permite realizar una propuesta de Estrategia de Pruebas para el centro, basada en estas y en la metodología de desarrollo utilizada por el ISEC. La estrategia que se propone contiene un conjunto de técnicas y herramientas con el objetivo de lograr el cumplimiento de las características de calidad identificadas en el ISEC como las de más importancia para los software que se desarrollan en dicho centro, además será aplicada a los niveles de unidad, integración y sistema. Para guiar la estrategia de pruebas se llevan a cabo las actividades principales del flujo de trabajo de pruebas, donde se planifican, diseñan, se configura el ambiente, se ejecutan y finalmente se concluyen las pruebas. Debido a que la metodología de desarrollo utilizada es RUP, la ejecución de las pruebas diseñadas como parte de la estrategia, se ejecutan durante el flujo de trabajo de Pruebas en las fases Construcción y Transición.

2.1. Trazabilidad

En el estudio realizado se obtuvo que todas las características de calidad antes mencionadas tienen tipos de pruebas que son las que le dan cumplimiento a dichas características y estos tipos se realizan en determinado nivel, todos estos datos son los que se muestran en la siguiente tabla.

Características de Calidad	Tipo	Nivel
Funcionalidad	Función	Unidad, Integración, Sistema
	Seguridad	Unidad, Sistema
	Validación	Sistema
Eficiencia	Carga	Sistema
Confiabilidad	Stress	Sistema
	Estructura	Sistema
Usabilidad	Usabilidad	Sistema

Tabla 2.1 Trazabilidad.

2.2. Actividades del flujo de trabajo prueba

La metodología de desarrollo utilizada en el ISEC, RUP propone como actividades en el flujo de trabajo de prueba planificar, diseñar, configurar escenario, ejecutar y concluir pruebas, con el objetivo de organizar el proceso de pruebas, para de esta manera garantizar la calidad que requiere el software.[8]

2.2.1. Planificar las pruebas

Al planificar las pruebas, se planifican los esfuerzos de prueba en una iteración y debe llevarse a cabo las siguientes tareas:

- Describir una estrategia de prueba: cuántos flujos básicos y alternativos deben ser probados, cuántas pruebas se realizarán automatizadas y cuántas manuales.
- Estimar los requisitos para el esfuerzo de la prueba: recursos humanos, sistemas necesarios, y los posibles riesgos que pueden presentarse.
- Planificar el esfuerzo de prueba.

Para obtener el plan de prueba, la estrategia de pruebas y los requisitos necesarios de pruebas de una iteración se debe partir de los artefactos de entrada: requisitos adicionales, modelo de casos de uso, modelo de análisis, modelo de diseño, modelo de implementación, descripción de la arquitectura (vistas arquitectónicas de los modelos).

2.2.1.1. Recursos

Los recursos materiales se identifican según los recursos necesarios para ejecutar las pruebas y las especificaciones del producto que se quiere probar, se necesitan como mínimo para la realización de las pruebas las siguientes especificaciones de hardware:

- 512 MB de RAM.
- Pentium IV con sistemas operativos Linux y Windows superior del Millenium.
- Tarjeta de Red de 10/100/1000 Giga bit.
- Disco Duro 80 Giga.
- Instalado Macromedia Flash 8.0 y Microsoft Office Access 2003 o superior.
- Navegador: Internet Explorer y Mozilla Firefox.

2.2.1.2. Riesgos

Muchas de las decisiones en un ciclo de vida iterativo como RUP se rigen por los riesgos, de ahí la importancia de detallar los que se pueden presentar durante la ejecución de las pruebas y mitigarlos. A continuación se muestran ejemplos de los riesgos que se pueden presentar según los siguientes tipos: Personas, Tiempo y Técnicos.

Tipos	Riesgos	Mitigación del riesgo
Personas	No hay suficientes personas disponibles.	Convocar la cantidad de personas necesarias.
	No tienen las habilidades y experiencias adecuadas.	Dar cursos de capacitación sobre el producto a desarrollar a los que integran el equipo.
	No han trabajado juntos antes.	Convocar a personas que ya hallan trabajando en conjunto dependiendo de sus capacidades.
Tiempo	El calendario no es realista.	Realizar un calendario que verdaderamente se adecue a las
	La fecha de entrega es crítica	La estimación de la fecha de entrega debe hacerse los más real posible.
Técnicos	Las computadoras asignadas para ejecutar las pruebas no tienen instalado el ambiente requerido para que el sistema funcione.	Inspeccionar con antelación las computadoras en las que se van a realizar las pruebas al producto con la idea que tengan el ambiente correcto para que el software funciones correctamente.

Tabla 2.2 Riesgos.

2.2.2. Diseñar pruebas

La actividad tiene como objetivo fundamental luego de identificadas las pruebas, seleccionar las técnicas, mecanismos y herramientas que se deben utilizar para el tipo de software con el que se trabaja, donde se analizan los objetivos de las pruebas que se han planificado y desarrollan los casos de prueba. La estrategia de pruebas propuesta tributa a esta actividad pues en ella se definen los tipos de pruebas a aplicar en los niveles de unidad, integración y sistema.

2.2.2.1. Nivel de unidad

✓ **Prueba Funcional**

- **Objetivo específico**

Se asegura de probar el correcto funcionamiento de las clases de un módulo, garantiza que cada uno de funcione correctamente por separado. Da cumplimiento a la característica de calidad Funcionalidad. (Ver Tabla. 2.1)

- **Técnica, mecanismo y herramienta de prueba**

La prueba de unidad siempre está orientada a la técnica de caja blanca, se hará uso del mecanismo del camino básico para el diseño de casos de prueba, se sugiere la utilización de la herramienta TestNG.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

✓ **Prueba de Seguridad**

- **Objetivo general**

Asegurar que los datos son accedidos solo por los actores que tienen autorización para ello. Verificar que el código de cada clase independientemente cumpla con las características necesarias para garantizar su seguridad. Se da cumplimiento con esta prueba a la característica de calidad Funcionalidad. (Ver Tabla 2.1)

- **Técnica, mecanismo y herramienta de prueba**

Se empleará la técnica de caja blanca, se hará uso del mecanismo del camino básico. Se sugiere el uso de las herramientas del Sistema de Análisis Estático de Vulnerabilidades (SAEV) desarrollado por el proyecto LABSI de la Facultad 2, analiza los lenguajes PHP, C++, y Python, el objetivo de este sistema es reducir la existencia de problemas de seguridad en su código.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

2.2.2.2. Nivel de integración

✓ **Prueba Funcional**

- **Objetivo general**

El objetivo de esta prueba es encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). Ejecutar las partes del sistema para comprobar que funcionan correctamente con los datos que necesitan de otros fragmentos del software. Se da cumplimiento con esta prueba a la característica de calidad Funcionalidad. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

Se empleará la técnica de caja negra y el mecanismo de partición de equivalencia y guiados por casos de prueba, se efectúa de forma manual.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

2.2.2.3. Nivel de sistema

✓ **Prueba Funcional**

- **Objetivo general**

Se asegura del trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. En esta prueba la característica de calidad Funcionalidad es a la que se da cumplimento. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

Se hará uso de la técnica de caja negra, aplicando como mecanismos el de particiones de equivalencia análisis de valores límites y guiados por casos de prueba para el diseño de casos de prueba que permita comprobar las entradas y salidas como respuesta del sistema. PushToTest TestMaker es la herramienta que se sugiere para este tipo de pruebas.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

✓ **Prueba Seguridad**

- **Objetivo general**

Verificar que los mecanismos de protección incorporados al sistema cumplen su objetivo, controlar el acceso al sistema para evitar alteraciones indebidas en los datos. Se da cumplimiento con esta prueba a la característica de calidad Funcionalidad. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

Se hará uso de la técnica basada en experiencia con su mecanismo de ataque al sistema, para esto se sugiere la utilización de la herramienta Brutus.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

- ✓ **Prueba Validación**
- **Objetivo general**

Proporciona una seguridad final de que el software satisface todos los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha sido construido. Se da cumplimiento con esta prueba a la característica de calidad Funcionalidad. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

Se hará uso de la técnica de caja negra, aplicando el mecanismo de análisis de valores límites y basados en casos de uso para el diseño de casos de prueba para analizar si la respuesta del sistema es la esperada. No se emplea herramientas para la realización de esta prueba.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

- ✓ **Prueba Stress**
- **Objetivo general**

Se asegura que el sistema funciona como se espera bajo grandes volúmenes de transacciones, usuarios, carga y demás. Esta prueba da cumplimiento a la característica de calidad Confiabilidad. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

Las técnicas que se proponen para este tipo de pruebas son las de caja negra y basadas en experiencia con los mecanismo partición de equivalencia y ataque al sistema respectivamente, se sugiere la herramienta PushToTest TestMaker.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

La prueba concluirá una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

✓ **Prueba Carga**

- **Objetivo general**

Probar el funcionamiento del software bajo condiciones extremas. Estudia la especificación del software, las funciones que debe realizar, las entradas y las salidas analizando los Valores Límite. Esta prueba da cumplimiento a la característica de calidad Eficiencia. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

La técnica para este tipo de pruebas es la de caja negra haciendo uso del mecanismo guiado por casos de prueba y análisis de valores límites, se recomendada el uso de la herramienta PushToTest TestMaker.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

✓ **Prueba Usabilidad**

- **Objetivo general**

Encaminadas a evaluar el uso natural de las interfaces gráficas de usuario de una aplicación. Se da cumplimiento con esta prueba a la característica de calidad Usabilidad. (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

Se utiliza la técnica de caja negra y el mecanismo guiado por casos de prueba, en este tipo de pruebas además de utilizar métodos como las encuestas y la opinión de expertos, no se recomienda hacer uso de herramientas.

- **Duración Estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

- ✓ **Prueba estructura**

- **Objetivo general**

Evalúa la capacidad de cómputo para comprobar la completitud de las formas estructurales y del software como un todo y así evaluar la consistencia. Este tipo de prueba es hecho a las aplicaciones Web asegurando que todos los enlaces están conectados, el contenido deseado es mostrado y no hay contenido huérfano. Se da cumplimiento con esta prueba a la característica de calidad Confiabilidad (Ver Tabla 2.1).

- **Técnica, mecanismo y herramienta de prueba**

La prueba de caja negra es la recomendada haciendo uso del mecanismo basados en grafos. Este tipo de prueba se propone hacerla de forma manual.

- **Duración estimada**

Se debe utilizar un método de estimación adecuado para que la planificación sea lo más real posible. Se propone el modelo de estimación Constructive Cost Model (COCOMO).

- **Criterios de éxito y culminación de la prueba**

Las pruebas concluirán una vez realizadas las últimas iteraciones programadas en el cronograma de trabajo y que no existan no conformidades significativas.

- **Responsable de diseño**

Diseñador de pruebas.

- **Responsable de ejecución**

Probador.

2.1.2. Implementar pruebas

El propósito de la implementación de pruebas es automatizar los procedimientos de prueba, creando componentes de pruebas cuando sea posible, casos de prueba que se han definido durante el diseño de las pruebas, automatizando todos los que sean posibles.

2.1.3. Realizar las pruebas

En esta actividad corresponde realizar las pruebas de unidad, integración y sistema al producto desarrollado, así como aplicar las listas de chequeo propuestas. En esta actividad se ejecutan las pruebas planificadas y diseñadas. Para ejecutar todas las pruebas el encargado es el Probador. Durante este

proceso se realizan reportes de No Conformidades por parte de cada probador y se registran los resultados.

2.1.4. Evaluar pruebas

Se realiza un análisis final de las pruebas aplicadas y se llevan a cabo informes finales. Los diseñadores de pruebas hacen esta actividad revisando y evaluando los resultados obtenidos de las pruebas. Luego el administrador de pruebas, se encarga de confeccionar el documento donde se registran las NC finales. Los artefactos que se generan son: casos de prueba, plan de pruebas, reporte de no conformidades (NC), reporte final de NC.

Reporte de NC

Documento elaborado por el probador donde se recogen las no conformidades detectadas en el proceso de ejecución de los distintos casos de prueba. Este documento se encuentra en el expediente de proyecto.

Reporte final de NC

Es el documento final que recoge todas las no conformidades detectadas en el proceso de pruebas. El documento se puede encontrar en el expediente de proyecto.

Los artefactos que no se describieron en este epígrafe se encuentran en el epígrafe 1.3.1.3

2.3. Conclusiones

Durante el desarrollo del presente capítulo:

- Se propusieron los tipos, técnicas, mecanismos y herramientas según el nivel correspondiente.
- Se especificaron los recursos necesarios para la ejecución de las pruebas.
- Se presentaron los roles involucrados en cada una de las pruebas.
- Se identificaron los posibles riesgos que pueden presentarse en determinada prueba y la posible manera de mitigarlos.
- Se especificaron cuáles son los artefactos que se deben tener en cuenta en una prueba.

Capítulo 3 :Validación de la propuesta

3 Introducción

En el presente capítulo se realiza la validación de la Estrategia de Pruebas para el ISEC, para ello lo ideal sería aplicar la estrategia en los proyectos que pertenecen al centro, pero debido a que estos proyectos no tienen la documentación suficiente para lograr una comparación con los resultados que han tenido ellos anteriormente con los actuales no se podrá comprobar si la estrategia es efectiva, por lo que se efectuará la validación por el método de expertos, que procede por medio de la interrogación a estos con la ayuda de cuestionarios referentes a la evaluación. Finalmente se muestran los resultados de dicha evaluación en cuanto a la factibilidad de la aplicación o no de los procesos descritos en la investigación.

3.1. Formas de Validación

En la actualidad existen tres tipos de métodos generales para evaluar proyectos, los métodos cuantitativos basados fundamentalmente en criterios económicos, los métodos multicriterios basados en los aspectos cualitativos evaluados por expertos y la revisión por pares que se basa en que el juicio del mérito lo dan expertos que trabajan la temática mediante consenso.

El presente trabajo combina los métodos cuantitativos con los multicriterios, que permiten tomar decisiones tales como aceptar o rechazar determinada propuesta, en correspondencia con los criterios definidos y la evaluación dada por estos. Tales expertos fueron seleccionados debido a la experiencia profesional que tienen en el área de calidad, la cual les permite poder realizar una valoración crítica en cuanto a la factibilidad de la estrategia propuesta.

3.2. Descripción del Método

A continuación se ofrecen los pasos seguir en la evaluación de un proyecto utilizando el método seleccionado.

Paso1: convocar y organizar un Comité de Expertos conformado como mínimo por 7 miembros.

Paso2: identificar los criterios de evaluación en función de la estrategia y agruparlos convenientemente en grupos.

Paso3: Establecer y evaluar el peso de cada grupo dependiendo de su valor representativo dentro de la estrategia.

Paso4: Entregar a cada experto la estrategia, para un análisis previo a la evaluación de la misma. Serán entregados además a cada experto tres modelos en cuestión: en el primero se podrá evaluar de forma cualitativa la apreciación de cada experto sobre la propuesta siguiendo los términos calificativos siguientes: excelente, bueno, aceptable, cuestionable y malo, también se brinda a los expertos la posibilidad de registrar criterios y consideraciones personales sobre la propuesta en el primero (Consultar Anexo 2), en el segundo se evalúan los criterios en una escala del 1 al 5, seguidamente (Consultar anexo 3). En el otro modelo (Consultar Anexo 4) será evaluado el peso relativo que asigna el experto a cada criterio según el peso total correspondiente al grupo al que este pertenece.

Una vez recogidos los modelos se comienza el proceso de verificación de la consistencia del trabajo de los expertos y se determina el índice de aceptación de la propuesta y probabilidad de éxito de la misma.

Paso5: Determinar por cada criterio en cuestión el peso promedio, partiendo de los pesos dados por los expertos.

Paso6: Verificar la consistencia del trabajo de los expertos mediante el coeficiente de concordancia de Kendall y el estadígrafo Chi Cuadrado. En caso de que no exista consistencia en los criterios de los expertos el trabajo de evaluación debe realizarse nuevamente.

Paso7: Calcular el producto de: el peso relativo de cada criterio (P) y la calificación promedio dada por los expertos (c) o sea (P x c).

Paso8: Calcular índice de aceptación de la propuesta.

Paso9: Determinar la probabilidad de éxito de la propuesta.

3.3. Aplicación del Método

Paso1: Para conformar el del Comité de Expertos han sido seleccionados 8 especialistas, valorando para su selección las siguientes características:

- Experiencia alcanzada en el campo de las Pruebas: Los especialistas seleccionados para conformar el Comité de Expertos tienen como experiencia acumulada en el área de las pruebas 2.5 años de experiencia / experto.
- Rol desempeñado por los expertos: Todos los miembros del Comité de Expertos están vinculados actualmente a la dirección de CALISOFT.
- Grado científico de los expertos: Todos los expertos consultados son ingenieros.

Paso2: Los criterios de evaluación en función de la estrategia están agrupados en los siguientes grupos:

Grupo 1 Criterios de Novedad Científica

- Calidad de la investigación
- Valor científico de la estrategia propuesta
- Carácter innovador de la propuesta

Grupo 2 Criterio de Implantación

- Claridad y precisión de la estrategia de pruebas.
- Necesidad de realizar un proceso de pruebas adecuado en el ISEC.
- Necesidad de implantación de la estrategia de pruebas en el ISEC.

Grupo 3 Criterio de Relevancia Social

- Garantiza el desarrollo de productos con la calidad necesaria para los servicios de la seguridad ciudadana.
- Posibilidad de beneficiar con la estrategia de prueba a todo el software que brinde servicios a los sectores de la seguridad ciudadana.
- Posibilidad de reducir costos económicos y de tiempo.

Grupo 4 Criterio de Adaptabilidad e Impacto

- Integración de la estrategia de pruebas en las actividades organizativas del proyecto.
- Aceptación de la propuesta por parte del encuestado.
- Adaptación de la estrategia de pruebas a proyectos pertenecientes a otro centro.

Paso3: A partir de la importancia de los criterios a evaluar se le asigna el siguiente peso a cada grupo sumando estos en total 100:

Grupo 1: Criterio de Novedad Científica----------20

Grupo 2: Criterio de Implantación----------20

Grupo 3: Criterio de Relevancia Social----------30

Grupo 4: Criterio de Adaptabilidad e Impacto-------30

Paso4: Fueron entregados los modelos a cada experto, registrándose en los modelos la información de la evaluación y la clasificación final de la misma por cada uno de ellos.

Paso5: Partiendo de los pesos dados por los expertos se realiza el cálculo por cada criterio en cuestión del peso promedio.

Grupo	C/E	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	ΣE	Exp.P
20	C_1	10	8	10	8	10	10	10	10	76	9.5
	C_2	6	6	6	6	6	5	5	6	46	5.75
	C_3	4	6	4	6	4	5	5	4	38	4.75
20	C_4	5	7	8	7	5	6	10	5	53	6.625
	C_5	10	7	6	7	10	10	5	10	65	8.125
	C_6	5	6	6	6	5	4	5	5	42	5.25
30	C_7	10	12	11	10	10	9	8	10	80	10
	C_8	10	10	9	10	13	12	12	10	86	10.75
	C_9	10	8	10	10	7	9	10	10	74	9.25
	C_{10}	11	10	10	10	10	9	8	10	78	9.75

30	C_{11}	9	10	9	12	10	12	12	13	87	10.875
	C_{12}	10	10	11	8	10	9	10	7	75	9.375
Totales		100	100	100	100	100	100	100	100	800	100

Tabla 3.1 Cálculo por cada criterio en cuestión del peso promedio

Paso6: Determinación de la consistencia en el trabajo de los expertos: Dado C el número total de criterios a evaluarse, y E el número de expertos involucrados en la evaluación, se realiza el siguiente procedimiento para determinar la consistencia en el trabajo de los expertos:

- Calcular para cada criterio: ΣE, que representa la sumatoria del peso dado por los expertos.
- Determinar el valor de PE: puntuación promedio de cada criterio.
- Calcular peso medio de cada criterio MΣE.
- Hallar el valor de ΔC, diferencia existente entre ΣE y MΣE.
- Determinar la desviación de la media, que posteriormente se eleva al cuadrado para obtener la dispersión S, dada por la expresión: $S = \Sigma (\Sigma E - \Sigma E / C)^2$.
- Conociendo la dispersión se puede calcular el coeficiente de concordancia de Kendall W, dado por la expresión: $W = S / E^2 (C^3 - C)/ 12$.
- Calcular el Chi cuadrado real a partir del valor del coeficiente de Kendall teniendo en cuenta la siguiente expresión: $X^2 = E (C-1) W$.

A continuación se muestran los datos obtenidos luego de realizar los pasos anteriores:

C/E	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	ΣE	Exp.P	ΔC	ΔC ²
C_1	10	8	10	8	10	10	10	10	76	9.5	9.4	88.36
C_2	6	6	6	6	6	5	5	6	46	5.75	20.6	424.36
C_3	4	6	4	6	4	5	5	4	38	4.75	28.6	817.96
C_4	5	7	8	7	5	6	10	5	53	6.625	13.6	184.96

C_5	10	7	6	7	10	10	5	10	65	8.125	1.6	2.56
C_6	5	6	6	6	5	4	5	5	42	5.25	24.6	605.16
C_7	10	12	11	10	10	9	8	10	80	10	13.4	179.56
C_8	10	10	9	10	13	12	12	10	86	10.75	19.4	376,36
C_9	10	8	10	10	7	9	10	10	74	9.25	7.4	54.76
C_{10}	11	10	10	10	10	9	8	10	78	9.75	11.4	129.96
C_{11}	9	10	9	12	10	12	12	13	87	10.87	20.4	416.16
C_{12}	10	10	11	8	10	9	10	7	75	9.375	8.4	70.56
Totales	100	100	100	100	100	100	100	100	800	100	178.8	2369.92
MΣE	66.6											
W	0.25											
X^2	22											

Tabla 3.2 Consistencia en el trabajo de los expertos.

Posteriormente, se compara el X^2 real, con el valor del dato estadístico, siendo $\alpha=0{,}005$, y $C=12$ y debe cumplirse que $X^2 < X^2(\alpha; c-1)$ para que el trabajo realizado por los expertos sea valorado de consistente. $X^2(\alpha; c-1) = X^2(0{,}005; 11)$ = Por tanto $X^2 < X^2(\alpha; c-1)$ lo que es, $22 < 26{,}7569$, quedando demostrada la consistencia del trabajo realizado por los expertos.

Paso7: Calcular el producto de: el peso relativo de cada criterio (P) y la calificación promedio dada por los expertos (c) o sea (P x c).

Criterios	P	c	P*c
C_1	0.095	4	0.38
C_2	0.057	3	0.171
C_3	0.047	3	0.141
C_4	0.066	4	0.264
C_5	0.081	5	0.405
C_6	0.052	5	0.26
C_7	0.1	4	0.4
C_8	0.107	5	0.535
C_9	0.092	4	0.368
C_{10}	0.097	4	0.388
C_{11}	0.108	4	0.432
C_{12}	0.093	4	0.372
Total			4.104

Tabla 3.3 Cálculo de P*C

Paso8: Calcular Índice de Aceptación (IA) de la estrategia propuesta. Partiendo de la siguiente fórmula.

IA = (P * c) / 5

Si (P * c) = 4.104 entonces IA = 4.104 / 5 obteniéndose IA =0.8208.

Paso9: Probabilidad de éxito de la estrategia propuesta.

Se determina a partir de los rangos predefinidos del índice de aceptación:

IA > 0,7. . . . . Existe Alta probabilidad de éxito.

0,7 > IA > 0,5. . . . .Existe probabilidad Media de éxito.

0,5 > IA > 0,3. . . . .Existe Baja probabilidad de éxito.

0,3 > IA. . . . . No existe probabilidad Ninguna de éxito.

Dado el resultado de IA igual a 0.8208 entonces podemos concluir que la probabilidad de éxito es: Alta

3.4. Conclusiones

- Se abordaron los tipos de métodos generales para evaluar proyectos, se seleccionó el método a aplicar según la investigación.

- Se brindó una breve descripción del método utilizado, ofreciendo una serie de pasos necesarios para su ejecución.
- Se aplicó el método de Método Experto, mostrándose los resultados obtenidos en las encuestas realizadas a los expertos obteniéndose en caso de aplicar la estrategia una probabilidad de éxito alta.

Conclusiones generales

Para la mejora del proceso de prueba en la facultad 2 específicamente para el Centro de Informatización de la Seguridad Ciudadana se debe establecer una estrategia de pruebas que permita un buen desarrollo de las mismas.

Después de la investigación realizada, se puede concluir que:

- Se estudiaron definiciones de calidad, de gestión y de pruebas de software, además de los tipos, técnicas y herramientas que posibilitaron la selección de las más adecuadas para aplicar en el centro.
- Se analizaron las características de la NC ISO/IEC 9126-1: 2005 determinando las de mayor peso dentro del ISEC luego de realizada una entrevista a los líderes de dichos proyectos.
- Se diseñó una estrategia de pruebas de software para los proyectos del ISEC.
- Se validó la propuesta establecida mediante el método de expertos obteniendo una probabilidad de éxito alta.

Recomendaciones

Para lograr el buen proceso de pruebas en los proyectos del Centro de Informatización de la Seguridad Ciudadana se recomienda:

- Aplicar la estrategia de pruebas en el centro en cuestión.
- Ampliar la estrategia de pruebas haciendo estudio de otras metodologías de software en caso de que al ISEC llegara algún proyecto que no utilice la metodología aquí analizada.
- Capacitar a todo el personal del grupo de pruebas, en los diferentes aspectos presentes en la estrategia para que puedan dar cumplimiento de manera efectiva a las actividades orientadas en la propuesta.

Referencias Bibliográficas

1. Pressman, R., ed. *Ingeniería de SW Un enfoque práctico. 2005.* 2005.
2. *ISO 8402: 1994.Gestión de la Calidad y Garantía.*
3. 9126-1, N.I. Ingeniería de Software—Calidad del Producto, 2005.
4. González, C., *Conceptos generales de la Calidad Total.*
5. Lovelle, J.M.C., *Mejora Continua en el Desarrollo del Software.*
6. Sánchez, M.a.M., *Metodologías de desarrollo de software.*
7. *Ayuda Extendida de RUP.* 2003.
8. Ivar Jacobson, G.B., James Rumbaugh, *El proceso unificado de desarrollo de software.* 2000.
9. Aguilar, M.V.H., *Estrategia de Evaluaciones.*
10. Tecnológicos, C.d.E.p.e.D.d.P., *Estrategia de Evaluación de Software.*
11. Pressman, R.S., *Ingeniería de Software.* 2005.
12. Muñiz, A.M. *Ordenando nuestras pruebas*. [Disponible en: http://amunizmartin.wordpress.com/2008/06/21/pruebas-con-testng-ordenando-nuestras-pruebas/.
13. Pérez, K.A.C., *Propuesta de herramientas para pruebas de software automáticas*, Universidad de las Ciencias Informáticas(UCI).
14. José Bertolini, M.L., Nicolás Stobbia, *FDD:Feature Driven Development Desarrollo basado en funcionalidades.*
15. Anna. C Grimán, M.P., Luis. E Mendoza, *Estrategia de Pruebas para Software OO que garantiza Requerimientos No Funcionales.*

16. *Conociendo herramienta Nessus, escaneador de vulnerabilidades*. [Disponible en: http://www.arrayexception.com/desarrollo/seguridad/conociendo-la-herramienta-nessus-escaneador-de-vulnerabilidades.

17. *RUP & ISO 9126.*

18. *Introducción conceptos de Calidad*. [Disponible en: http://mgar.net/soc/isointro.htm#def1.

19. Harvey Alférez. *Estrategias de Prueba del Software* [Disponible en: http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=2&ved=0CA4QFjAB&url=http%3A%2F%2Ffit.um.edu.mx%2Fharvey%2Ffiles%2F2008-2009%2F1st%2520Semester%25202008-2009%2FIngenier%25C3%25ADa%2520del%2520Software%2F13.pptx&rct=j&q=pasos+para+la+estrategia+de+pruebas+de+software&ei=T2uWS4_1B8qJlAfZs6CHDQ&usg=AFQjCNEGGrYuNyTrr7Daa0B159dzisgyFg.

Bibliografía Consultada

Estrategia de pruebas. (n.d.). Retrieved febrero 2, 2010, from http://technet.microsoft.com/es-es/library/dd567668.aspx

Torío, J. M. *Estrategias de pruebas de líneas de producto de sistemas de tiempo real especificados con diagramas de estados jerárquicos.* Retrieved febrero 2, 2010, from http://oa.upm.es/246/

Casañola, Y. T (n.d.). *Apuntes de la disciplina de ambiente del proceso unificado de desarrollo (RUP)*.Retrieved febrero 2,2010, from http://www.uci.cu/files/investigaciones/vol2/SC%20Num%201.pdf

Pressman, R., ed. *Ingeniería de SW Un enfoque práctico. 2005.* 2005.

ISO 8402: 1994.

9126-1, N.I. Ingeniería de Software—Calidad del Producto, 2005.

González, C., *Conceptos generales de la Calidad Total.*

Lovelle, J.M.C., *Mejora Continua en el Desarrollo del Software.*

Sánchez, M.a.M., *Metodologías de desarrollo de software.*

Ayuda Extendida de RUP. 2003.

Ivar Jacobson, G.B., James Rumbaugh, *El proceso unificado de desarrollo de software.* 2000.

Aguilar, M.V.H., *Estrategia de Evaluaciones.*

Tecnológicos, C.d.E.p.e.D.d.P., *Estrategia de Evaluación de Software.*

Pressman, R.S., *Ingeniería de Software.* 2005.

Muñiz, A.M. *Ordenando nuestras pruebas*. [Disponible en: http://amunizmartin.wordpress.com/2008/06/21/pruebas-con-testng-ordenando-nuestras-pruebas/.

Pérez, K.A.C., *Propuesta de herramientas para pruebas de software automáticas*, Universidad de las Ciencias Informáticas(UCI).

José Bertolini, M.L., Nicolás Stobbia, *FDD:Feature Driven Development Desarrollo basado en funcionalidades.*

Anna. C Grimán, M.P., Luis. E Mendoza, *Estrategia de Pruebas para Software OO que garantiza Requerimientos No Funcionales.*

Conociendo herramienta Nessus, escaneador de vulnerabilidades. [Disponible en: http://www.arrayexception.com/desarrollo/seguridad/conociendo-la-herramienta-nessus-escaneador-de-vulnerabilidades.

RUP & ISO 9126.

Introducción conceptos de Calidad. [Disponible en: http://mgar.net/soc/isointro.htm#def1.

Harvey Alférez. *Estrategias de Prueba del Software* [disponible en: http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=2&ved=0CA4QFjAB&url=http%3A%2F%2Ffit.um.edu.mx%2Fharvey%2Ffiles%2F2008-2009%2F1st%2520Semester%25202008-2009%2FIngenier%25C3%25ADa%2520del%2520Software%2F13.pptx&rct=j&q=pasos+para+la+estrategia+de+pruebas+de+software&ei=T2uWS4_1B8qJlAfZs6CHDQ&usg=AFQjCNEGGrYuNyTrr7Daa0B159dzisgyFg.

Glosario de Términos

Artefacto: Método o proceso de desarrollo específico. Información que es utilizada o producida mediante un proceso de desarrollo de software.

Auditorías y Revisiones: Dirección de CALISOFT donde se planifican y ejecutan las inspecciones, comprobaciones, y revisiones a los proyectos de la universidad.

Aplicaciones Web: Aplicaciones que los usuarios pueden utilizar accediendo mediante un navegador.

Benchmark: Tipo de prueba de software que compara el rendimiento de un elemento nuevo o desconocido a uno de carga de trabajo de referencia conocido.

CALISOFT: Dirección de Calidad de Software de la UCI que garantiza el crecimiento continuo de una producción de software con calidad en la organización

CMS: Configuration Management System.

CU: Caso de Uso.

COCOMO: Constructive Cost Model

Informatización: Aplicar los métodos de la informática en un negocio, un proyecto, etc.

ISEC: Centro de Informatización de la Seguridad Ciudadana.

Java: Lenguaje de programación orientado a objetos.

LABSI: Proyecto de la Facultad 2 referente a la seguridad informática.

LIPS: Laboratorio de Pruebas de Software.

Navegador: Aplicación que se utiliza para navegar por Internet y que permite visualizar la información contenida en las páginas web.

NC: No Conformidades.

OO: Orientado a objetos.

Oracle: Sistema de gestión de base de datos relacional desarrollado por Oracle Corporation.

Performance profile: Tipo de prueba de software enfocadas a monitorear el tiempo en flujo de ejecución, acceso a datos, en llamada a funciones y sistema para identificar y direccional los cuellos de botellas y los procesos ineficientes.

Proyecto productivo: En la UCI está definido como un equipo de trabajo compuesto por estudiantes y profesores con el objetivo de desarrollar software para elevar la economía del país.

Prueba: Proceso al que es sometido un producto para verificar su calidad.

Roles: Papel que juega cada integrante de un proyecto productivo.

RUP: Rational Unified Process.

SAEV: Sistema de Análisis Estático de Vulnerabilidades.

Software: Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

Stakeholders: Quienes pueden afectar o son afectados por las actividades de una empresa.

UCI: Universidad de las Ciencias Informáticas.

Anexo 1

Entrevista sobre las características del Software según la NC ISO 9126 para los proyectos del ISEC

Marzo, 2010. "Año del 51 aniversario del Triunfo de la Revolución Cubana"

La siguiente encuesta es parte del proceso de investigación del Trabajo de Diploma ***"Propuesta de estrategia de pruebas para el Centro de Informatización Seguridad Ciudadana"*** realizado en la Universidad de las Ciencias Informáticas. Su objetivo es analizar las características de calidad de mayor importancia para los software de Seguridad Ciudadana. Gracias por su colaboración.

1. ¿De las características marque con una X cuáles son las que más se evidencian en el software para la Seguridad Ciudadana? Diga cuál o cuáles considera de más importancia (Funcionalidad, Confiabilidad, Usabilidad, Eficiencia, Mantenibilidad, Portabilidad)? ¿Por qué?

   Funcionalidad

   - Precisión: Capacidad del software para proporcionar efectos o resultados correctos o convenidos con el grado de exactitud necesario.
   - Seguridad: Asegurar que los datos o el sistema solamente es accedido por los actores deseados.
   - Interoperabilidad: Capacidad del producto de software para interactuar recíprocamente con uno o más sistemas especificados.

   Confiabilidad

   - Madurez: Capacidad del producto de software de evitar un fallo total como resultado de haberse producido un fallo del software.
   - Tolerancia ante fallos: Capacidad del producto de software de mantener un nivel de ejecución o desempeño especificado en caso de fallos del software o de infracción de su interface especificada.

- Recuperabilidad: Capacidad del producto de software de restablecer un nivel de ejecución especificado y recuperar los datos directamente afectados en caso de fallo total.

Usabilidad

- Comprensibilidad: capacidad del producto de software para permitirle al usuario entender si el software es idóneo, y cómo puede usarse para las tareas y condiciones de uso particulares.
- Operabilidad: capacidad del producto del software para permitirle al usuario operarlo y controlarlo.
- Atracción: capacidad del producto del software de ser atractivo o amigable para el usuario.

Eficiencia

- Rendimiento: capacidad del producto de software para proporcionar apropiados tiempos de respuesta y procesamiento, así como tasas de producción de resultados, al realizar su función bajo condiciones establecidas.
- Utilización de recursos: capacidad del producto de software para utilizar la cantidad y el tipo apropiado de recursos cuando el software realiza su función bajo las condiciones establecidas.

Mantenibilidad

- Flexibilidad: capacidad del producto del software para permitir la aplicación de una modificación especificada.
- Estabilidad: capacidad del producto de software para minimizar los efectos inesperados de las modificaciones realizadas al software.
- Contrastabilidad: capacidad del producto del software para permitir la validación de un software modificado.

Portabilidad

- Adaptabilidad: capacidad del producto de software de ser adaptado a los ambientes especificados sin aplicar acciones o medios de otra manera que aquellos suministrados con el propósito de que el software cumpla sus fines.

- Instalabilidad: capacidad del producto de software de ser instalado en un ambiente especificado.
- Remplazabilidad: capacidad del producto de software de ser usado en lugar de otro producto de software especificado para los mismos fines y en el mismo ambiente.

Anexo 2

Cuestionario de evaluación de la Propuesta de estrategia de pruebas de software para el Centro de Informatización de la Seguridad Ciudadana (ISEC)

Compañero (a):

En el desarrollo de la presente investigación, se desea contar con su criterio como experto para validar la Propuesta de estrategia de pruebas de software para el Centro de Informatización de la Seguridad Ciudadana (ISEC). Usted como experto se le solicita evaluar la propuesta según el grado de aceptación que esta tenga a partir del cumplimiento del objetivo general de la investigación.

Con este fin se requiere que responda el cuestionario a continuación presentado.

Datos personales:

Nombre y apellidos:

Proyecto:

Rol:

Años de experiencia:

1. ¿Considera usted que la estrategia da cumplimiento a las características de calidad identificadas para el software desarrollado en el Centro?
   Justifique según su respuesta.

   Si __ No __

2. Considera que los componentes identificados para la estrategia de pruebas son los adecuados.

Sin cree que se debe incorporar alguno menciónelo.

Si __ No__

_____

_____

_____

_____

_____

3. ¿Considera que el conjunto de técnicas y herramientas descritas en la estrategia de pruebas garantizan la calidad del software desarrollado en el Centro ISEC? Justifique según su respuesta.

   Si __ No __

   _____

   _____

   _____

   _____

   _____

Anexo 3

1. Asigne el peso relativo a cada criterio en una escala del 1 al 5.

Grupo1: Criterio de Novedad Científica	
Criterios	**Peso**
Calidad de la investigación.	
Valor científico de la propuesta.	
Carácter innovador de la propuesta.	
Grupo 2: Criterio de Implantación	
Claridad y precisión de la estrategia de pruebas.	
Necesidad de realizar un proceso de pruebas adecuado en el Centro de Informatización de la Seguridad Ciudadana.	
Necesidad de implantación de la estrategia de pruebas en el Centro de Informatización de la Seguridad Ciudadana.	

Grupo 3: Criterio de Relevancia Social	
Garantiza el desarrollo de productos con la calidad necesaria para los servicios de la seguridad ciudadana.	
Posibilidad de beneficiar con la estrategia de prueba a todo software que brinda servicios los sectores de la seguridad ciudadana.	
Posibilidad de reducir costos económicos y de tiempo.	
Grupo 4: Criterio de Adaptabilidad e Impacto	
Integración de la estrategia de pruebas en las actividades organizativas del proyecto.	
Aceptación de la propuesta por parte del encuestado.	
Adaptación de la estrategia de pruebas a proyectos pertenecientes a otro centro.	

2. Categoría final de la propuesta.

____ Excelente: alta novedad científica, grandes posibilidades de aplicabilidad y relevantes resultados esperados.

___ Bueno: novedoso científicamente y con buenos resultados esperados.

___ Aceptable: no es lo suficientemente bueno, pero puede aplicarse.

___Cuestionable: sin relevancia científica y con resultados esperados no satisfactorios.

____ Malo: no aplicable

3. Evaluación final:

- Sugerencias del evaluador para mejorar la estrategia propuesta.

_____

_____

_____

_____

_____

- Elementos que deben mejorarse.

_____

_____

_____

_____

_____

Anexo 4

1. Asigne el peso relativo a cada criterio según el peso establecido por cada grupo de criterio, es decir, a cada criterio se le asigna un peso relativo según el porciento que representa el grupo del total de acuerdo con las características de la convocatoria y sus intereses.

Grupo1: Criterio de Novedad Científica… 20	
Criterios	**Peso**
Calidad de la investigación.	
Valor científico de la propuesta.	
Carácter innovador de la propuesta.	
Grupo 2: Criterio de Implantación… 20	
Claridad y precisión de la estrategia de pruebas.	
Necesidad de realizar un proceso de pruebas adecuado en el Centro de Informatización de la Seguridad Ciudadana.	
Necesidad de implantación de la estrategia de pruebas en el Centro de Informatización de la Seguridad Ciudadana.	
Grupo 3: Criterio de Relevancia Social… 30	
Garantiza el desarrollo de productos con la calidad necesaria para los servicios de la seguridad ciudadana.	
Posibilidad de beneficiar con la estrategia de prueba a todo el software	

que brinda servicios los sectores de la seguridad ciudadana.	
Grupo 4: Criterio de Adaptabilidad e Impacto… 30	
Integración de la estrategia de pruebas en las actividades organizativas del proyecto.	
Aceptación de la propuesta por parte del encuestado.	