UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 6

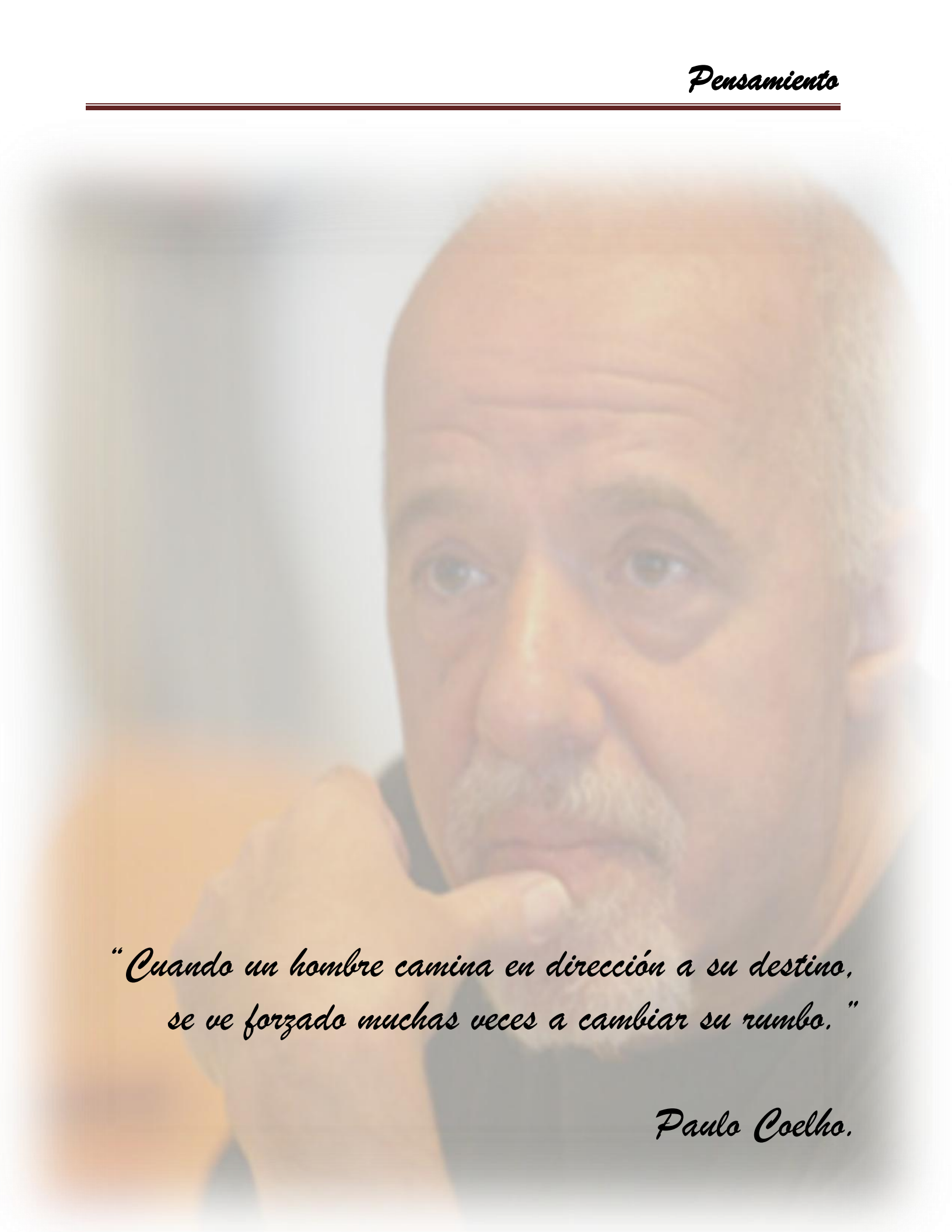
TÍTULO:" **Diseño y aplicación de pruebas al proyecto Video Vigilancia."**

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

AUTOR: **Mailén González Berenguer.**

TUTOR: **Ing. Erieilis Reyes González.**

La Habana, Cuba. Junio ,2011
"Año 53 de la Revolución."



"Cuando un hombre camina en dirección a su destino,
se ve forzado muchas veces a cambiar su rumbo."

Paulo Coelho.

Dedicatoria

Este trabajo de diploma, lo dedico con todo mi amor y cariño a las dos personas que me dieron la vida y la felicidad de crecer en una familia maravillosa. A las dos personas que guiaron mis pasos siempre por el buen camino, que hicieron de mí una persona de bien, de éxito; que me ayudaron a dar hoy este paso tan importante para mí, por no dejarme nunca a pesar de la distancia. Sin ustedes no hubiese podido lograr este éxito. Todo el esfuerzo realizado por mi está dedicado a ustedes mamá y papá.

Agradecimientos

Quiero agradecer primeramente a mi familia que son la razón de mi vida, especialmente a mi mamá, a mi papá y a mis tíos.

A mi abuelo Benito que siempre me ha dado ánimos para seguir adelante y a mi abuelita María que aunque ya no esté conmigo se que está muy orgullosa de mi.

A mi hermano y a mis primos, en especial a (Lewis) que me han apoyado en todo momento y siempre estuvieron ahí para mí.

A mi tutora Erieilis Reyes un GRACIAS especial, por guiarme todo este tiempo y brindarme su ayuda incondicional.

A mis compañeras de apto (Eliadys (la pina), Yanet (la clarea), Lisandra (la miche), Leidys, Lusmey, Elizabetha, Yara, Nixys, Yugle, Yanet (palmolive)) a todas ellas gracias por su amistad.

A mis amigas incondicionales (Elsa, Yane, Beylen, Sucel, Solecita, Grechita, Yas, Miriam, Liset, Sandra) a ellas muchas gracias por soportarme y brindarme su amistad todo este tiempo.

A (Yosiel, Yansel, Alejandro, Lucho) que a pesar de conocerlos hace muy poco tiempo los aprecio y estimo como amigos.

En fin, a todos los demás que aunque no los mencione, no dejan de ser importantes para mí. Gracias por ayudarme a llegar hasta aquí.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Mailén González Berenguer	Ing. Erieilis Reyes González
_____	_____
Firma del Autor	Firma del Tutor

OPINIÓN DEL TUTOR

Resumen

El propósito del presente trabajo de diploma es ejecutar el plan de prueba del producto Video Vigilancia. Para ello fue necesario realizar una fundamentación teórica en la que se abordan los diferentes conceptos emitidos por varios autores referentes a aspectos relacionados con la calidad de software y las pruebas de software en el proceso de desarrollo.

Dentro de esas actividades estuvo la aplicación de las pruebas de Caja Blanca y Caja Negra para lo cual se diseñaron los casos de prueba utilizando las técnicas del Camino Básico y Partición Equivalente respectivamente. Además, se realizaron pruebas de Integración y de Sistema, con las técnicas de Integración Incremental Ascendente y Prueba de Volumen respectivamente. Se archivaron los resultados obtenidos de la ejecución del proceso y se realizó una evaluación de cada parte probada, lo que permitió la detección de un gran por ciento de errores.

Todo lo antes planteado garantiza un producto de software con la calidad requerida y con un alto nivel técnico dentro de la institución que lo desarrolla.

PALABRAS CLAVES

Calidad, Casos de prueba, Estrategia, Pruebas de software, Plan de prueba.

Índice de figuras

Índice de tablas

Índice

Introducción

Los sistemas de video vigilancia han atravesado por varias etapas de su desarrollo, desde simples sistemas constituidos básicamente por una cámara conectada a un monitor y a un grabador de cinta, para de ser necesario, grabar un determinado acontecimiento delictivo, hasta los modernos sistemas actuales de tercera generación, basados en cámaras IP y redes de alta velocidad.

Los mismos tienen cada vez mayor demanda, especialmente, para garantizar la seguridad de interiores y alrededores de edificios. La presencia de numerosas cámaras de seguridad en cualquier entorno urbano es un hecho. Gracias a la evolución tecnológica se ha logrado que instalar cámaras de captura de vídeo no precisa de altas inversiones económicas y, por tanto, la mayoría de bancos, estaciones, aeropuertos, tiendas, parkings, etc., incorporan en sus instalaciones algún sistema, más o menos complejo, de seguridad basado en vídeo vigilancia.

Actualmente, se está produciendo una migración de los sistemas de vídeo clásicos a los sistemas de tercera generación. Estos aprovecharán el progreso de las redes de ordenadores de bajo costo y alto rendimiento; las comunicaciones multimedias fijas y móviles. Con esto se provee que todos estos trabajos proporcionen a las aplicaciones de vigilancia, resultados cada vez más interesantes, gracias a la disponibilidad de una potencia de cálculo muy elevada a unos precios aceptables, por lo que el desarrollo de las redes de acceso de banda ancha va a permitir a clientes residenciales utilizar estos sistemas para diferentes aplicaciones.

Sin embargo, los sistemas de vigilancia presentan requerimientos específicos que implican una necesidad de investigación dedicada y el desarrollo de nuevas herramienta.

El gobierno revolucionario cubano a partir del año 2002 da inicio a un proyecto que intenta preparar las bases para masificar el conocimiento de la informática en la sociedad actual, con el objetivo de crear software que reporten beneficios económicos y sociales entre otras tareas. Este proyecto es conocido como Universidad de las Ciencias Informáticas (UCI), como parte del proceso de informatización del país surge el proyecto Video Vigilancia en la universidad con el objetivo de desarrollar video sensores para el seguimiento de objetos, y otros que permitan realizar tareas de detección de movimiento y segmentación de imágenes. Pero fundamentalmente surge con el propósito, en todo momento, de crear un sistema completamente modular que

proporcione una gran flexibilidad. Dicho proyecto en la actualidad cuenta con las primeras versiones de la estación de visualización y del módulo central gestor de todo el sistema.

En vista de las exigencias de la industria del software de hoy día, ha sido necesario y de suma importancia hacer programas que posean una buena calidad, para poder entrar en el mercado tanto nacional como internacional. El incremento de la complejidad de los productos informáticos aumenta; unido a esto crece de forma acelerada la necesidad de asegurar la calidad de los mismos.

Por esto, el producto Video Vigilancia como todo software, debe ser probado y evaluado por un grupo de desarrolladores y un equipo dedicado a garantizar su calidad dentro de la institución que lo desarrolla antes de ser desplegado en manos del cliente.

Hay que tener en cuenta que dicho producto de software no va a estar al 100% libre de errores, ya que las pruebas que se le realizarán al mismo son para garantizar que tenga la menor cantidad de faltas posibles. Se debe considerar que las pruebas que se le aplicarán a dicho producto suelen ser costosas y molestas pero su realización va a garantizar que el producto cuente con la calidad requerida y a su vez se logre la satisfacción del cliente.

Teniendo en cuenta estas ideas se tiene como **problema a resolver**: ¿Cómo medir la calidad del producto Video Vigilancia a partir de la validación y verificación de los requisitos planteados por el cliente?, el **objeto de estudio** de esta investigación está enmarcado en el proceso de pruebas para proyectos de software, teniendo como **campo de acción** los diseños de casos de prueba del proyecto Video Vigilancia. Se concibe como **objetivo general**: Diseñar y aplicar pruebas al proyecto Video Vigilancia que permita certificar la calidad del producto.

Como **idea a defender** se plantea que: Con el diseño y aplicación de un plan de pruebas al proyecto

Video Vigilancia, se garantizará detectar la mayor cantidad de errores posibles.

Para dar cumplimiento al objetivo se definen las siguientes **tareas de la investigación:**

1. Caracterización de los diferentes tipos de prueba que existen para conformar el marco teórico de la investigación.
2. Definición de los tipos de pruebas a realizar en el proyecto.
3. Definición de los procedimientos para cada una de las pruebas seleccionadas tanto las de caja blanca como las de caja negra.

4. Diseño de los casos de prueba para cada uno de los casos de uso del sistema.
5. Confección del cronograma con todas las pruebas que se van a aplicar.
6. Ejecución de las pruebas planificadas al sistema.
7. Documentación del resultado de las pruebas.

Posibles resultados:

- Plan de prueba para el proyecto Video Vigilancia.

Para darle cumplimiento a las tareas propuestas se utilizaron diferentes métodos de investigación que posibilitaron conseguir resultados favorables, estos fueron:

Empíricos:

La observación: Este se utilizó durante toda la investigación, pues se recoge información de los aspectos tratados en la tesis desde el punto de vista de otros autores así como sus definiciones y resultados para permitir realizar un análisis del arte en ese campo.

Teóricos:

Análisis y Síntesis: Ha permitido analizar las teorías y los documentos referentes al objetivo de la investigación, posibilitando de esta forma la extracción de los elementos más importantes relacionados con el objeto de estudio. Además de permitir la construcción de una teoría y el camino a seguir, a partir del análisis detallado de cada uno de los documentos a utilizar durante la investigación.

Análisis histórico y el lógico: El método facilitó la realización de la primera parte de la investigación, permitiendo hacer un análisis a nivel nacional e internacional de las empresas que producen software de Video Vigilancia.

El contenido de este trabajo está estructurado de la siguiente manera:

En el capítulo 1, “Fundamentos Teóricos” se realizará un estudio de todos los aspectos teóricos relacionados con el proceso de pruebas que se le aplican al software y se abordan los conceptos asociados a este proceso. Además del estudio de los niveles y métodos más importantes de pruebas.

En el capítulo 2, “Plan de prueba” se mostrarán todas las actividades del Plan de Prueba incluyendo la Estrategia de Prueba, expone los tipos de pruebas que se le realizaron al proyecto

Video Vigilancia, sus objetivos, las técnicas aplicadas para la realización de las mismas así como los recursos necesarios para llevar a cabo un buen proceso de detección de errores.

En el capítulo 3, “Diseño, ejecución y evaluación de las pruebas” se realizarán los diseños de casos de pruebas y se aplicarán las pruebas definidas en el Plan de pruebas, para hacer una evaluación de todos los defectos y dificultades encontrados en las funcionalidades escogidas; de esta manera se logra una mayor calidad en el producto y que este cuente con la menor cantidad de defectos posibles.

CAPÍTULO 1: Fundamentación Teórica.

1. Introducción.

En el capítulo se abordan los temas fundamentales referidos a la ingeniería de pruebas, así como los conceptos y las definiciones expuestos por varios autores. Se profundiza en los tipos de pruebas además de sus técnicas para verificar y validar el producto desarrollado, también se mostrará la metodología de desarrollo de software escogida en el proyecto para llevar a cabo dicho proceso de pruebas.

1.1 Metodología de desarrollo de software:

Las metodologías de desarrollo de software especifican un conjunto de criterios que rigen la forma en que se emplea la ingeniería del software en un proyecto productivo; o sea, es la base para la realización de un proyecto de software, la principal etapa para obtener los objetivos buscados con dicho proyecto. Que no se utilice la metodología adecuada en un proceso de desarrollo de un proyecto también garantiza con seguridad la poca o nula calidad del mismo. La producción de un software con calidad involucra el uso de metodologías o estándares para el análisis, diseño, programación y prueba de este, que permitan equilibrar la estética de trabajo, para así lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, que a la vez realcen la productividad, tanto para la labor de desarrollo como para el control de la calidad del software. La idea es controlar de forma transparente el proceso de desarrollo completo, esencialmente permite producir sistemas en el tiempo planificado para este y con el costo estimado, esto es importantísimo pues en la mayoría de las veces se planea hacer algo que finalmente toma más tiempo de lo planeado. (McCall.J, 1977)

¿Por qué aplicar RUP en el proyecto Video Vigilancia?

Se aplica RUP porque sus características son factibles para la realización del producto, primero porque el Proceso Unificado es una propuesta de proceso para el desarrollo de software orientado a objetos que utiliza lenguaje modelado unificado (UML) como único lenguaje para describir el proceso. Además el mismo como metodología de desarrollo al recorrer cada una de sus 4 fases, especifica una serie de disciplinas dentro de las cuales están las pruebas. Disciplina que se encuentra presente en cada una de estas fases, pero que alcanza su mayor peso en la fase de construcción. Dicha disciplina es de mucha importancia, ya que va a ser la encargada de probar la calidad del producto final.

1.2 Calidad del software. Conceptos y definiciones

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La misma es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. Además de ser el desarrollo de software basado en estándares con la funcionalidad y rendimiento total que satisfacen los requerimientos del cliente. (Oscar M. Fernández Carrasco, 1995)

La calidad del software es la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" R.S. Pressman (1992), además se pudiera tener en cuenta que puede ser el grado con el que un sistema, componente o proceso cumple los requerimientos específicos y las necesidades o expectativas del cliente o usuario". (IEEE, 1990), también es la propiedad o conjunto de propiedades inherentes a un objeto que permiten apreciarlo como mejor, igual o peor que otros objetos de su especie" [DRAE: Diccionario de la Real Academia Española]

En cierta medida todos los conceptos expuestos hacen referencia a lo que es la calidad de software por lo que se puede definir de forma general que la calidad de software no es más que: el grado con el que un componente o un software cumplen con las necesidades del cliente, que cuenta con una serie de parámetros definidos y lo dotan de cualidades únicas, cumpliendo la totalidad de características de un producto de software, posibilitando satisfacer las necesidades explícitas o implícitas del usuario.(Pressman, 2002)

El término calidad de software siempre se entenderá de disímiles maneras por cada persona, ya que para algunas la calidad reside en un producto y para otras en el servicio posventa de la misma. Lo cierto es que nunca se alcanzará definir exactamente lo que representa dicho término a pesar de que en los últimos tiempos se ha puesto de moda el mismo.
Analizando estos planteamientos, si un software es desarrollado por alguien que estudió y se preparó para eso, o sea, un profesional en esa rama, el mismo debe tener una calidad excelente, además para

obtener esta calidad, el producto debe brindar a sus usuarios todas las posibilidades que estos pidieron al solicitar la creación del software. Siguiendo con el tema de la calidad, se debe tener en cuenta que para lograr esta calidad de la que se habla, se debe comprobar que el software en cuestión cumpla con los factores de calidad requeridos y definidos por la misma.

1.3 Factores en la calidad del software

El término de calidad de software es muy amplio por lo que se interpreta de varias maneras. Una de las definiciones de calidad más publicada es la planteada por (McCall, 1977), la cual específica una serie de factores. Estos factores a pesar de haberse definido hace mucho tiempo, conservan en gran medida su vigencia en la actualidad.

Algunos de estos factores son:

- *Confiabilidad*: Es el grado en que se puede esperar que un programa lleve a cabo sus funciones esperadas con la precisión requerida. La pregunta asociada a este factor sería: *¿Lo hace de forma fiable todo el tiempo?*
- *Eficiencia:* Es la cantidad de recursos de computadoras y de código requeridos por un programa para llevar a cabo sus funciones, además de la capacidad para hacer un buen uso de los recursos del ordenador.La pregunta asociada a este factor sería: *¿Se ejecutará en mi hardware lo mejor que pueda?*
- *Portabilidad*: Es el esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistema de software a otro. Este factor tiene una pregunta asociada: *¿Podré usarlo en otra máquina?*
- *Facilidad de Prueba:* Es el esfuerzo requerido para probar un programa de forma que se asegure que realiza su función requerida. La pregunta asociada a este factor sería: *¿Puedo probarlo?*
- *Funcionalidad:* Es el conjunto de posibilidades que proporciona el software. Este factor de calidad tiene características asociadas como las que se refieren a las capacidades de un programa, la generalidad de las funciones y la seguridad del sistema. Como subcaracterísticas tiene asociada la adecuación, la cual se refiere al refinamiento de los grafos de secuencias.
- *Reusabilidad:* Es el grado en que un programa (o partes de este) se pueden reusar en otras aplicaciones. Este factor tiene una pregunta asociada: *¿Podré reusar alguna parte el software?* (McCall.J, 1977)

Según (Pressman, 1995), basándose en los estudios de (McCall, 1977), los factores que afectan a la calidad del software se centran en tres aspectos importantes de un producto software: sus características operativas, su capacidad de soportar los cambios (revisión del producto) y su adaptabilidad a nuevos entornos (o transición del producto).

1.4 Estrategias de prueba

En cierta manera la estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. Describe el enfoque y los objetivos generales de las actividades de prueba, define: niveles, tipos, técnicas (manual o automática) de prueba a utilizar en el proyecto. Además de, ¿qué criterios de éxito y culminación de la prueba serán usados?, y alguna que otras consideraciones especiales afectadas por requerimientos de recursos o que tengan implicaciones en la planificación. (Pressman, 1998.)

La planificación de una Estrategia de Prueba puede disminuir representativamente el esfuerzo preciso para el desarrollo de las pruebas proporcionadas, reducir el tiempo de elaboración y ejecución de las mismas, así como minimizar los altos costos que se forman. Es permitido aclarar que toda estrategia de pruebas debe llevar a cabo una planificación de las pruebas, diseño de casos de prueba, ejecución de las mismas, y por último, la agrupación y evaluación de los datos obtenidos.

La formulación de la estrategia de pruebas de software debe tener en cuenta los siguientes pasos:

- Identificar las distintas pruebas y las etapas en la que se aplicarán. Propuesta del instrumento de medición.
- El diseño y registro de casos de prueba.
- Aplicación de las pruebas propuestas.
- Documentación y evaluación de los resultados de la implementación de la estrategia propuesta.

Sus características generales son:

- Las pruebas comienzan a nivel de módulo y trabajan "hacia fuera".
- La prueba y la depuración son actividades diferentes, pero la depuración se debe incluir en cualquier estrategia de prueba.

- La prueba la lleva a cabo el responsable del desarrollo del software y un grupo independiente de pruebas.

  Según el momento, son apropiadas diferentes técnicas de prueba. (Granada)

Verificación y validación (V&V)

Verificación: se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica.

Validación: se refiere a un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente.

La estrategia de prueba del software se puede ver desde dos puntos de vista, el espiral y el procedimental.

Desde el punto de vista procedimental, la prueba consta de una serie de cuatro pasos secuenciales, estos son:

- *Prueba de unidad:* La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo. Además hace uso intensivo de las técnicas de prueba de caja blanca.
- *Prueba de integración:* La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la integración. Además en la misma prevalecen las técnicas de prueba de caja negra, aunque se pueden llevar a cabo, algunos de caja blanca con el fin de asegurar que se cubren los principales caminos de control.
- *Pruebas de alto nivel:* Se realizan después que se ha integrado (construido) el software.
- *Pruebas de validación:* La validación puede definirse de muchas formas, pero una simple definición es que la validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente, las cuales están definidas en la Especificación de Requisitos del Software. También proporciona una seguridad final de que el software satisface todos los requisitos funcionales, de comportamiento y de rendimiento, y se utilizan exclusivamente técnicas de prueba de caja negra.

Luego de destacar cuál es la importancia de trazar una estrategia de prueba para cada software, es necesario estudiar y profundizar en; qué son las pruebas y cuáles son los diferentes tipos de estas que se pueden aplicar en un proyecto determinado o específicamente en el proyecto que se está probando.

1.5 Pruebas de software

Las pruebas posibilitan medir la calidad del software en términos de los defectos encontrados, tanto en los requerimientos y características funcionales o no funcionales del software. Las mismas brindan confianza en la calidad del producto a realizar al encontrar defectos y a su vez su calidad se incrementa a medida que los defectos encontrados por las pruebas son reparados.

Una buena prueba según Kaner[1], Falk y Nguyen posee los siguientes atributos:

1. Tiene una alta probabilidad de encontrar un error.
2. No debe ser redundante.
3. Debería ser "la mejor de la cosecha".
4. No debería ser ni demasiado sencilla ni demasiado compleja.

Según la IEEE las pruebas constituyen "Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente". Pero muy particularmente Pressman plantea que "Las pruebas de software son un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación."

Las pruebas al software constituyen un proceso centrado en el objetivo de encontrar fallas en un software. Tratar de descubrir cada vez más errores, principalmente donde no han surgido hasta el momento, es el punto de partida de todo proceso de pruebas. Una prueba tiene éxito si descubre un error no detectado hasta entonces. (Weitzenfeld.,1989)

1 Cem Kaner J.D Profesor de Tecnología de dotación lógica en el Instituto de la Florida de la tecnología, y director del centro del Tech de la Florida para la educación y la investigación de prueba (CSTER) del software desde 2004. Trabajó en la industria del software que comenzaba en 1983 en Silicon Valley.

1.6 Tipos de prueba

Un aspecto primordial de esta parte es la definición de los casos de prueba. Pressman define los casos de pruebas como “un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo en particular.

Para este diseño de casos de prueba se identifican 3 enfoques de pruebas:

- El enfoque estructural o de caja blanca consiste en centrarse en la estructura interna (implementación) del programa para elegir los casos de prueba. Este enfoque se muestra en la figura 1.
- El enfoque funcional o de caja negra consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos. Este enfoque se muestra en la figura 2.
- El enfoque aleatorio, consistente en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

A continuación se describe con detalle los métodos de caja blanca y caja negra:

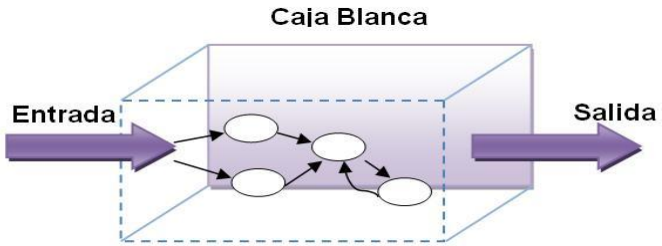


Figura 1. Prueba de caja blanca. (Figuras)

1. *Prueba de Caja Blanca:* denominada a veces prueba de caja de cristal es una técnica de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante las técnicas de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo, ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; ejecuten todos los ciclos en sus límites y con sus límites operacionales, y ejerciten las estructuras internas de datos para asegurar su validez. (Wiley, 1983)

Con este método se determina cuáles son los casos de prueba a partir del código fuente del software y se utilizan las especificaciones para determinar el resultado esperado del caso.
Mediante el mismo, el ingeniero de software puede obtener casos de prueba que:

- Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
- Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez. (Wiley, 1983)

Algunas técnicas empleadas en las pruebas de caja blanca son los siguientes:
Métodos de pruebas basados en caja blanca.

- *Prueba de Condición:* Método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. Esta prueba asegura que cada condición del programa no contenga errores.
- *Prueba de Flujo de Datos:* Selecciona los caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del mismo. Las estrategias de prueba de flujo de datos son útiles para seleccionar caminos de prueba de un programa que contenga sentencias if o bucles anidados.
- *Prueba de Bucles:* Técnica de prueba de caja blanca que se centra en la validez de las construcciones de bucles. Se pueden definir 4 clases diferentes de bucles estas son: Simples, Concatenados, Anidados y No estructurados.
- *Prueba del camino básico:* El método del camino básico, propuesto por Tom McCabe, permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (Pressman, 2005.)

A continuación se muestra en qué consiste el método de caja negra.

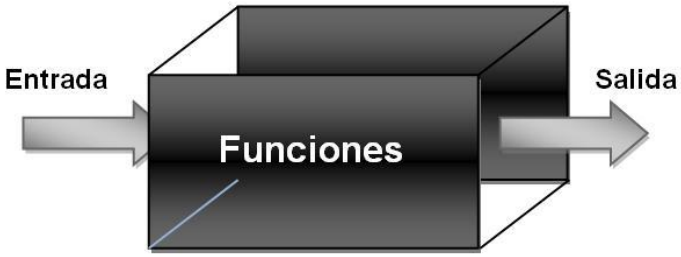


Figura 2. Prueba de caja negra. (Figuras)

2. *Prueba de Caja Negra:* Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro. Las mismas están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario. (Mañas, 1994)

Técnicas para desarrollar la prueba de caja negra.

- *Partición equivalente:* Es un método de prueba de caja negra, 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 objetivo de partición equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software. (Collazo, 2003.)
- *Análisis de Valores Límite (AVL):* Es una técnica de diseño de casos de prueba que complementa la prueba de Partición Equivalente. Pero en lugar de realizar la prueba con cualquier elemento de la partición equivalente, se escogen los valores en los extremos de la clase. (Collazo, 2003.)
- *Adivinando el error:* dado un programa particular, se conjetura, por la intuición y la experiencia, ciertos tipos probables de errores y entonces se escriben casos de prueba para exponer esos errores. Es difícil dar un procedimiento para esta técnica puesto que es en gran parte un proceso. (Collazo, 2003.)
- *Técnica de Grafos de Causa-Efecto:* Es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones. (Pressman, 2005.)

3. *Pruebas de Bajo Nivel*

- *Pruebas de unidad:* esta prueba se centra en la verificación de los elementos más pequeños del software que se puedan probar. Normalmente, se aplican a componentes representados en el modelo de implementación para verificar que se cubren los flujos de control y los flujos de datos y que funcionan como se esperaba. El implementador las realiza mientras se desarrolla la unidad. Los detalles de estas pruebas se describen en la disciplina de implementación. (TecnoRetales., 2009)
- *Pruebas de Integración:* Las mismas se llevan a cabo durante la construcción del sistema, involucran a un número creciente de módulos y terminan probando el sistema como conjunto. El objetivo de las mismas es tomar los módulos probados en unidad y estructurar un programa que esté de acuerdo con el que dicta el diseño. Estas pruebas se pueden plantear desde un punto de vista estructural o funcional. Las funcionales de integración son similares a las pruebas de caja negra y las estructurales de integración son similares a las pruebas de caja blanca; pero trabajan a un nivel conceptual superior. (Mañas, 1994)

Existen dos tipos de estrategias de integración incremental:

1. *Integración descendente (Top-Down):* es una estrategia de integración incremental en la cual se integran los módulos moviéndose en dirección hacia abajo por la jerarquía comenzando por el control principal (Programa principal). Los módulos subordinados de control principal se van incorporando a la estructura, bien, de forma *primero-en-profundidad* el cual integra todos los módulos del camino de control principal de la estructura, o bien *primero-en-anchura* que incorpora todos los módulos directamente subordinados a cada nivel. (Pressman, 2005)

Este proceso de integración consta de una serie de cinco pasos, estos son:

I) Se usa el módulo de control principal como controlador de la prueba, disponiendo de resguardos para todos los módulos directamente subordinados al módulo de control principal.

II) Dependiendo del enfoque de integración se van sustituyendo uno a uno los resguardos subordinados por los módulos reales.

III) Se llevan a cabo las pruebas cada vez que se integra un nuevo módulo real.

IV) Tras terminar cada conjunto de pruebas, se reemplaza otro resguardo con el módulo real.

V) Se hace la prueba de regresión para asegurarse de que no se han introducido errores nuevos.

A continuación se muestra en la figura 3 un ejemplo de dicha integración.

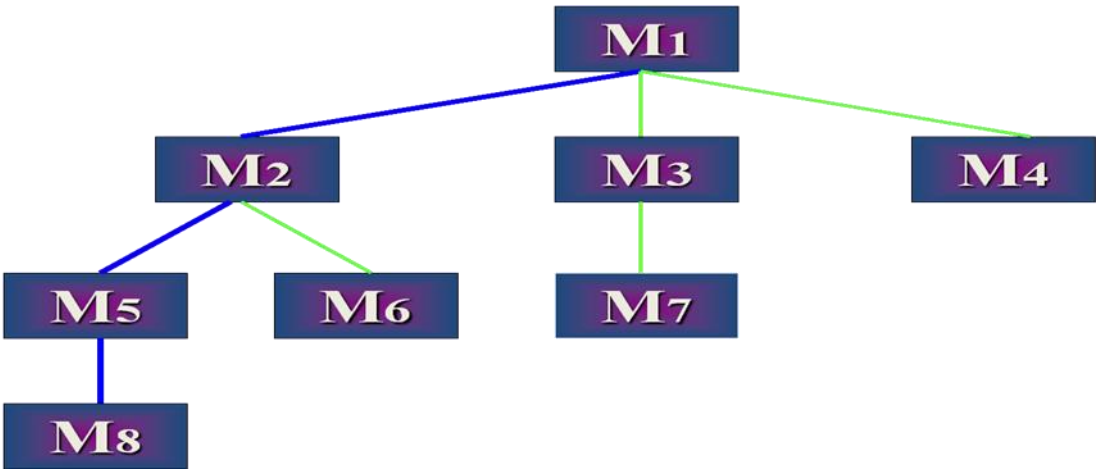


Figura 3. Integración descendente (Top-Down). (Figuras)

2. *Integración ascendente(Bottom-Up):* Comienza la construcción del diseño desde los módulos más bajos hacia arriba (módulo atómico). Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados a un nivel dado siempre van a estar disponibles. (Pressman, 2005.

Se puede implementar una estrategia de integración ascendente mediante los pasos siguientes:

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

Se debe tener en cuenta que, a medida que la integración progresa hacia arriba, disminuye la necesidad de controladores de prueba diferentes.

Una estrategia de prueba consta de pruebas de bajo nivel, así como también de pruebas de alto nivel.

4. Pruebas de Alto Nivel

➢ *Pruebas de validación:* la validación comienza tras la culminación de la prueba de integración. Se asume para esta parte que el software ha cumplido la etapa de verificación, por lo tanto está libre de errores de tiempo de ejecución, lo que no significa que esté libre de errores lógicos (diferencias entre la estrategia propuesta y la implementada). La misma se alcanza cuando el software funciona de tal manera que satisface las expectativas razonables del cliente. (Myers, 2004)

Existe un proceso para descubrir los errores que surgen bajo la operación de un usuario, el mismo es denominado pruebas de alfa y beta.

Prueba Alfa: esta prueba es realizada por un cliente en el lugar de desarrollo. De manera que es el usuario el que usa el software de forma natural, mientras que el encargado de desarrollo ó sea, el implementador solo observa y registra los errores y los problemas de uso. Estas pruebas se llevan a cabo en un entorno controlado.

Prueba Beta: la misma se lleva a cabo con los clientes en el sitio donde será enviado el software como producto final. A diferencia de la prueba alfa, en esta el implementador no está presente, es decir, que el software no puede ser controlado ya que el grupo de desarrollo no tiene acceso al entorno donde se desarrolla el mismo. Por lo que el cliente tiene que registrar todos los problemas (reales o imaginarios) que se encuentren e informarlos a intervalos regulares a los desarrolladores.

➢ *Prueba del sistema:* tienen el propósito de comparar el sistema o el programa con sus objetivos originales (Requerimientos funcionales y no funcionales).Las mismas no se limitan a los sistemas y por definición, son imposibles de realizar sino están los requerimientos por escrito. (Myers, 2004)

Algunas de estas pruebas son:

Prueba de recuperación: es una prueba del sistema que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente. (Pressman, 2002)

Prueba de seguridad: consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos. (Pressman, 2002)

Prueba de rendimiento: determinan los tiempos de respuesta que están dentro de los intervalos establecidos en las especificaciones del sistema. (Pressman, 2002)

Prueba de instalación: son una parte importante del proceso de prueba del sistema. Si la fase de instalación se realiza mal, entonces el usuario/el cliente puede buscar otro producto o tener poca confianza en la validez de la aplicación. (Myers, 2004)

Prueba de resistencia: determina hasta donde puede soportar el programa determinadas condiciones extremas. (Pressman, 2002)

1.7 Otros tipos de pruebas:

Recorridos (walkthroughs): Quizás es una técnica más aplicada en control de calidad que en pruebas. Consiste en sentar alrededor de una mesa a los desarrolladores y a una serie de críticos, bajo las órdenes de un moderador que impida un recalentamiento de los ánimos. El método consiste en que los revisores se leen el programa línea a línea y piden explicaciones de todo lo que no está meridianamente claro. Esta técnica es muy eficaz localizando errores de naturaleza local; pero falla estrepitosamente cuando el error deriva de la interacción entre dos partes alejadas del programa. Nótese que no se está ejecutando el programa, sólo mirándolo con lupa, y de esta forma sólo se ve en cada instante trocito del listado.

Aleatorias (random testing): Ciertos autores consideran injustificada una aproximación sistemática a las pruebas. Alegan que la probabilidad de descubrir un error es prácticamente la misma si se hacen una serie de pruebas aleatoriamente elegidas, que si se hacen siguiendo las instrucciones dictadas por criterios de cobertura (caja negra o blanca). Como esto es muy cierto, probablemente sea muy razonable comenzar la fase de pruebas con una serie de casos elegidos al azar. Esto pondrá de manifiesto los errores más patentes. No obstante, pueden permanecer ocultos errores más furtivos que sólo se muestran ante entradas muy precisas. Si el programa es poco crítico (una aplicación personal, un juego,) puede que esto sea suficiente. Pero si se trata de una aplicación militar o con riesgo para vidas humanas, es de todo punto insuficiente.

Solidez (robustness testing): Se prueba la capacidad del sistema para salir de situaciones embarazosas provocadas por errores en el suministro de datos. Estas pruebas son importantes en sistemas con una

interfaz al exterior, en particular cuando la interfaz es humana. Por ejemplo, en un sistema que admite una serie de órdenes (commands) se deben probar los siguientes extremos:

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

Aguante (stress testing): En ciertos sistemas es conveniente saber hasta dónde aguantan, bien por razones internas (¿hasta cuantos datos podrá procesar?), bien externas (¿es capaz de trabajar con un disco al 90%?, ¿aguanta una carga de la CPU del 90?, etc.).

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

Conformidad u Homologación (conformance testing): En programas de comunicaciones es muy frecuente que, además de los requisitos específicos del programa que estamos construyendo, aparezca alguna norma más amplia a la que el programa deba atenerse. Es frecuente que organismos internacionales como ISO y el CCITT elaboren especificaciones de referencia a las que los diversos fabricantes deben atenerse para que sus ordenadores sean capaces de entenderse entre sí. Las pruebas, de caja negra, que se le pasan a un producto para detectar discrepancias respecto a una norma de las descritas en el párrafo anterior se denominan de conformidad u homologación. Suelen realizarse en un centro especialmente acreditado al efecto y, si se pasan satisfactoriamente, el producto recibe un sello oficial que dice: "homologado".

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

Regresión (regression testing): Todos los sistemas sufren una evolución a lo largo de su vida activa. En cada nueva versión se supone que o bien se corrigen defectos, o se añaden nuevas funciones, o ambas cosas. En cualquier caso, una nueva versión exige una nueva pasada por las pruebas. Si éstas se han sistematizado en una fase anterior, ahora pueden volver a pasarse automáticamente, simplemente para comprobar que las modificaciones no provocan errores donde antes no los había. El mínimo necesario para usar unas pruebas en una futura revisión del programa es una documentación muy clara. Las pruebas de regresión son particularmente espectaculares cuando se trata de probar la interacción con un agente externo. Existen empresas que viven de comercializar productos que "graban" la ejecución de una prueba con operadores humanos para luego repetirla cuantas veces haga falta "reproduciendo la grabación". Y, obviamente, deben monitorizar la respuesta del sistema en ambos casos, compararla, y avisar de cualquier discrepancia significativa.

Mutación (mutation testing): Es una técnica curiosa consistente en alterar ligeramente el sistema bajo pruebas (introduciendo errores) para averiguar si la batería de pruebas es capaz de detectarlo. Sino, más vale introducir nuevas pruebas. Todo esto es muy laborioso y francamente artesano. (Mañas, 1994)

1.8 Pruebas a aplicar

- Pruebas de Unidad
  - ✓ De Caja blanca
    - Camino básico
- Pruebas de aceptación(validación)
  - ✓ De Caja negra
    - Partición equivalente

- Pruebas de integración
  - ✓ Incremental ascendente
- Pruebas del sistema
  - ✓ Prueba de Volumen

Conclusiones parciales del capítulo.

Es importante destacar que la calidad del software está estrechamente relacionada con las pruebas de software, ya que es en esta actividad donde se detectan los errores y las inconsistencias que pueda tener el mismo.

El análisis realizado en este capítulo ha demostrado de manera clara, la importancia que tiene la calidad del software, a lo que contribuyó el abarcamiento de los temas tratados.

CAPÍTULO 2: Plan de pruebas.

2 Introducción

En el presente capítulo se precisará un plan de prueba como base para todo el proceso que se llevará a cabo en el mismo, así como la configuración del entorno de pruebas, especificando cada uno de los elementos de las pruebas definidos. Además quedará plasmado que la construcción de un plan de prueba será la piedra angular y en efecto uno de los principales factores críticos de un proceso de prueba que permitirá entregar un software de mejor nivel.

2.1 Pruebas a aplicar al proyecto Video Vigilancia

Inicialmente las pruebas que se le aplicarán al producto son las de Función. Estas pruebas se centran en la validación de funciones del destino de la prueba. Los niveles al cual se llevará a cabo son unidad e integración, asegurando que funcionan adecuadamente.

2.1.1 Pruebas de Unidad

De Caja blanca

Es aquí donde se realizarán los diseños de casos de prueba de caja blanca a los casos de usos (CU) seleccionados, utilizando la técnica del camino básico. La misma se le aplicará sólo a los CU más críticos con los que cuenta el proyecto Video Vigilancia, pues esta práctica de prueba requiere mucho tiempo para su ejecución. Si bien es cierto que entre más detalladas sean las pruebas que se le apliquen al producto, mayor será el grado de calidad que tendrá el mismo, hay que tener en cuenta el esfuerzo que requiere cada una de estas pruebas. Con esta técnica, por cada caso de prueba es necesaria la construcción de un grafo que represente todas las variantes que tiene el algoritmo implementado para darle solución al caso de uso.

Camino básico: Es un método que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizarán que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (Pressman, 1998).

Los pasos del diseño de pruebas a seguir mediante el camino básico son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.

2. Se calcula la complejidad ciclomática del grafo de flujo.
3. Se determina el conjunto básico de caminos independientes.
4. 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 del Grafo de flujo: Este tipo de método utiliza para su ejecución la construcción de un: ***Grafo de flujo o grafo del programa:*** este representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de éste. (Cada nodo representa una o más sentencias procedimentales y cada arista representa el flujo de control). (Pressman, 1998.)

Para construir el grafo se debe tener en cuenta la notación para cada una de las instrucciones correspondientes como se muestra a en la figura 4 a continuación.

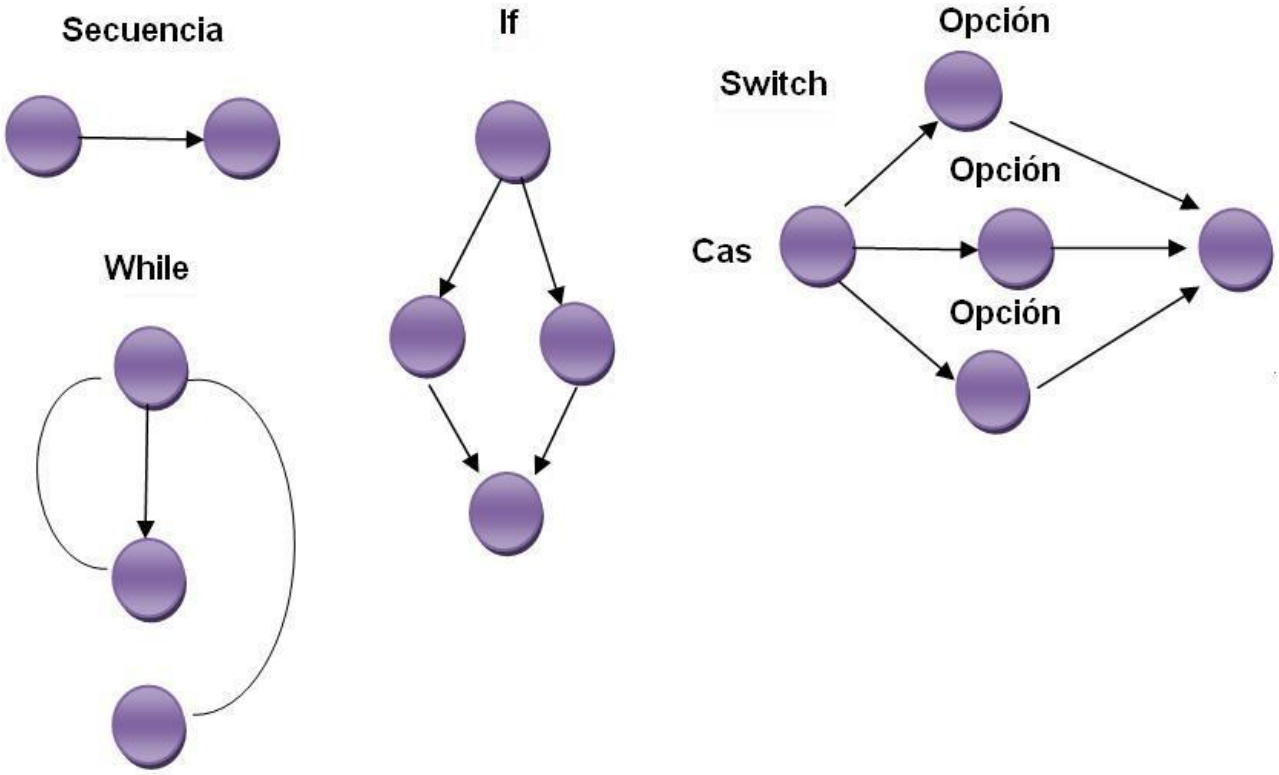


Figura 4. Notación de grafos de flujo para las instrucciones: Secuenciales, If, While, Switch. (Figuras)

Un ejemplo de representación de Grafo de Flujo es el mostrado en la Figura 5 en el cual aparecen sus elementos fundamentales:

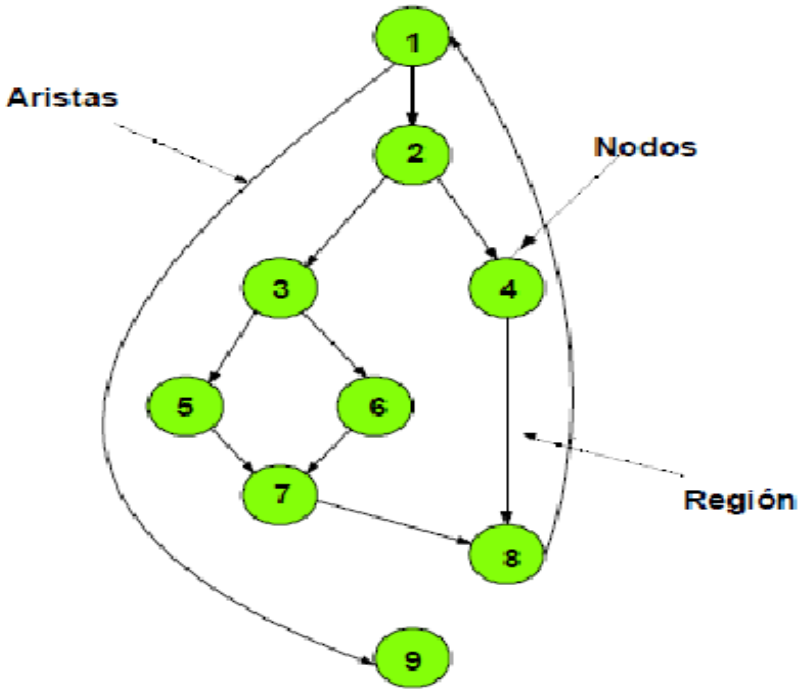


Figura 5. Elementos de los grafos. (Figuras)

Cuando en una condición aparecen uno o más operadores lógicos (AND, OR, XOR) se crea un nodo distinto por cada una de las condiciones simples por lo que, la generación del grafo de flujo se hace un poco más complicada. Cada nodo generado de esta forma se denomina nodo predicado.

El grafo de flujo se utiliza para representar flujo de control lógico de un programa. Para ello se utilizan los tres elementos siguientes:

- *Nodos*: representan cero, una o varias sentencias en secuencia. Cada nodo comprende como máximo una sentencia de decisión (bifurcación).
- *Aristas*: líneas que unen dos nodos.
- *Regiones*: áreas delimitadas por aristas y nodos. Cuando se contabilizan las regiones de un programa debe incluirse el área externa como una región más.

Complejidad Ciclomática: Para contar el número de ciclos diferentes que se siguen en un fragmento de código de un programa, habiendo creado una rama imaginaria desde el nodo de salida al nodo de entrada se utiliza la Complejidad Ciclomática (Cyclomatic Complexity), esta es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software más ampliamente aceptada, ya que ha sido concebida para ser independiente del lenguaje. (Pressman, 1998.)

Esta métrica, propuesta por Thomas McCabe[2], se basa en el diagrama de flujo determinado por las estructuras de control de un determinado código. De dicho análisis se puede obtener una medida cuantitativa de la dificultad de crear pruebas automáticas del código y también es una medición orientativa de la fiabilidad del mismo.

El resultado obtenido en el cálculo de la complejidad Ciclomática define el número de caminos independientes dentro de un fragmento de código y determina la cota superior del número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. (Pressman, 1998.)

La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingieniería para estimar el riesgo, costo y estabilidad. (Mora., 2007)

Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo de flujo:
1ra vía V (G)= Número de Regiones.
2da vía V (G)= Número de Aristas – Número de Nodos + 2
3ra vía V (G)= Número de Nodos Predicados +1

La Figura 6 representa, por ejemplo, las cuatro regiones del grafo de flujo, obteniéndose así la complejidad ciclomática de 4. Análogamente se puede calcular el número de aristas y nodos predicados para confirmar la complejidad ciclomática. Así:

V (G) = Número de regiones = 4
V (G) = Aristas – Nodos + 2 = 11-9 + 2 = 4
V (G) = Nodos Predicado + 1 = 3 +1 = 4

[2] Propone la métrica de la complejidad Ciclomática en 1976

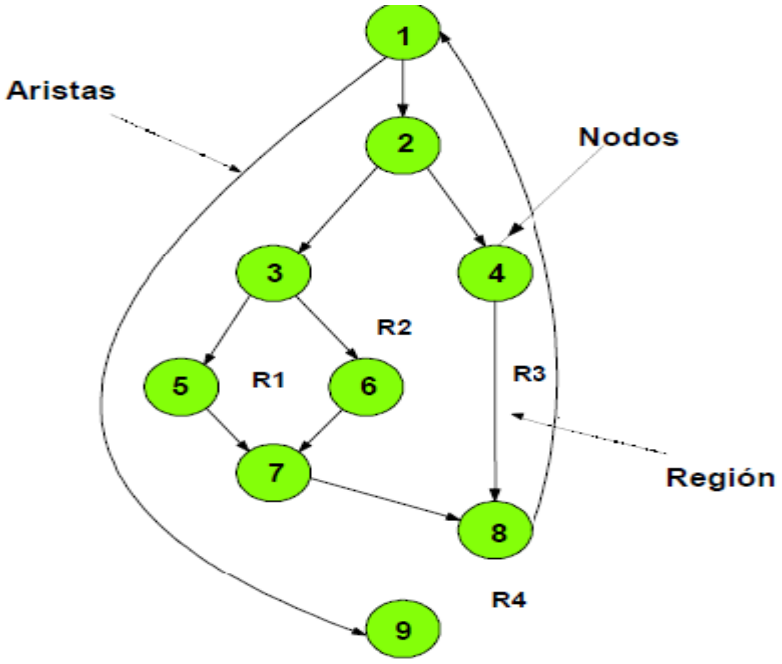


Figura 6. Número de regiones del grafo de flujo. (Figuras)

Camino independiente: cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición. (Mora., 2007)

El conjunto de caminos independientes de un grafo no es único. No obstante, a continuación, se muestran algunas heurísticas (término con el que se refiere al método o procedimiento usado en la investigación o en el descubrimiento de algo) para identificar dichos caminos:

(a) Elegir un camino principal que represente una función válida que no sea un tratamiento de error. Debe intentar elegirse el camino que atraviese el máximo número de decisiones en el grafo.

(b) Identificar el segundo camino mediante la localización de la primera decisión en el camino de la línea básica alternando su resultado mientras se mantiene el máximo número de decisiones originales del camino inicial.

(c) Identificar un tercer camino, colocando la primera decisión en su valor original a la vez que se altera la segunda decisión del camino básico, mientras se intenta mantener el resto de decisiones originales.

(d) Continuar el proceso hasta haber conseguido tratar todas las decisiones, intentando mantener como en su origen el resto de ellas.

Este método permite obtener V (G) caminos independientes cubriendo el criterio de cobertura de decisión y sentencia.

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.

Se selecciona el método de camino básico como prueba de caja blanca, ya que este permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizarán que durante la prueba se ejecute por lo menos una vez cada sentencia del programa, a diferencia de la prueba de bucles que se centra exclusivamente en la validez de las construcciones de bucles (simples, anidados, concatenados y no estructurados). Es decir, que la prueba más completa y la que mejor resultados brinda es la de camino básico.

Se podría pensar que con la aplicación de pruebas de caja blanca se lograrían tener programas libres de errores y defectos, lo único necesario para cumplir lo antes planteado sería probar todos los caminos lógicos del programa exhaustivamente. El único problema es que se estaría violando uno de los principios de las pruebas que es “La prueba exhaustiva es imposible para los grandes sistemas.” (Pressman, 1998.)

Queda demostrado que no se pueden realizar las pruebas de caja blanca exhaustivamente pero esto no quiere decir que no se realicen. Se puede elegir y ejercitar una cantidad de caminos lógicos importantes. Se puede comprobar las estructuras de datos más importantes para comprobar su validez y combinándose con los métodos de caja negra los resultados serán satisfactorios.

2.1.2 Pruebas de Aceptación (validación)

De caja negra

Las pruebas de caja negra se centran fundamentalmente en los requisitos funcionales del software. Es decir, la prueba de caja negra permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Se trata de un enfoque que intenta descubrir diferentes tipos de errores que no se encuentran con los métodos de caja blanca. Esta

técnica de prueba es realizada a nivel de sistema es decir no se tiene ninguna relación con el código del producto.

Las pruebas de caja negra, intentan encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Partición equivalente: Una partición equivalente es una técnica de prueba de Caja Negra que divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba para la partición equivalente se basa en la evaluación de las clases de equivalencia. Esto quiere decir que si el caso de prueba correspondiente a una clase de equivalencia detecta un error, el resto de los casos de prueba de dicha clase de equivalencia deben detectar el mismo error. Y viceversa, si un caso de prueba no ha detectado ningún error, es de esperar que ninguno de los casos de prueba correspondientes a la misma clase de equivalencia encuentre ningún error.

El diseño de casos de prueba según esta técnica consta de dos pasos:

1ro. Identificar las clases de equivalencia.

2do. Identificar los casos de prueba.

Identificación de las clases de equivalencia.

Se identifican clases de equivalencia válidas e inválidas con la siguiente tabla

Condición externa	Clase de equivalencia válidas(CEV)	Clase de equivalencia inválidas(CEI)
Condición entrada	Clases válidas para esa condición de entrada	Clases inválidas para esa condición de entrada

Tabla 1. Características de los grafos

A continuación se siguen estas directrices:

- Si una condición de entrada especifica un rango de valores (p.e, entre 1 y 999), se define una CEV (1 <= valor <= 999) y dos CEI (valor < 1 y valor > 999).

- Si una CE requiere un valor específico (p.e., el primer carácter tiene que ser una letra), se define una CEV (una letra) y una CEI (no es una letra).
- Si una CE especifica un conjunto de valores de entrada, se define una CEV para cada uno de los valores válidos, y una CEI (p.e., CEV para "Moto", Coche" y "Camión", y CEI para "Bicicleta").
- Si una condición de entrada especifica el número de valores (p.e., una casa puede tener uno o dos propietarios), identificar una CEV y dos CEI (0 propietarios y 3 propietarios).

Identificación de casos de prueba.

Para la identificación de casos de prueba se deben seguir varios pasos, estos son:

- Asignar un número único a cada clase de equivalencia.
- Escribir casos de prueba hasta que sean cubiertas todas las CEV, intentando cubrir en cada caso tantas CEV como sea posible.
- Para cada CEI, escribir un caso de prueba, cubriendo en cada caso una CEI.

Con la aplicación de esa técnica se obtiene un conjunto de pruebas que reduce el número de casos de pruebas y dice 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 posibles errores.

Se escoge como técnica de caja negra a aplicar la de partición de equivalencia ya que esta divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

Luego de haber realizado todas las pruebas de Función, se deben ensamblar o integrar los módulos para formar el paquete de software completo. Aquí es donde se aplicarán las pruebas de Integridad, en este caso específicamente la integración ascendente, dado que los módulos se integran de abajo hacia arriba. Durante la integración, las técnicas que más prevalecen son las de diseño de caso de prueba de caja negra utilizando la técnica de partición equivalente, probando cada una de las entradas con las salidas esperadas y el nivel de prueba que se lleva a cabo es el de Integración.

2.1.3 Pruebas de Integración.

Integración incremental ascendente: es donde la construcción del diseño empieza desde los módulos más bajos hacia arriba (módulo principal), el procesamiento requerido de los módulos subordinados siempre está disponible y elimina la necesidad de resguardo. La sección de una estrategia de integración depende de las características del software y, a veces, del plan del proyecto, en algunos de los casos se puede combinar ambas estrategias.

Ventajas de la integración incremental ascendente:

- Las entradas para las pruebas son más fáciles de crear ya que los módulos inferiores suelen tener funciones más específicas.
- Es más fácil la observación de los resultados de las pruebas puesto que es en los módulos inferiores donde se elaboran.
- Resuelve primero los errores de los módulos inferiores que son los que acostumbran a tener el procesamiento más complejo, para luego nutrir de datos al resto del sistema.

Para la generación de casos de prueba de integración, ya sea descendente o ascendente se utilizan técnicas de caja negra.

Después de que el software se ha integrado se deben aplicar las pruebas de Función a nivel de sistema para verificar que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento total del sistema, utilizando pruebas de caja negra empleando la técnica de partición equivalente.

2.1.4 Prueba de Sistema.

Prueba de Volumen: Consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas. Estas pruebas ejercitan la aplicación con volúmenes muy altos o muy bajos de entrada de datos para determinar o probar la resistencia de la misma.

El objetivo principal de esta prueba es someter al sistema a grandes volúmenes de datos para determinar si el mismo puede manejar el volumen de datos especificado en sus requisitos.

Al realizar esta prueba se debe comprobar:

✓ La conformidad (sub característica de la ISO 9126): Capacidad del software para adherirse a las normas que se le apliquen, convenciones, regulaciones, leyes y las prescripciones similares.

✓ La madurez (sub característica de la ISO 9126): Capacidad del software de evitar una avería como resultado de haberse producido un fallo del software. Confiabilidad (Característica de la ISO 9126).

✓ La tolerancia ante fallos (sub característica de la ISO 9126): Capacidad del software de mantener un nivel de ejecución específico en caso de fallos del software o de infracción de sus interfaces especificadas. Confiabilidad (Característica de la ISO 9126).

✓ La utilización de recursos (sub característica de la ISO 9126): Capacidad del software para usar los recursos apropiados en un plazo de tiempo adecuado cuando el software realiza su función bajo las condiciones declaradas. Eficiencia (Característica de la ISO 9126).

Para la realización de esta prueba se hace uso del principio de Pareto el cual plantea que el 20% producirá el 80% de los efectos, mientras que el 80% restante sólo cuenta para el 20% de los efectos. (Pareto)
Dicha prueba se llevará a cabo con la herramienta EMS Data Generator para PostgreSQL.
.

2.2 Procedimientos para cada una de las pruebas seleccionada

2.2.1 Pruebas de Unidad

De Caja blanca

Camino básico

Procedimiento:
1. Identificar dentro del código a revisar los nodos.
2. Elaborar el grafo de flujo.
3. Calcular la complejidad ciclomática del grafo.
4. Determinar un conjunto básico de caminos independientes.
5. Preparar los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico

2.2.2 Pruebas de Aceptación (validación)

De caja negra

Partición equivalente

Procedimiento:

1. Se identifican clases de equivalencia válida (CEV) e inválida (CEI).
2. Asignar un número único a cada clase de equivalencia.
3. Escribir casos de prueba hasta que sean cubiertas todas las CEV, intentando cubrir en cada caso tantas CEV como sea posible.
4. Para cada CEI, escribir un caso de prueba, cubriendo en cada caso una CEI.

2.2.3 Pruebas de Integración

Incremental ascendente

Procedimiento:

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

2.2.4. Pruebas de sistema

Prueba de volumen

Procedimiento:

1. Usar múltiples clientes, corriendo las mismas pruebas o pruebas complementarias para producir el peor caso de volumen por un período extendido.
2. Utilizar un tamaño máximo de Base de datos (actual o con datos representativos) y múltiples clientes para correr consultas simultáneamente para períodos extendidos.

Conclusiones parciales del capítulo.

Una vez estudiado los diferentes tipos de prueba así como sus técnicas y procedimientos se pudo definir la estructura de los casos de prueba para cada uno de los tipos de prueba escogidos, lo que permite que se tenga una organización de cómo aplicar cada una de ellas. Además, quedó definida

la importancia de tener un plan de prueba correctamente estructurado y que en su confección abarcara las pruebas de alto nivel así como las de bajo nivel.

CAPÍTULO 3: Diseño, ejecución y evaluación de las pruebas.

3. Introducción

Este capítulo contiene todo lo referente al diseño de los casos de prueba ya elaborados para cada caso de uso y para cada tipo de prueba a utilizar según los niveles antes definidos, además del cronograma para la aplicación de cada una de estas pruebas. También se evalúan los resultados obtenidos de la ejecución de los casos de pruebas y se realiza un resumen de los mismos.

3.1 Diseño de los casos de pruebas.

3.1.1 Pruebas de Unidad

De Caja blanca

Camino básico

Nombre CU: Gestionar cámaras
Para el Camino 1: 1-2-3-4-5-7
Caso de Prueba: Probando la función AddCamera
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan adicionar las cámaras.

Nombre CU: Gestionar cámaras
Para el Camino 2: 1-2-3-9
Caso de Prueba: Probando la función AddCamera
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan adicionar las cámaras.

Nombre CU: Gestionar cámaras
Para el Camino 1: 1-2-3-4-7

Caso de Prueba: Probando la función DeleteCamera

Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan eliminar las cámaras.

Nombre CU: Gestionar cámaras
Para el Camino 1: 1-2-3-4-5-7
Caso de Prueba: Probando la función UpdateCamera
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan actualizar las cámaras.

Nombre CU: Gestionar cámaras
Para el Camino 2: 1-2-3-9
Caso de Prueba: Probando la función UpdateCamera
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan actualizar las cámaras.

Nombre CU: Gestionar usuarios
Para el Camino 1: 1-2-3-4
Caso de Prueba: Probando la función UpdateUser
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.

Resultado: Comprobar que todas las condiciones se cumplan para que se puedan actualizar los usuarios.

Nombre CU: Gestionar usuarios

Para el Camino 2: 1-2-3-5

Caso de Prueba: Probando la función UpdateUser

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado: Comprobar que todas las condiciones se cumplan para que se puedan actualizar los usuarios.

Nombre CU: Gestionar usuarios

Para el Camino 1: 1-2-3-8

Caso de Prueba: Probando la función RemoveUser

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado: Comprobar que todas las condiciones se cumplan para que se puedan eliminar los usuarios.

Nombre CU: Gestionar usuarios

Para el Camino 2: 1-9

Caso de Prueba: Probando la función RemoveUser

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.

Resultado: Comprobar que todas las condiciones se cumplan para que se puedan eliminar los usuarios.

Nombre CU: Gestionar usuarios

Para el Camino 3: 1-2-3-4-5-7

Caso de Prueba: Probando la función RemoveUser

Entrada:

Valor de rol: por definir.

Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan eliminar los usuarios.

Nombre CU: Gestionar roles
Para el Camino 1: 1-2-3-4
Caso de Prueba: Probando la función UpdateRol
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan actualizar los roles.

Nombre CU: Gestionar roles
Para el Camino 1: 1-2-3-5
Caso de Prueba: Probando la función AddRol
Entrada:
Valor de rol: por definir.
Valor de JSONpermisos: por definir.
Resultado: Comprobar que todas las condiciones se cumplan para que se puedan adicionar los roles.

3.1.2 Pruebas de Aceptación (validación)

De caja negra

Partición equivalente

Nombre del Caso de Uso: Enlazar eventos.
Descripción General: Este CU se inicia cuando el actor Cliente General, desea enlazarse a eventos que ocurren en otros módulos y termina cuando el sistema registra al cliente para los eventos seleccionados.
Condiciones de Ejecución: El cliente está registrado.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC1: Enlazar eventos.	EC 1.1: Enlazar eventos exitosamente.	El cliente solicita enlazarse a un evento determinado en el gestor. El sistema chequea el identificador del cliente y los permisos que tiene.	Ejecutar el Módulo Visor/ Volver a ejecutar el Módulo visor/hacer cambios en alguna de las cámaras en uno de los visores/ el otro visor debe reflejar dichos cambios.
	EC 1.2: Enlazar eventos fallido.	En caso de ser un ID inválido (este ID lo asigna el Gestor a un módulo, mediante un código Hash, para el usuario es trasparente), o que el cliente no tenga los permisos se levanta una excepción.	Ejecutar el Módulo Visor/ Volver a ejecutar el Módulo visor/ hacer cambios en alguna de las cámaras en uno de los visores/ el otro visor debe reflejar dichos cambios.

Tabla 2. Enlazar eventos

Nombre del Caso de Uso: Gestionar calendario de grabación.

Descripción General: Este CU se inicia cuando el actor Grabador desea obtener, adicionar, actualizar y eliminar las reglas de grabación y termina cuando se añade la regla al planificador.

Condiciones de Ejecución: El cliente está registrado.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Adicionar regla de grabación	EC 1.1: Adicionar regla de grabación exitosamente.	El cliente solicita adicionar una regla de grabación. El sistema chequea el ID (este ID lo asigna el Gestor a un módulo, mediante un código Hash, no hay manera de probarlo, se cumple para los demás ID) del cliente. Se almacenan los datos de la regla en la Base de Datos.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic derecho en una cámara/ opción “Adicionar Regla”.
	EC 1.2: Adicionar regla de grabación fallido.	En caso de ser un ID inválido se levanta una excepción.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic derecho en una cámara/ opción “Adicionar Regla”. También puede adicionar una regla siguiendo el siguiente camino: Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder /Clic sobre una de las cámaras/ en la ficha Calendario/Clic en el botón adicionar regla de grabación.

SC 2: Actualizar regla de grabación.	E.C 2.1: Actualizar regla de grabación exitosamente.	El cliente solicita actualizar una regla de grabación. El sistema chequea el ID del cliente. El sistema actualiza los datos de la regla en la Base de Datos.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic en una cámara/ opción / en Panel "Listado de Reglas"/ seleccionar la regla/ botón "Editar Regla" en barra de herramientas.
	EC 2.2: Actualizar regla de grabación fallido.	En caso de ser un ID inválido se levanta una excepción.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic en una cámara/ opción / en Panel "Listado de Reglas"/ seleccionar la regla/ botón "Editar Regla" en barra de
SC 3: Eliminar regla de grabación.	E.C 3.1: Eliminar regla de grabación exitosamente.	El cliente solicita eliminar una regla de grabación. El sistema chequea el ID del cliente. El sistema elimina los datos de la regla en la Base de Datos.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic en una cámara/ opción / en Panel "Listado de Reglas"/ seleccionar la regla/ botón "Eliminar Regla" en barra de herramientas.
	E.C 3.2: Eliminar regla de grabación fallido.	En caso de ser un ID inválido se levanta una excepción.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic en una cámara/ opción / en Panel "Listado de Reglas"/ seleccionar la regla/ botón "Eliminar Regla" en barra de herramientas.

SC 4: Obtener regla de grabación.	E.C 4.1: Obtener regla de grabación exitosamente.	El cliente solicita obtener una regla de grabación. El sistema chequea el ID del cliente. El sistema obtiene los datos de la regla en la Base de Datos.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic en una cámara/ se visualiza en el panel derecho.
	E.C 4.2: Obtener regla de grabación fallido.	En caso de ser un ID inválido se levanta una excepción.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / Clic en una cámara/ se visualiza en el panel derecho.

Tabla 3. Gestionar calendario de grabación

Nombre del Caso de Uso: Gestionar Roles

Descripción General: Este caso de uso se inicia cuando el Administrador desde la opción Gestionar roles, puede crear, modificar o eliminar roles en el sistema. El caso de uso termina con la creación o actualización de un rol, la eliminación de un rol o la cancelación de cualquier operación de las anteriores.

Condiciones de Ejecución: Que el administrador se haya autenticado.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC:1 Adicionar rol	EC 1.1: Adicionar rol exitosamente.	Añadir rol introduciendo el nombre del nuevo rol y pulsando el botón “Aceptar”. El sistema adiciona el nuevo rol.	Módulo Visor / Administrar Usuario/Menú/Adicionar/Rol/ Aceptar.

	EC 1.2: Adicionar rol. Falla.	Si no se introdujo ningún valor en el campo nombre del rol el sistema muestra un mensaje de error indicando que verifique los datos.	Módulo Visor / Administrar Usuario/Menú/Adicionar/Rol/ Aceptar
	EC 1.3: Adicionar rol. Rol existente en el sistema.	Si existe un rol con el nombre introducido el sistema muestra un mensaje indicando que verifique los datos.	Módulo Visor / Administrar Usuario/Menú/Adicionar/Rol/ Aceptar
SC:2 Actualizar rol.	EC 2.1: Actualizar rol exitosamente.	Editar rol seleccionando la opción de actualizar rol, el sistema guarda los cambios y regresa al formulario de gestión de roles.	Módulo Visor / Administrar Usuario/Menú/Actualizar/Rol /Aceptar
	EC 2.2: Actualizar Cancelar	El Administrador selecciona la opción “Cancelar”, el sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor / Administrar Usuario/Menú/Actualizar/Rol /Cancelar
SC 3: Eliminar rol	EC 3.1: Eliminar rol exitosamente.	Eliminar rol, seleccionando la opción “Eliminar rol”, el sistema elimina el rol seleccionado.	Módulo Visor / Administrar Usuario/Menú/Eliminar/Rol/Aceptar.
	EC 3.2 Eliminar rol. Cancelar.	El Administrador selecciona la opción “Cancelar”, el sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor / Administrar Usuario/Menú/Eliminar/Rol/ Cancelar.
SC4: Actualizar permisos de rol	EC 4.1: Actualizar permisos de rol exitosamente.	Actualizar permisos de rol, el Administrador selecciona o deselecciona los permisos que desee asignarle al rol seleccionado y presiona el botón “Aceptar”. El sistema guarda los cambios realizados.	Módulo Visor / Administrar Usuario/Menú/Actualizar/Rol /Aceptar.

	EC 4.2: Actualizar permisos de rol existe. Cancelar	El Administrador selecciona la opción “Cancelar”, el sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor / Administrar Usuario/Menú/Actualizar/Rol /Cancelar.

Tabla 4. Gestionar Roles

Nombre del Caso de Uso: Gestionar usuarios

Descripción General: Este caso se inicia cuando el Administrador desde la opción de Gestionar usuarios, podrá crear, modificar o eliminar usuarios. El caso de uso termina con la creación o actualización de usuarios, la eliminación de un usuario o la cancelación de cualquier operación de las anteriores.

Condiciones de Ejecución: Que el administrador se haya autenticado.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1 Adicionar usuario.	EC 1.1: Adicionar usuario exitosamente.	El sistema muestra un formulario para entrar los datos del usuario (“Usuario”, “contraseña”, “confirmar contraseña”, y selecciona el “rol” y selecciona la opción Aceptar. El sistema almacena los datos del usuario.	El flujo central debería ser: Módulo Visor/ Clic en ficha “Configuración”/ botón “Administrar Usuarios”/ Menú/Adicionar/Usuario /Aceptar.
	EC 1.2: Adicionar usuario. Falla.	En caso de entrar los datos incorrectos, el sistema muestra un mensaje indicando que los datos de entrada no son válidos.	Módulo Visor/ Clic en Administrar usuario/ Menú/Adicionar/Usuario /Aceptar.

	EC 1.3: Adicionar usuario. (Usuario registrado en el sistema).	En el caso de que al entrar los datos, el usuario exista en la base de datos se muestra un mensaje indicando que el usuario ya existe en el sistema.	Módulo Visor/ Clic en Administrar usuario/ Menú/Adicionar/Usuario /Aceptar.
	EC 1.4: Adicionar usuario. Cancelar.	El sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor/ Clic en Administrar usuario/ Menú/Adicionar/Usuario /Cancelar.
SC 2: Actualizar datos de usuario.	EC 2.1: Actualizar usuario exitosamente.	El administrador al seleccionar un usuario, el sistema muestra un formulario con la información del usuario que desea modificar "nuevo nombre", "contraseña", "confirmar contraseña" y "rol". El administrador modifica datos del usuario y selecciona la opción "Aceptar".	Módulo Visor/ Clic en Administrar usuario/ Menú/Actualizar/Usuario /Aceptar.
	EC 2.2: Actualizar usuario falla.	El sistema comprueba los datos y al encontrar errores muestra un mensaje indicando que los datos entrados son incorrectos	Módulo Visor/ Clic en Administrar usuario/ Menú/Actualizar/Usuario /Aceptar.
	EC 2.2: Actualizar usuario. Cancelar	El sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor/ Clic en Administrar usuario/ Menú/Actualizar/Usuario /Cancelar.

SC 3: Eliminar usuario	EC 3.1: Eliminar usuario exitosamente.	. El sistema elimina el usuario o los usuarios seleccionado de la Base de Datos.	Módulo Visor/ Clic en Administrar usuario/ Menú/Eliminar/Usuario /Borrar.
	EC 3.2 Eliminar usuario. Cancelar.	El Administrador selecciona la opción "Cerrar", el sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor/ Clic en Administrar usuario/ Menú/Eliminar/Usuario /Aceptar.

Tabla 5. Gestionar usuarios

Nombre del Caso de Uso: Gestionar cámaras.

Descripción General: El caso de uso se inicia cuando el Administrador selecciona la opción de adicionar, eliminar o editar una cámara para realizar dichas acciones y termina cuando da clic al botón aceptar o cancelar.

Condiciones de Ejecución: Se debe haber autenticado como administrador.

Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central

.

SC: 1 Adicionar cámara	EC 1.1: Adicionar satisfactoriamente.	El Administrador selecciona la opción "Adicionar Cámara" el Sistema le muestra una ventana para introducir los datos necesarios (Ejemplo: Nombre, Descripción y Fuente de Video, luego selecciona el botón "Siguiente" donde introduce los datos específicos de cada cámara según el plugin correspondiente para la conexión). El Administrador entra los datos necesarios correctamente y acepta la entrada, luego el Sistema	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre un grupo de cámaras formado (Ejemplo: Lab 304)/ opción "Adicionar Cámaras"
	EC 1.2: Adicionar una cámara ya existente.	El Administrador realiza los pasos necesarios para adicionar una cámara (EC 1.1). El sistema muestra un mensaje de error que alerta que la cámara ya existe.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre un grupo de cámaras formado (Ejemplo: Lab 304)/ opción "Adicionar Cámaras"/botón
SC:2 Eliminar Cámara	EC 2.1: Eliminar satisfactoriamente.	El Administrador selecciona la opción de eliminar una cámara, el Sistema muestra una advertencia "Está seguro que desea eliminar la cámara seleccionada" El Administrador elige continuar con la operación y el Sistema elimina la cámara.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre la cámara que se desea eliminar, (Ejemplo: Axis 211)/ opción "Eliminar Cámara"/ botón "Si"

	EC 2.2 Eliminar cámara. Cancelar.	El Administrador selecciona la opción "No", el sistema cancela la operación y vuelve a la interfaz anterior.	Módulo Visor/ Clic derecho en la cámara que se desea eliminar/ Eliminar cámara/No.
SC 3: Editar Cámara	EC 3.1: Editar satisfactoriamente.	Una vez que el Administrador selecciona la opción de editar una cámara el Sistema muestra los datos de la cámara. Luego modifica los datos, acepta y el Sistema almacena los datos modificados.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre la cámara que se desea editar, (Ejemplo: Axis 211)/ opción "Editar Cámara"/ clic en botón "OK"
	EC 3.2: Editar con campos vacíos o con datos incorrectos.	El Administrador realiza los pasos necesarios para Editar (EC 3.1). El sistema muestra un mensaje de error que alerta que existen campos vacíos o con datos incorrectos.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre la cámara que se desea editar, (Ejemplo: Axis 211)/ opción "Editar Cámara"/ clic en botón "OK"

Tabla 6. Gestionar cámaras.

3.1.3 Prueba de Sistema

Prueba de Volumen

Entidades relacionada	Atributos relacionados	Limites de datos inferior	Límite superior
Tbl_storagelog	Id_log	0	15
	descripción	0	200
	type	0	30
	fromwho	0	5
	eventdate	null	null
Tbl_camera	Id_camera	0	15
	tipo_camera	0	200
	fabricación	0	500
	name_camera	0	200
	ip	0	255
	provider	0	200
	descripción	0	255
	configprovider	0	100

Tabla 7. Herramientas necesarias para realizar prueba de volumen

3.2 Cronograma de aplicación de las pruebas.

Nro.	Período de aplicación(abril)	Prueba
1	2 - 5	Caja Blanca(Camino Básico)
2	6 - 9	Caja Negra(Partición Equivalente)
3	10 - 12	Integración(Inc. Ascendente)
4	13 – 16	Sistema(Volumen)

Tabla 8. Cronograma de aplicación de las pruebas

3.3 Resultados obtenidos

3.3.1 Pruebas de Unidad

De Caja blanca

Camino básico.

No del camino.	Caso de Prueba.	Objetivo.	Resultado.
1	Probando la función AddCamera.	Comprobar que todas las condiciones se cumplan para que se puedan adicionar las cámaras.	Satisfactorio.
1	Probando la función DeleteCamara.	Comprobar que todas las condiciones se cumplan para que se cumplan para que se puedan eliminar las cámaras.	Satisfactorio.
2	Probando la función UpdateCamera.	Comprobar que todas las condiciones se cumplan para que se puedan actualizar las cámaras.	Satisfactorio.
2	Probando la función UpdateUser.	Comprobar que todas las condiciones se cumplan para que se puedan actualizar los usuarios.	Satisfactorio.
1	Probando la función RemoveUser.	Comprobar que todas las condiciones se cumplan para que se puedan eliminar los usuarios.	Satisfactorio.
1	Probando la función UpdateRol.	Comprobar que todas las condiciones se cumplan para que se puedan actualizar los roles.	Satisfactorio
1	Probando la función AddRol.	Comprobar que todas las condiciones se cumplan para que se puedan adicionar los roles.	Satisfactorio

Tabla 9. Resultados de las pruebas de Caja Blanca

Después de analizados estos resultados se llega a la conclusión de que el sistema no muestra ningún problema en las funciones implementadas.

3.3.2 Pruebas de Aceptación (validación)

De caja negra

Partición equivalente

Luego de una primera iteración de pruebas al producto Video Vigilancia se obtuvieron un total de 73 no conformidades, de estas 38 de documentación y 35 de sistema. Para una segunda iteración se alcanzaron los resultados que se muestran a continuación.

Secciones	Resultados esperados	Resultados obtenidos
Analizar eventos.	Que el cliente pueda enlazarse a un evento determinado en el gestor. Para esto el sistema chequea el identificador del cliente y los permisos que este tiene. En caso de ser un ID inválido o que el cliente no tenga los permisos se levanta una excepción.	El sistema chequea el ID del cliente y realiza el enlace al evento satisfactoriamente.
Adicionar regla de grabación.	Que el cliente pueda solicitar adicionar una regla de grabación. Para esto el sistema debe chequear el ID del mismo. Si es válido el ID del cliente, entonces se almacenan los datos de la regla en la Base de Datos. En caso de ser un ID inválido se levanta una excepción.	El sistema verifica el ID del cliente y realiza la solicitud del mismo de forma satisfactoria.
Actualizar regla de grabación.	Que el cliente pueda solicitar actualizar una regla de grabación. Para esto el sistema debe chequear el ID del cliente. Si es válido el ID, entonces el sistema actualiza los datos de la regla en la Base de Datos. En caso de ser un ID inválido se levanta	El sistema realiza la solicitud del cliente de manera satisfactoria.

	una excepción.	
Eliminar regla de grabación	Que el cliente pueda solicitar eliminar una regla de grabación. El sistema chequea el ID del cliente. Si es válido este ID, entonces el sistema elimina los datos de la regla en la Base de Datos. En caso de ser un ID inválido se levanta una excepción.	El sistema realiza la solicitud del cliente satisfactoriamente.
Obtener regla de grabación.	Que el cliente pueda obtener una regla de grabación. Para esto el sistema chequea el ID del cliente. El sistema obtiene los datos de la regla en la Base de Datos. En caso de ser un ID inválido se levanta una excepción.	El sistema realiza la solicitud del cliente satisfactoriamente.
Adicionar rol.	Que al añadir el rol se introduzca el nombre del nuevo rol y al pulsar el botón “Aceptar”. El sistema adicione el nuevo rol. En caso de que no se introduzca ningún valor en el campo nombre del rol el sistema mostrará un mensaje de error indicando que verifique los datos. Si existe un rol con el nombre introducido el sistema mostrará un mensaje indicando que verifique los datos.	El sistema realiza la solicitud del cliente de forma satisfactoria.
Actualizar rol.	Que al editar en rol seleccionando la opción de actualizar rol, el sistema guarde los cambios y regrese al formulario de gestión de roles. Luego el Administrador selecciona la opción “Cancelar”, y el sistema cancela la operación y vuelve a la interfaz anterior.	El sistema realiza la operación satisfactoriamente,

Eliminar rol.	Que al Eliminar rol, seleccionando la opción "Eliminar rol", el sistema elimine dicho rol. Luego el Administrador selecciona la opción "Cancelar", el sistema cancela la operación y vuelve a la interfaz anterior.	La operación fue realizada de manera satisfactoria por parte del sistema.
Actualizar permisos de rol	Que al Actualizar permisos de rol, el Administrador seleccione o deseleccione los permisos que desee asignarle al rol seleccionado y presione el botón "Aceptar". Luego el sistema guarda los cambios realizados. Al finalizar la operación el Administrador selecciona la opción "Cancelar", el sistema cancela la operación y vuelve a la interfaz anterior.	La operación fue realizada satisfactoriamente.
Adicionar usuario.	Que el sistema muestre un formulario para entrar los datos del usuario ("Usuario", "contraseña", "confirmar contraseña", y selecciona el "rol") y seleccione la opción Aceptar. Luego el sistema almacena los datos del usuario. En caso de entrar los datos incorrectos, el sistema mostrará un mensaje indicando que los datos de entrada no son válidos. En el caso de que al entrar los datos, el usuario exista en la base de datos se mostrará un mensaje indicando que el usuario ya existe en el sistema. Por último el sistema cancela la operación y vuelve a la interfaz anterior.	El sistema adiciona el usuario satisfactoriamente.
	Que el administrador al seleccionar un usuario, el sistema muestre un	El sistema realiza la operación satisfactoriamente.

Actualizar usuario.	formulario con la información del usuario que desea modificar “nuevo nombre”, “contraseña”, “confirmar contraseña” y “rol”. Luego de esto el administrador modifica los datos del usuario y selecciona la opción “Aceptar”. Al finalizar la operación el sistema comprueba los datos y al encontrar errores muestra un mensaje indicando que los datos entrados son incorrectos. Por último el sistema cancela la operación y vuelve a la interfaz anterior.	
Eliminar usuario.	Que el sistema elimine el usuario o los usuarios seleccionados de la Base de Datos. Luego el Administrador selecciona la opción “Cerrar”, el sistema cancela la operación y vuelve a la interfaz anterior.	El sistema elimina el usuario o los usuarios de la Base de Datos satisfactoriamente.
Adicionar cámara	Que el Administrador seleccione la opción “Adicionar Cámara” el Sistema le muestre una ventana para introducir los datos necesarios (Ejemplo: Nombre, Descripción y Fuente de Video, luego seleccione el botón “Siguiente” donde introduce los datos específicos de cada cámara según el plugin correspondiente para la conexión). El Administrador entra los datos necesarios correctamente y acepta la entrada, luego el Sistema almacena los nuevos datos. Por último el Administrador realiza los pasos necesarios para adicionar una cámara (EC 1.1) y el sistema muestra un	La solicitud es realizada de manera exitosa por el sistema.

	mensaje de error que alerta que la cámara ya existe.	
Editar cámara	Que una vez que el Administrador seleccione la opción de editar una cámara el Sistema muestre los datos de la cámara. Luego modifica los datos, acepta y el Sistema almacena los datos modificados. Por último el Administrador realiza los pasos necesarios para Editar (EC 3.1), y el sistema muestra un mensaje de error que alerta que existen campos vacíos o con datos incorrectos.	La operación es realizada satisfactoriamente por parte del sistema.
Eliminar cámara	Que el Administrador seleccione la opción de eliminar una cámara, el Sistema muestre una advertencia "Está seguro que desea eliminar la cámara seleccionada" Luego el Administrador elige continuar con la operación y el Sistema elimina la cámara. Por último el Administrador selecciona la opción "No", el sistema cancela la operación y vuelve a la interfaz anterior.	El sistema realiza la operación de manera satisfactoria.

Tabla 10. Resultados de las pruebas de Caja Negra

3.3.3 Pruebas de Integración

Incremental ascendente

A medida que se van terminando los módulos se integran hasta tener el sistema construido totalmente y ver que funciona como un todo. Luego de realizar las pruebas de caja negra se integraron los módulos y se pudo comprobar que están debidamente integrados y que funcionan correctamente. Cuando se crea un usuario y se le otorgan los permisos según el rol que se le asigne, el usuario accede a los módulos a los cuales puede acceder según las restricciones de

permisos asignadas. También el sistema permite adicionar o eliminar usuarios correctamente, así como actualizarlo, en caso de no poder adicionarlo porque ya exista en la base de datos entonces aparecerá un mensaje informándolo, y en el caso de eliminarlo, si no existe también aparece un mensaje dándolo a conocer. Todas estas funciones entre otras están validadas y funcionan correctamente.

3.3.4 Prueba de sistema.

Prueba de volumen.

Nombre del CU relacionado con la entidad	Nombre de la sección.	Técnicas a Realizar			Descripción de la funcionalidad	Flujo central	Resultado de la Prueba
		Gran cantidad de peticiones con gran cantidad de datos cada una.	Una petición con gran cantidad de datos.	Gran cantidad de peticiones con una cantidad de datos baja cada una.			
Registrar logs	Registrar logs	X			El Sistema realiza cualquier actividad y se registra un logs de dicha operación.	Módulo Visor/ en Lista de Cámaras/ /clic derecho sobre “Cámaras” o sobre un grupo existente/ opción “Adicionar Grupo”	El sistema reacciona de forma satisfactoria antes las peticiones que se le realizan.

Gestionar cámara	Adicionar cámara.	X	X		El Administrador selecciona la opción "Adicionar Cámara" el Sistema le muestra una ventana para introducir los datos necesarios.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre un grupo de cámaras formado (Ejemplo: Lab 304)/ opción "Adicionar Cámaras"	El sistema reacciona de forma satisfactoria antes las peticiones que se le realizan.
	Eliminar cámara.	x		X	El Administrador selecciona la opción de eliminar una cámara, el Sistema muestra una advertencia "Está seguro que desea eliminar la cámara seleccionada" El Administrador elige continuar con la operación y el Sistema elimina la cámara.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre la cámara que se desea eliminar, (Ejemplo: Axis 211)/ opción "Eliminar Cámara"/ botón	El sistema reacciona de forma satisfactoria antes las peticiones que se le realizan.

	Editar cámara.	X			Una vez que el Administrador selecciona la opción de editar una cámara el Sistema muestra los datos de la cámara. Luego modifica los datos, acepta y el Sistema almacena los datos modificados.	Módulo Visor/ ficha "Configuración"/ clic en "Lista de cámaras"/ clic derecho sobre la cámara que se desea editar, (Ejemplo: Axis 211)/ opción "Editar Cámara"/ clic en botón "OK"	El sistema reacciona de forma satisfactoria antes las peticiones que se le realizan.
Manejar la conexión a la BD	Manejar la conexión a la Base de Datos.	x			El sistema, cada cierto tiempo chequea el estado de la conexión a la base de datos y actualiza el estado a disponible.	Clic en "Suria Service Controller"/ botón "Configurar"/ cambiar datos en la BD/ botón "OK".	El sistema realiza las peticiones de conexión a la base de datos satisfactoriamente
Obtener usuarios	Obtener usuarios	x			Una vez que el cliente solicita obtener un usuario, el sistema chequea el ID del mismo, y luego termina cuando el sistema obtiene los	Módulo Visor/ loguearse/ botón "Aceptar.	El sistema realiza las peticiones de obtener usuarios satisfactoriamente

					datos del usuario en la Base de Datos.		

Tabla 11 Resultados de las pruebas de volumen

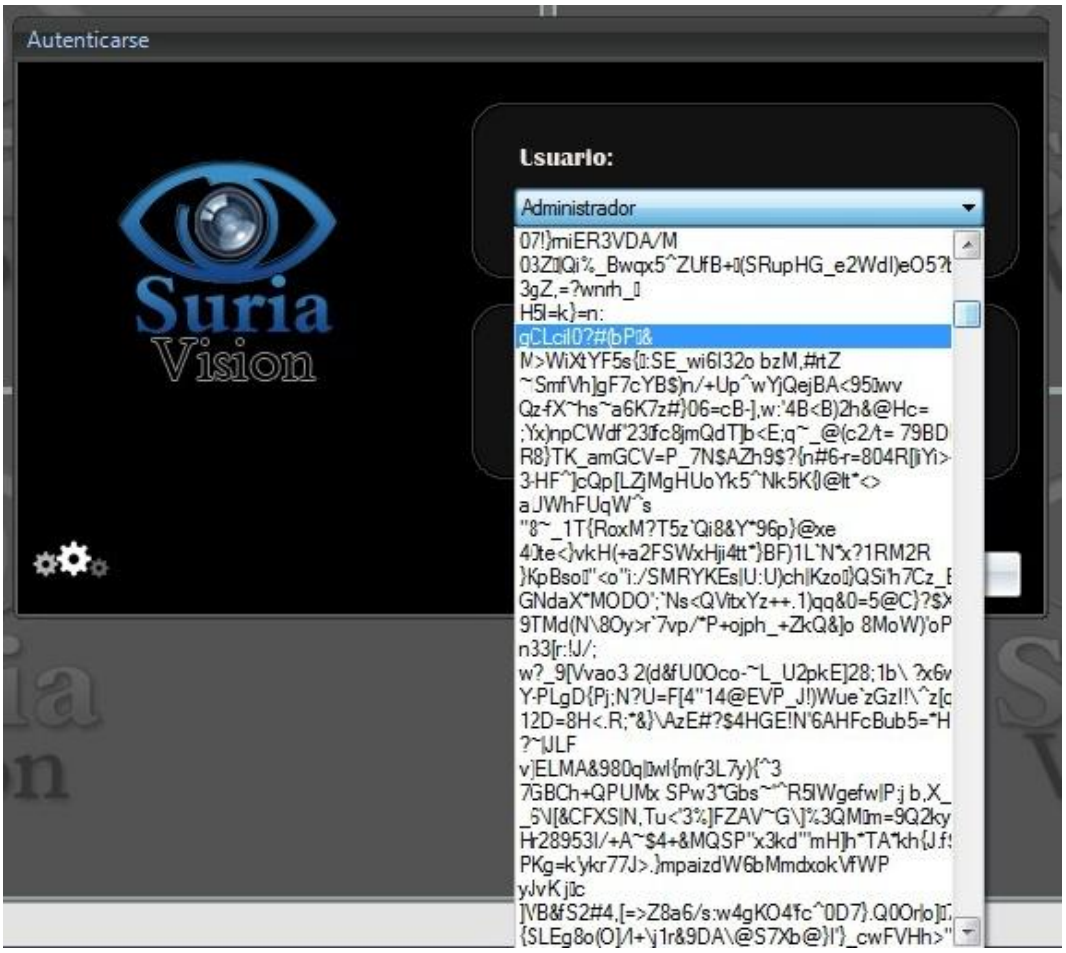


Figura 7: Resultado de las pruebas de volumen (Obtener Usuario)

Conclusiones parciales del capítulo.

Al finalizar este capítulo, se puede decir que el proceso de pruebas realizado al producto Video Vigilancia se llevó a cabo de forma satisfactoria, pues mediante los casos de prueba confeccionados se obtuvieron muchos de los errores con los que contaba este producto. En este capítulo quedaron documentados todos los defectos encontrados, luego de haber realizado las pruebas de función (caja negra y caja blanca), integración y sistema, con el fin de que fueran corregidos por el equipo de desarrollo en el menor tiempo posible. En fin luego de haber

culminado este capítulo se cumplió el objetivo fundamental del mismo, que era dar a conocer todos los defectos con los que contaba el producto luego de aplicar las pruebas correspondientes al mismo.

Conclusiones Generales

Durante la realización de este trabajo de diploma se ejecutaron y evaluaron una serie de casos de prueba diseñados con diferentes tipos de pruebas para obtener la mayor cantidad de errores posibles existente. Para esto se enfatizó en la metodología de desarrollo del software definida por el proyecto Video Vigilancia, específicamente a la hora de aplicar las pruebas de Caja Negra y Caja Blanca para la ejecución de los casos de prueba, así como en las de integración y sistema.

Para lograr un alto nivel de calidad en el producto se hizo necesario realizar un proceso de prueba basándose en comprobaciones específicas en cada caso y en cada fase de desarrollo del producto.
Al planificarse y aplicarse las pruebas pertinentes al producto Video Vigilancia, se llegó a las siguientes conclusiones:

- Se logró alcanzar un alto nivel de conocimiento en cuanto a los tipos de pruebas y métodos aplicados al producto Video Vigilancia.

- Durante el proceso de pruebas se crearon los artefactos: Diseño de Caso de Prueba, Plan de prueba y la Planilla de No Conformidades.

- Se diseñaron casos de prueba para verificar el buen funcionamiento de la aplicación.

- Se logró documentar los errores detectados en la planilla de No Conformidades, los cuales serán corregidos posteriormente por el equipo de desarrollo del proyecto.

De forma general se puede decir que el flujo de prueba se ejecutó exitosamente partiendo de que el éxito de las pruebas se encuentra en la detección de la mayor cantidad de errores posibles.
Además, de que se lograron los objetivos propuestos a inicios de este trabajo, logrando que se llevara a cabo un proceso de pruebas al producto Video Vigilancia con el fin de garantizarle la calidad.

Recomendaciones

Una vez concluido este trabajo y dada la importancia que tiene el proceso de pruebas para cualquier sistema informático, se recomiendan los siguientes aspectos que se mencionan a continuación.

1. Hacer extensiva la fase de prueba a los demás proyectos de la facultad, llevando la documentación necesaria cada vez que se le apliquen.
2. Perfeccionar el plan de pruebas teniendo en cuenta la retroalimentación obtenida con la aplicación de las pruebas a los módulos del proyecto Video Vigilancia.

Además se recomienda que se siga sometiendo el producto a pruebas de sistema de manera exhaustiva para así lograr una mejor calidad del mismo, también que se le aplique pruebas de seguridad, una vez que el producto ya esté desplegado, para verificar cómo funciona el mismo ante situaciones anormales y que los datos solo sean accedidos por las personas que cuentan con los privilegios necesarios.

Trabajos citados

Collazo, Manuel. 2003.. Técnicas de prueba del software. Estrategias de prueba del software. 2003.

IEEE Standard Glossary of Software Engineering Terminology 1990, .

Granada.

Mañas, José A. 1994. Laboratorio de programación. [Online] 3 16, 1994. [Cited: 12 2, 2010.] http:// www.lab.dit.upm.es/ Material de estudio/› Apuntes/pruebas/testing.htm..

McCall.J. 1977. *Factors in software Quality:General Electric.Factors in software Quality:General Electric. 1977.* 1977.

Myers, Glenford J. 2004. The art of sotware Testing, page 234.Wiley, second edition. [Online] 2004. [Cited: 12 2, 2010.] www.wiziq.com/tutorial/41789-pruebas -.

Oscar M. Fernández Carrasco, Delba García León y Alfa Beltrán Benavides. 1995. *Un enfoque actual sobre la calidad del software.* 1995.

Pressman, R.S. 2005.. *Ingeniería del Software, un enfoque práctico.* 2005.

Pressman, Roger S. 2002. Ingeniería del software, un enfoque práctico. [Online] McGrawHill, 2002. [Cited: 11 29, 2010.] www.wiziq.com/tutorial/41789-pruebas -.

Sánchez, María A. Mendoza. 2004. Informatizate. [Online] 2004. [Cited: 11 15, 2010.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html..

TecnoRetales. 2009. TecnoRetales is proudly using the Buddy theme. Powered by WordPress. [Online] 2009. [Cited: 12 3, 2010.] http://www.tecnoretales.com/programacion/phpunit-tests-unitarios-en-php/..

Vidal., Francisco Vega Leyte. 2010. *Diseño de la arquitectura de software.* 2010.

Weitzenfeld., Alfredo. 1989. *Ingieneria de Software Orientada a Objetos.* s.l. : Thomson, 1989.

Wiley, John. 1983. *Software Engineering Standards. s.l. : IEEE Press, 2001. ANSI/.* 1983. IEEE Standard 829-1983.

Técnicas de Evaluación Dinámica. Técnicas de Evaluación Dinámica. [En línea] [Citado el: 28 de 11 de 2010.] www.lsi.us.es/docencia/get.php?id=361..

Computación., LCD. Laboratorio Docente de. 2008. LCD. [Online] 2008. [Cited: 2 3, 2011.] http://www.ldc.usb.ve/~teruel/ci4713/clases2001/planPruebas.html...

Mora., Oberto Canales. 2007. Adictos al Trabajo. [Online] 2007. [Cited: 2 1, 2011.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=cmmi.

Hernández, Meléndrez, Dra. Edelsys. 2007.. Cómo Escribir una Tesis. [aut. libro] León, Rolando Alfredo y Coello, González, Sayda. Hernández. EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA. Cuba : Escuela Nacional de Salud Pública, 2007.

Jorge. (Mayo de 2005). Pruebas de Funcionalidad. Recuperado el 5 de 4 de 2011, de carolina.terna.net/ingsw3/datos/Pruebas_Funcionales.pdf

Figuras. Técnicas de Evaluación Dinámica . [Online] [Cited: 2 10, 2011.] www.lsi.us.es/docencia/get.php?id=361.

Pareto. taller-unidad-ii-teora-de-pareto-y-la.html . [Online] [Cited: 5 5, 2011.] tecnicasdedecision.blogspot.com.

Bibliografía

Collazo, Manuel. 2003.. *Técnicas de prueba del software. Estrategias de prueba del software.* 2003.

IEEE Standard Glossary of Software Engineering Terminology 1990, .

Granada.

Mañas, José A. 1994. Laboratorio de programación. [Online] 3 16, 1994. [Cited: 12 2, 2010.] http:// www.lab.dit.upm.es/ Material de estudio/› Apuntes/pruebas/testing.htm..

McCall.J. 1977. *Factors in software Quality:General Electric.Factors in software Quality:General Electric. 1977.* 1977.

Myers, Glenford J. 2004. The art of sotware Testing, page 234.Wiley, second edition. [Online] 2004. [Cited: 12 2, 2010.] www.wiziq.com/tutorial/41789-pruebas -.

Oscar M. Fernández Carrasco, Delba García León y Alfa Beltrán Benavides. 1995. *Un enfoque actual sobre la calidad del software.* 1995.

Pressman, R.S. 2005.. *Ingeniería del Software, un enfoque práctico.* 2005.

Pressman, Roger S. 2002. Ingeniería del software, un enfoque práctico. [Online] McGrawHill, 2002. [Cited: 11 29, 2010.] www.wiziq.com/tutorial/41789-pruebas -.

Sánchez, María A. Mendoza. 2004. Informatizate. [Online] 2004. [Cited: 11 15, 2010.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html..

TecnoRetales. 2009. TecnoRetales is proudly using the Buddy theme. Powered by WordPress. [Online] 2009. [Cited: 12 3, 2010.] http://www.tecnoretales.com/programacion/phpunit-tests-unitarios-en-php/..

Vidal., Francisco Vega Leyte. 2010. *Diseño de la arquitectura de software.* 2010.

Weitzenfeld., Alfredo. 1989. *Ingieneria de Software Orientada a Objetos.* s.l. : Thomson, 1989.

Wiley, John. 1983. *Software Engineering Standards. s.l. : IEEE Press, 2001. ANSI/.* 1983. IEEE Standard 829-1983.

Técnicas de Evaluación Dinámica. Técnicas de Evaluación Dinámica. [En línea] [Citado el: 28 de 11 de 2010.] www.lsi.us.es/docencia/get.php?id=361.

Computación., LCD. Laboratorio Docente de. 2008. LCD. [Online] 2008. [Cited: 2 3, 2011.] http://www.ldc.usb.ve/~teruel/ci4713/clases2001/planPruebas.html..

Pressman, McGraw-Hill. R.S. 1998. . Ingeniería de Software. Un enfoque Práctico. 1998.

Mora., Oberto Canales. 2007. Adictos al Trabajo. [Online] 2007. [Cited: 2 1, 2011.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=cmmi.

M.Torres. Tecnicas de prueba. indalog.ual.es/mtorres/LP/Prueba.pdf

Jorge. (Mayo de 2005). Pruebas de Funcionalidad. Recuperado el 5 de 4 de 2011, de carolina.terna.net/ingsw3/datos/Pruebas_Funcionales.pdf

Ministerio de Administraciones Públicas. (2002.). Metodología MÉTRICA versión 3. Técnicas y Prácticas. Recuperado el 5 de 4 de 2011, de gemini.udistrital.edu.co/.../Pruebas/...%20Pruebas%20de%20software/node41.html –

(s.f.). Recuperado el 5 de 4 de 2011, de interop.cenditel.gob.ve/wiki/plan_PruebaNoFuncional.

Figuras. Técnicas de Evaluación Dinámica . [Online] [Cited: 2 10, 2011.] www.lsi.us.es/docencia/get.php?id=361.

Pareto. taller-unidad-ii-teora-de-pareto-y-la.html . [Online] [Cited: 5 5, 2011.] tecnicasdedecision.blogspot.com.

Glosario

Calidad: Medida de la cierta característica deseable.

V&V: Verificación y validación.

CEV: Clases de equivalencia válidas.

CEI: Clases de equivalencia inválidas.

RUP: Proceso unificado de desarrollo.

CU: Caso de uso.

Anexos:

Ejemplos de Caja Blanca:

```
private bool CanIDoIt()
{
   if (Status == ServerModelStatus.Online) 1
           {
           log.Info("Puede realizarse la operación"); 2
           return true; 2
           }
  else
           {
           log.Info("No Puede realizarse la operación"); 3
           return false; 3
           }
} 4
```

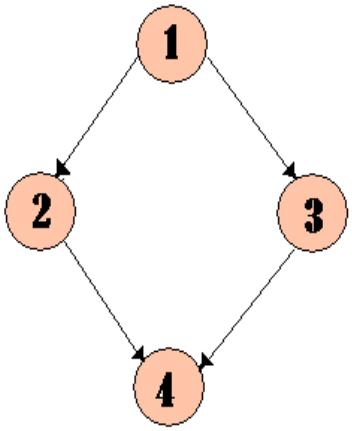


Figura 8. Grafo de Flujo

Cálculo de la Complejidad Ciclomática

V (G) = Numero de Aristas – Número de Nodos + 2

V (G) = 4 – 4 + 2

V (G) = 2

Caminos:
1: 1 - 2 - 4
2: 1 - 3 – 4

```
public bool UpdateUser(SIS_UserRemotingInfo pRemoteUser, int ClientID)
 {
     if (CheckClientID(ClientID) && this.CanIDoIt())1
    {
        SIS_User user=new SIS_User(pRemoteUser); 2
       bool ret= _loader.UpdateUser(user); 2
       try 2
       {
       if (ret) 3
         {
           SIS_UserRemotingInfo info =
SIS_Broker.RegistedSecurityManager.Users.GetUserByID(user.ID).GetRemotingProxy(); 4
           SendUserChangedMessage(_OnUserChange, ClientID, info, false); 4
         }
       }
        catch { } 5
        return ret; 5
    }
    else
    {
        log.Error("O bien el cliente no esta registrado en este Gestor o el Gestor esta offline,
chequee el estado del Gestor"); 6
        throw new InvalidOperationException("O bien el cliente no esta registrado en este Gestor o
el Gestor esta offline, chequee el estado del Gestor"); 6
    }
 }
```

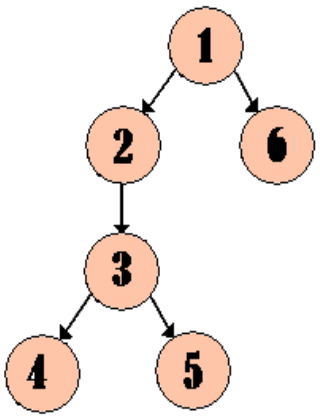


Figura 9.Grafo de flujo

Cálculo de la Complejidad Ciclomática

V (G) = Numero de Aristas – Número de Nodos + 2

V (G) = 5 – 6 + 2

V (G) = 3

Caminos:

1: 1-6

2: 1-2-3-4

3: 1-2-3-5

Ejemplos de Caja Negra:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Grabar por Demanda	EC 1.1: Grabar por Demanda exitosamente.	El cliente solicita comenzar a grabar de una cámara determinada. El sistema chequea el identificador del cliente (se refiere al ID el cual no es verificable por el usuario) y los permisos que tiene. El sistema selecciona al grabador que esté en mejor capacidad de atender la petición y entrega dicha solicitud a ese grabador.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / clic derecho sobre una cámara/ opción “Comenzar a grabar”
	EC 1.2: Grabar por Demanda fallido.	En caso de ser un ID inválido (mismo caso de ID del EC 1.1) o que el cliente no tenga los permisos se levanta una excepción.	Módulo Visor/ Pestaña Grabación/ clic Cliente de Grabación - Suria Recorder / clic derecho sobre una cámara/ opción “Comenzar a grabar”

Tabla 12 Caso de prueba grabar por demanda

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Autenticar usuario.	EC 1.1: Autenticar usuario correctamente.	El usuario inserta su nombre y su contraseña en los campos correspondientes, el sistema busca el usuario en la base de datos y compara si la contraseña es la correcta, si es así se le permite el acceso al sistema con los permisos que tenga asignado según su rol.	Módulo Visor/ clic en "Suria Viewer"/ botón "Aceptar".
	EC 1.2: Autenticar usuario incorrectamente.	Si el nombre del usuario seleccionado no coincide con la contraseña proporcionada se muestra un mensaje de "Error, verifique su contraseña". Se ofrece la oportunidad de volver a introducir sus datos.	Módulo Visor/ clic en "Suria Viewer"/ botón "Aceptar".
	EC 1.3: Cancelar Autenticar usuario.	El usuario selecciona el botón "Cancelar". El sistema cancela la operación y abre el Visor sin ninguna funcionalidad con la opción de volver a autenticarse.	Módulo Visor/ clic en "Suria Viewer"/ botón "Cancelar".

Tabla 13 Caso de prueba Autenticar Usuario

Descripción de variables:

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Usuario	Lista desplegable.	No.	Se selecciona el rol en dependencia de los privilegios de cada usuario.

2	Contraseña	Campo de texto.	No.	Se introduce la contraseña que el usuario tiene para su acceso al sistema.

Tabla 14 Descripción de variables del caso de prueba Autenticar Usuario

Matriz de datos:

ID del escenario	Escenario	Variable 1 Usuario	Variable 2 Contraseña	Respuesta del Sistema	Resultado de la Prueba
EC 1.1	Autenticar usuario correctamente.	V/Administrador	V/admin	El usuario accede al sistema con los permisos que le sean asignados según su rol.	
EC 1.2	Autenticar usuario incorrectamente.	V/Administrador	I/administrador	El usuario no puede acceder al sistema, y se le da la opción de insertar sus datos nuevamente.	
EC 1.3	Cancelar Autenticar usuario.	V/Administrador	V/admin	Se abre el Visor sin ninguna funcionalidad con la opción de volver a autenticarse	

Tabla 15 Matriz de datos del caso de prueba Autenticar Usuario

Ejemplos de Pruebas de Volumen

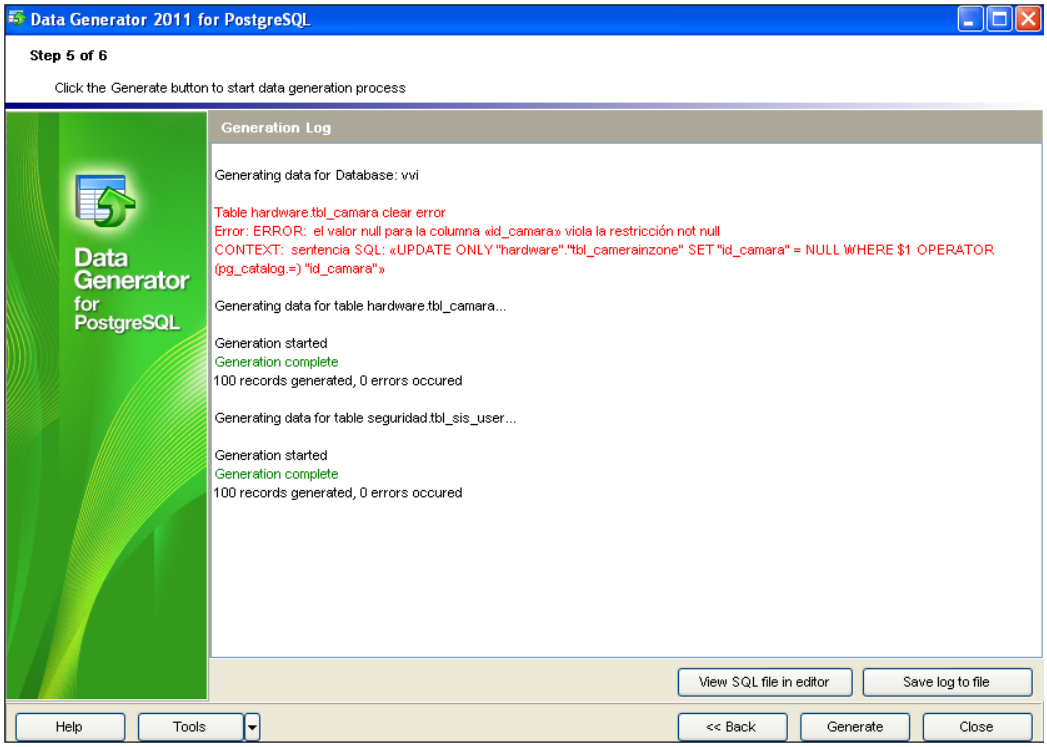


Figura 10 Prueba a la tabla Cámara.

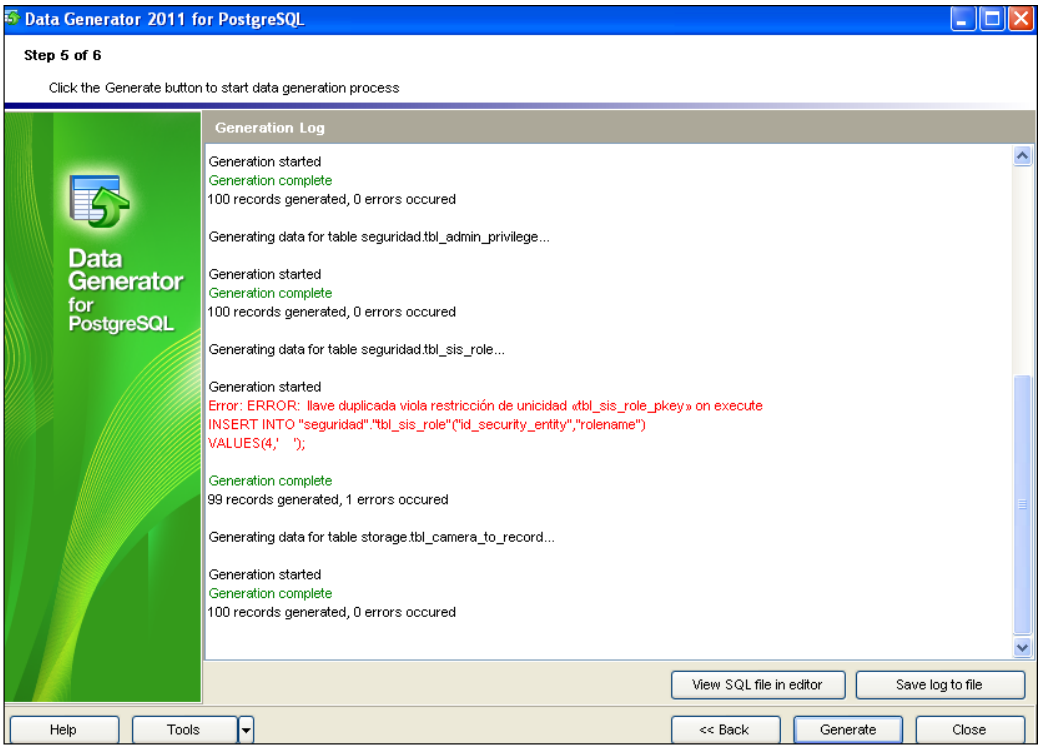


Figura 11 Prueba a la tabla Cámara_to_record.bmp

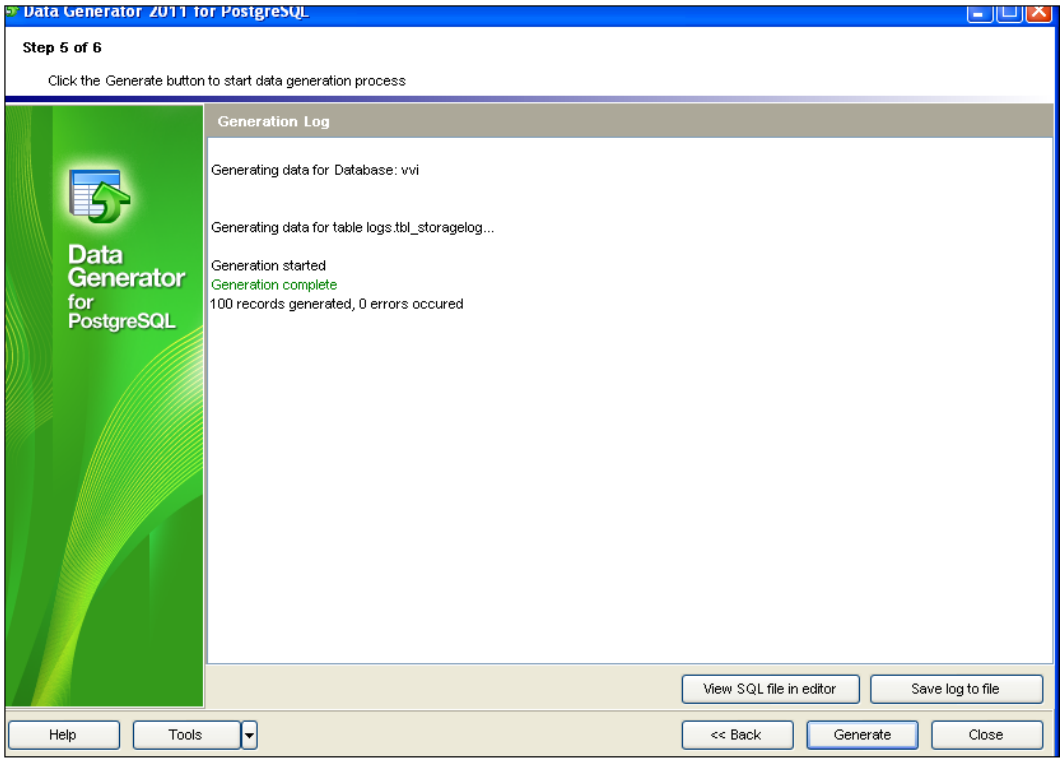


Figura 12 Prueba a la tabla Storagelog

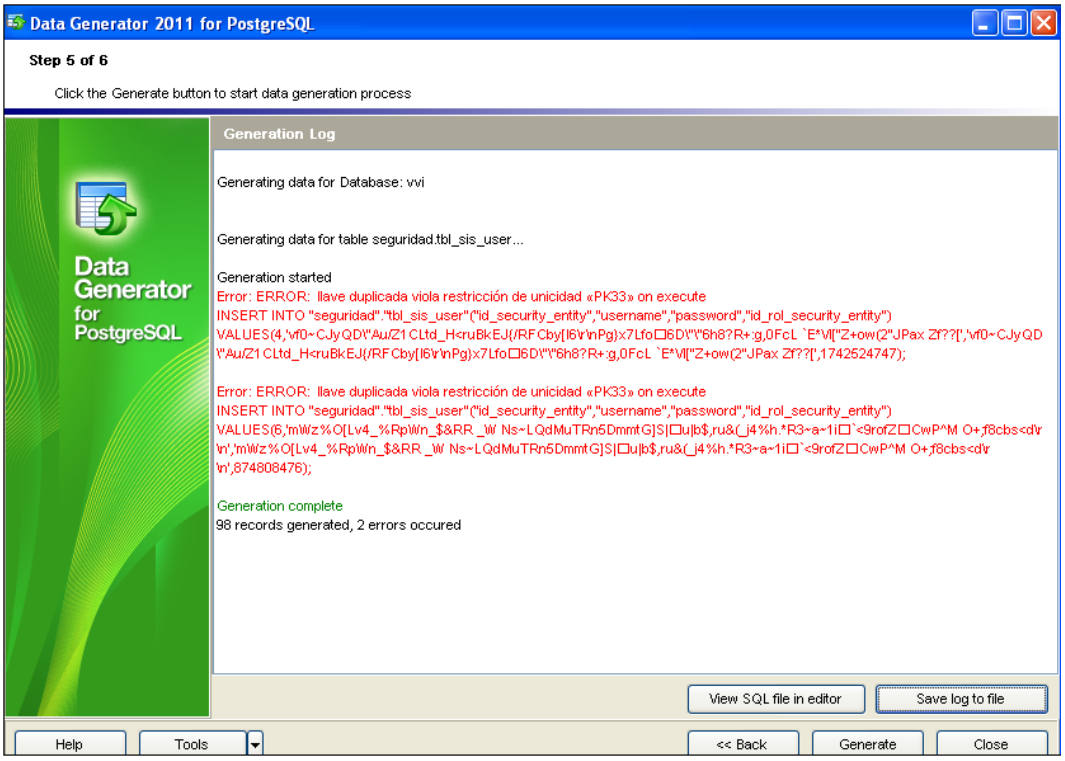


Figura 13 Prueba a la tabla_sis_user.jpg.