Universidad de las Ciencias Informáticas

Facultad 6

"Proceso de pruebas para la evaluación de la calidad del Generador Dinámico de Reportes en su versión 1.7.0"

Trabajo de Diploma para optar por el Título de

Ingeniero Informático

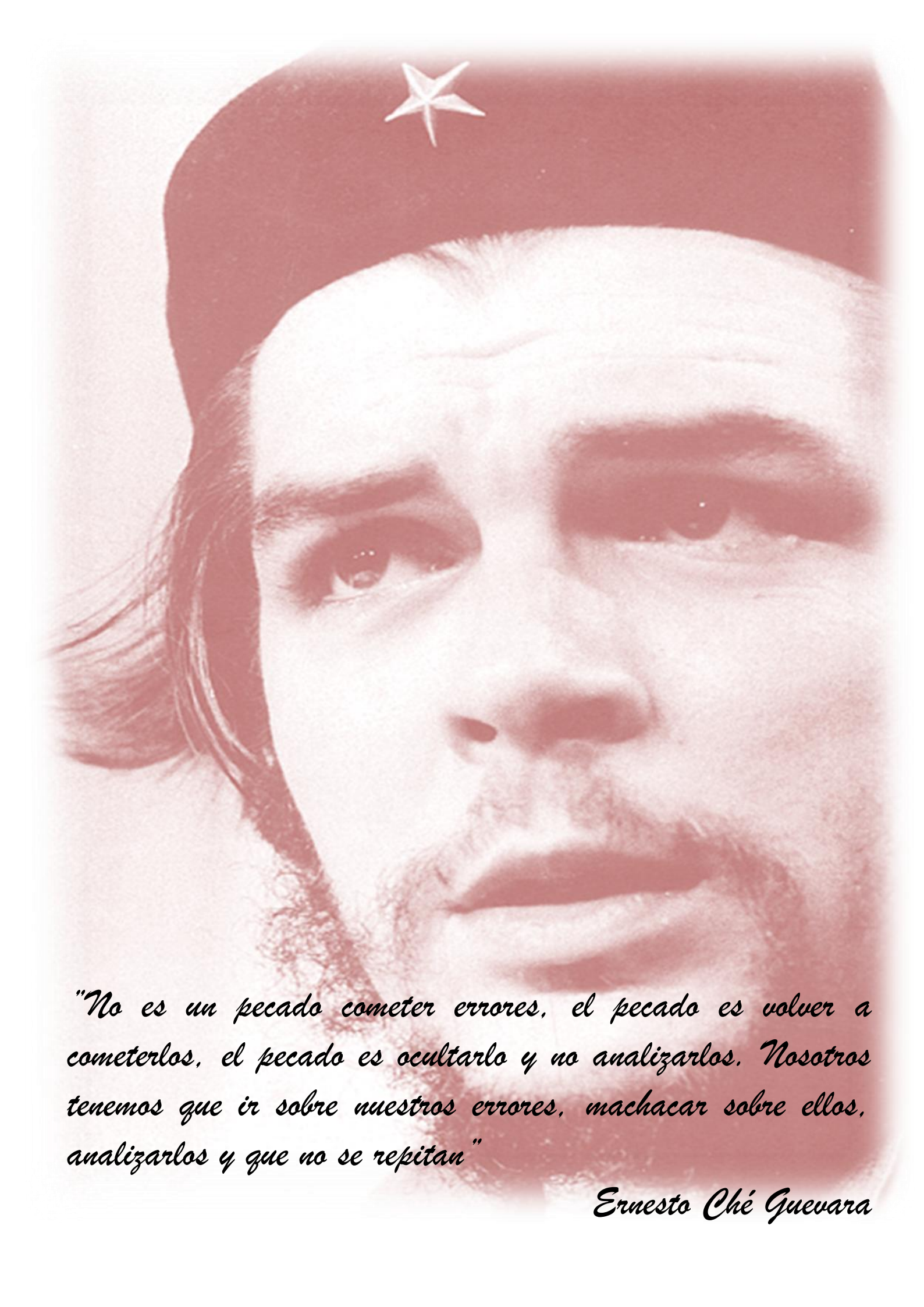
Autor: Yudelkis Olivera Argota.

Tutor: Ing. Reynaldo Alvarez Luna.

Co-Tutor: Ing. Claudia Núñez Sanz.

La Habana, junio del 2011

"Año 53 de la Revolución"



"No es un pecado cometer errores, el pecado es volver a cometerlos, el pecado es ocultarlo y no analizarlos. Nosotros tenemos que ir sobre nuestros errores, machacar sobre ellos, analizarlos y que no se repitan"

Ernesto Ché Guevara

Declaración de autoría Declaramos que somos autores de la presente tesis y reconocemos, a la Universidad de las Ciencias Informáticas, sus derechos patrimoniales sobre la misma con carácter exclusivo.

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

Yudelkis Olivera Argota:

_____

Firma del Autor

Ing. Reynaldo Alvarez Luna:

_____

Firma del Tutor

AGRADECIMIENTOS

A todos aquellos que con su amor y cariño estuvieron siempre presente y han formaron parte de mi vida:

A mis queridos padres por su comprensión, paciencia y dedicación. Gracias por enseñarme el camino correcto, sin ustedes, este momento no hubiese sido posible.

A mi tía Marbelis, por haberme enseñado la fuerza y voluntad para encaminarme en el camino.

A mis queridos tíos, por demostrarme su cariño. Gracias por la confianza y el amor que me brindaron.

A mis primos, abuelos, tíos, en fin, a toda mi familia en general, por ser tan especiales, por la que me tildo de privilegiada al poder contar con su amor y cariño y demostrarme que no hay nada más valioso en la vida que la familia.

A todos mis maestros y profesores por ayudarme a ser mejor persona y compartir su sabiduría conmigo.

A ti Javier, por tu amor incondicional y desmedido.

A todos aquellos vecinos que confiaron en mí y de esta manera deseo mandarles la mayor de las gratitudes por haber compartido

junto a mi familia.

A mis amistades del barrio, que aunque lejos, siempre los tuve presente.

A todos mis compañeros y amigos, por permitieron disfrutar al máximo estos maravillosos años a su lado, por ser partícipes de mis logros, confusiones, malos y buenos momentos y mis locuras, marcando una etapa inolvidable en mi vida.

He llegado al final de este camino y en mi han quedado marcadas huellas profundas de éste recorrido. En general, agradezco a todos aquellos que pusieron su granito de arena para que un día como hoy, pudiera dedicarles estas palabras.

DEDICATORIA

Les dedico especialmente este trabajo a quienes me han iluminado de amor y ternura. A quienes sin escatimar esfuerzo alguno, han sacrificado gran parte de su vida para formarme y educarme. A quienes la ilusión de su vida ha sido convertirme en una persona de provecho. A quienes nunca podré pagar todos sus desvelos ni aún con las riquezas más grandes del mundo. A ti madre y padre, son para ustedes mis más sentidos agradecimientos. Quiero que sientan que el objetivo logrado también es de ustedes y que la fuerza que me ayudó a conseguirlo fue su apoyo.

Por esto y mucho más… Gracias.

RESUMEN

El desarrollo de un proceso de pruebas para la evaluación de la calidad del Generador Dinámico de Reportes en su versión 1.7.0 está orientado al establecimiento y ejecución de una estrategia de pruebas que permitan determinar un reporte más amplio del estado del software.

Para lograr el objetivo propuesto, se estableció la argumentación de elementos importantes para el desarrollo de las pruebas, entre los que se trataron los niveles, técnicas y métodos de prueba existentes, así como diversas herramientas para automatizar algunas de las pruebas a ejecutar con vista a determinar la calidad del software. Se realizó de lo antes mencionado, una selección de los mejores procesos a utilizar.

Se estableció el diseño y aplicación de los casos de pruebas, en la que a partir de la elaboración de una estrategia se pudo garantizar el orden de los procesos. Luego de la ejecución de las pruebas se estableció un análisis de los resultados.

Para determinar la evaluación del sistema se realizó a partir de los atributos de calidad una valoración de los diferentes indicadores que caracterizan al generador a partir de la aplicación de listas de chequeo. Fue posible detectar los errores existentes con vista a erradicarlos en su futura versión. Para dar culminación, se propuso un proceso de mejora según lo obtenido.

PALABRAS CLAVES

Evaluación de la calidad del software, Proceso de pruebas, Casos de pruebas.

INTRODUCCIÓN

En el mundo globalizado de hoy, la calidad de los productos se ha convertido en una necesidad indispensable para permanecer en el mercado, es por ello que la industria del software realiza grandes esfuerzos por mejorar la calidad de los productos. No ha constituido una tarea fácil, ya que su tamaño y la complejidad aumenta rápidamente, mientras que los clientes y usuarios son cada vez más exigentes. A pesar de resultados alentadores con diversos enfoques de mejora de la calidad, la industria del software aún está lejos de lograr software sin defectos.

La calidad del software implica un conjunto de cualidades que caracterizan y determinan su utilidad y existencia. Calidad es sinónimo de eficiencia, flexibilidad, confiabilidad, usabilidad, seguridad e integridad. Es por ello que se hace imprescindible tener en cuenta su control durante todas las etapas del ciclo de vida del software. El lograr el éxito en la producción de software es hacerlo con calidad y demostrar su buena calidad. [1]

Según Pressman **"**La prueba es un elemento crítico para la calidad del software**"** [2]. Estas son utilizadas por todas las comunidades de desarrollo de sistemas para ayudar en la identificación de fallas, así como determinar si un producto puede desarrollarse según lo especificado en sus requisitos. La importancia de los costos asociados con los errores, promueve la definición y aplicación de un proceso de pruebas minuciosas y bien planificadas. [3]

Las empresas cubanas del software están llamadas a convertirse en una significativa fuente de ingresos para el país. El lograr la producción sostenida de alta calidad en el país, tendrá una positiva repercusión en el incremento de la exportación [4], lo que provoca que las empresas dedicadas a la producción de software tiendan a buscar las vías idóneas para su perfeccionamiento, permitiendo mejorar la producción, garantizar la calidad y lograr la satisfacción de todos los usuarios. Es por ello que la Universidad de las Ciencias Informáticas, una gran expresión del software cubano, tiene como propósito fundamental, contribuir en el desarrollo de la informatización y situar la producción cubana de software en el mercado mundial, por lo que la obtención de productos con un alto nivel de calidad debe encontrarse vigente en cada unos de los centros de desarrollo de la Universidad.

Uno de los productos principales del Centro de Tecnologías de Datos (DATEC) de la Universidad de las Ciencias Informáticas (UCI) es el Generador Dinámico de Reportes (GDR), por ser una solución genérica y adaptable a múltiples entornos ha sido parte de varias soluciones integrales brindadas en proyectos nacionales y de exportación, tales como, el proyecto reconocimiento de patrones (RP), así como proyectos vinculados con la Oficina Nacional de Estadísticas(ONE), desarrollados todos en la Universidad, se hace uso en el Ministerio del Poder Popular para la Energía y Petróleo(MENPET), el

Ministerio de Interior y Justicia (MIJ), el Servicio Autónomo de Registros y Notarías(SAREN) y el Ministerio del Poder Popular para la Comunicación y la Información (MinCI), todas entidades de la república Bolivariana de Venezuela.

Este proyecto brinda la posibilidad de chequear el funcionamiento periódico de una o varias entidades, mediante la formulación de reportes en diferentes formatos y modelos personalizables, obtenidos a través de datos estadísticos existentes en las bases de datos que de este se pueden importar. Actualmente la versión 1.7.0 del producto concluyó su proceso de liberación. En esta etapa el producto requiere un mayor nivel de validación y comprobación de su rendimiento y calidad para hacerlo más competitivo, esta razón viene aparejada a partir de la propuesta de la realización de una versión 2.0.0 del Generador Dinámico de Reportes, que amplíe los niveles de calidad. Es por ello que se hace necesaria la intensificación de pruebas que muestren un amplio reporte del grado de eficiencia en que se encuentra el producto. El proceso de liberación concluido incluyó únicamente pruebas de integración y de sistema para comprobar la satisfacción de los requisitos funcionales del sistema y validar su funcionamiento, pero carece de otros niveles de pruebas más específicos y avanzados que permitirían una evaluación más completa que tenga en cuenta otros factores tales como: la evaluación de la complejidad de sus funciones, el rendimiento, así como pruebas de volumen; factores importantes para el conocimiento del equipo de desarrollo, que les permiten erradicar los problemas que se presenten en su nueva versión.

Por lo tanto se identifica como **problema de la investigación**, que guía a la realización del trabajo de diploma: ¿Cómo evaluar la calidad del Generador Dinámico de Reportes en su versión 1.7.0?; teniendo como **objeto de estudio:** Proceso de pruebas para la evaluación de la calidad de productos de software; y enmarcando la investigación en el **campo de acción:** Proceso de pruebas para la evaluación de la calidad del software del Generador Dinámico de Reportes.

Dada la problemática planteada y con el propósito de dar solución al problema propuesto se plantea como **objetivo general:** Desarrollar el proceso de pruebas en varios niveles para la evaluación de la calidad del Generador Dinámico de Reportes en su versión 1.7.0**.** Contando con los **objetivos específicos** siguientes**:**

- Diseñar las pruebas de software para el sistema integrado.
- Aplicar las pruebas de software.
- Analizar los resultados del proceso de pruebas desarrollado.

Entre las **tareas de la investigación científica** que se proponen desarrollar para el cumplimiento de los objetivos se señalan:

- Análisis de las pruebas de software y herramientas para automatizar las pruebas.

- Selección de los niveles, técnicas y métodos de pruebas a aplicar durante el proceso de evaluación.
- Definición de las herramientas para la automatización del proceso de pruebas.
- Realización del plan de pruebas.
- Realización de pruebas exploratorias.
- Diseño de los casos de pruebas.
- Realización de pruebas:
  - De unidad a las funcionalidades críticas para el desempeño del sistema.
  - De volumen.
  - De instalación.
  - De carga y estrés.
- Revisión de la documentación del software a evaluar.
- Análisis y evaluación de los resultados obtenidos.
- Realización de propuestas de mejoras según los resultados obtenidos.

El presente documento está estructurado en tres capítulos.

Capítulo 1: Fundamentación teórica.

En el presente capítulo se exponen algunos conceptos y aspectos de interés de la calidad de software. Se presentan aspectos relevantes para el desarrollo de las pruebas, tales como los niveles de pruebas, técnicas y métodos, de los cuales se establece una selección, en la que se obtienen las más apropiadas a desarrollar tras el proceso que se desea realizar con el desarrollo del trabajo. Además se realiza un análisis de las herramientas existentes para el desarrollo de algunas de las pruebas a utilizar.

Capítulo 2: Diseño de casos de pruebas.

En el capítulo se presenta una descripción del sistema al cuál se le aplicarán las pruebas, en el que a partir del desarrollo del plan de pruebas se presentarán los diseños de las pruebas que se aplicarán al sistema en cuestión.

Capítulo 3: Aplicación de las pruebas y análisis de los resultados.

Tras el proceso de aplicación de las pruebas al sistema, se hace necesario la evaluación de los resultados, teniendo en cuenta los atributos de calidad. Es por ello que en este capítulo se ejecutarán el conjunto de pruebas, de los que se mostrarán y analizarán los resultados obtenidos, en la que para dar culminación al trabajo, se realizará un conjunto de propuestas, que permitirán mejorar el desarrollo de posteriores versiones del software. .

CAPÍTULO 1

Fundamentación Teórica

Introducción

El presente capítulo tiene como objetivo exponer los fundamentos teóricos generales que sirven de punto de partida para determinar la calidad para el producto GDR a partir de la ejecución de pruebas que permitan establecer una evaluación. En su contenido se ofrecen algunos conceptos que permitirán una mejor comprensión del trabajo, entre los que se encuentra aspectos relacionados con la calidad de software, así como el estudio de los elementos que permitan llevar a cabo la realización de las pruebas de software, estos son: sus objetivos, estrategias, niveles, técnicas y métodos, así como herramientas para automatizar algunas de las pruebas a aplicar. De cada uno de ellos de establece una selección en la que se distinguen las más favorables a aplicar, tras el proceso en que se encuentra enmarcado el desarrollo del trabajo.

1.1. Calidad de software

El gran impacto que ha alcanzado la era computacional, y los recursos que se han implementado para el desarrollo informático, principalmente el papel primordial que han alcanzado los software para muchas organizaciones, en las que su devenir y supervivencia a partir de la amplia competitividad existente en el mercado mundial radica en la creación de elementos de alta competitividad, con elevados niveles de calidad, hace que este recurso se convierta en un importante eslabón para la sociedad. Es por ello que se hace necesario que el surgimiento de la calidad sea un factor importante en la sociedad de la información.

Muchos han sido los que han enfocado la calidad del software desde diferentes puntos de vista. Según el fundador de la consultora en gestión de la calidad más grande y experimentada del mundo Philip Crosby, define la calidad de software como: la conformidad con los requisitos que la propia compañía ha establecido para sus productos basados directamente en las necesidades de sus clientes“[5]

Para la ISO 8402 la calidad de software es: “El conjunto de características de un producto o servicio que le confieren su aptitud para satisfacer necesidades expresadas e implícitas”. [2]

Por su parte El Instituto de Ingenieros Eléctricos y Electrónicos ([IEEE.Std.610-1990]) conceptualiza a la calidad de software como: “El grado con el que un sistema componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”. [6]

Mientras que en 1988 R.S. Pressman plantea que la calidad de software es: “La concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario”. [7]

Al valorar las diferentes definiciones que se establecen alrededor del tema de calidad de software, es posible determinar que en cada una de estas, el término de los requisitos es abordado en tales enunciaciones. La no concordancia de los elementos definidos en los requisitos disminuye el nivel de calidad de un producto. El establecimiento de los requisitos, y cumplimiento con lo establecido en cada uno de ellos, es la principal vía para establecer un software con calidad; y de esta forma lograr la satisfacción del cliente. La universidad ha puesto en práctica conjunto de mejoras que posibilitan el mejoramiento sustancial de los productos, pero aún carece de fundamentos y estrategias que intensificarían su calidad.

1.2. Modelo de calidad

En el año 1991 la ISO(International Organization for Standardization) publicó su modelo de calidad para la evaluación del producto de software(ISO 9126:1991), que fue extendiendo con revisiones hasta el 2004, dando lugar a la actual norma ISO/IEC 9126” Software Product Quality”. La norma ISO/IEC 9126 propone un conjunto de características, subcaracterísticas y atributos para descomponer la calidad de un producto de software. Propone seis propiedades (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad), que se dividen en subcategorías [8], como se muestra en la Figura 1.1.

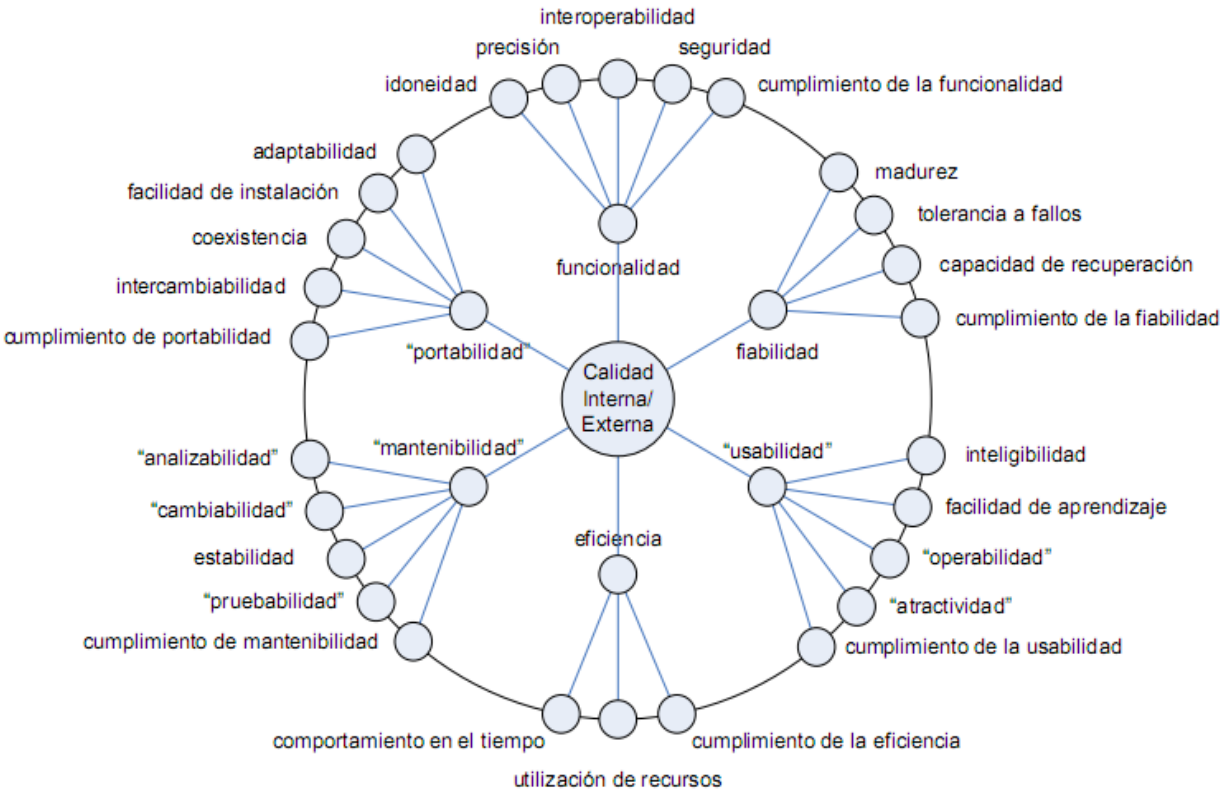


Figura 1: Modelo de Calidad ISO

1. Funcionalidad: En este grupo se establece una serie de atributos que permiten calificar la capacidad de un producto software de satisfacer los requisitos funcionales prescriptos y las necesidades implícitas de los usuarios. La pregunta central que atiende a esta característica es: ¿Las funciones y propiedades satisfacen las necesidades explícitas e implícitas; esto es, el qué?
2. Fiabilidad: Aquí se agrupan un conjunto de atributos que se refieren a la capacidad de un producto software de mantener su nivel de desempeño, bajo condiciones establecidas, por un periodo de tiempo. La pregunta central que atiende a esta característica es: ¿Puede mantener el nivel de rendimiento, bajo ciertas condiciones y por cierto tiempo?
3. Usabilidad: Consiste de un conjunto de atributos que permiten evaluar la capacidad de un producto software de ser comprendido, aprendido, usado, atractivo y conforme con las reglamentaciones y guías de usabilidad. La pregunta central que atiende a esta característica es: ¿El software es fácil de usar y de aprender?
4. Eficiencia: Esta característica permite evaluar la capacidad de un producto software de proporcionar un rendimiento apropiado, de acuerdo a la cantidad de recursos usados bajo condiciones

establecidas. La pregunta central que atiende a esta característica es: ¿Es rápido y minimalista en cuanto al uso de recursos?

5. Mantenibilidad: Se refiere a los atributos que permiten medir la capacidad de un producto software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, en los requisitos o en las especificaciones funcionales. La pregunta central que atiende a esta característica es: ¿Es fácil de modificar y verificar?

6. Portabilidad: En este caso, se refiere a la habilidad del software de ser transferido de un ambiente a otro. Nota: El ambiente puede ser organizacional, de software o de hardware. La pregunta central que atiende a esta característica es: ¿Es fácil de transferir de un ambiente a otro? [9]

1.3. Pruebas de software

La importancia del desarrollo de las pruebas de software ha aumentado considerablemente en los últimos años. Estas son consideradas un importante eslabón en la que van a constituir un conjunto de herramientas, técnicas y métodos que permiten verificar y revelar la calidad de un producto de software mostrando confianza en los sistemas antes de su puesta en funcionamiento [10], evitando así errores en la operación, y por tanto costes imprevistos, provocando la satisfacción de los usuarios finales y un incremento en la confianza de los responsables. Con el desarrollo de estas es posible obtener una evaluación, posibilitando determinar en qué condiciones se encuentra el producto, en la que paso a paso se determinará si este cumple con las expectativas del cliente. Es por ello que hay que integrar las pruebas dentro del flujo de desarrollo software como una parte inseparable, y no como una parte opcional.

Concretamente se puede definir las pruebas de software como:

Cualquier intento de demostrar que el software tiene propiedades por debajo de la calidad requerida [11]

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. [12]

Un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. [13]

Son muchos los criterios que se desprenden alrededor de las pruebas de software, en cada una de las definiciones dadas es posible determinar que se trata a las pruebas de software como la actividad que permite dar un enfoque de si un producto presenta la calidad requerida. El establecimiento y ejecución

de las mismas constituye un eslabón importante para la identificación de fallas del sistema, en la que permite verificar y validar cada una de las funciones, determinando el grado en que se encuentra cada uno de los atributos de calidad a través de la evaluación del mismo. Un factor importante que hay que tener en cuenta, es el manejo de cada una de las pruebas a realizar, el manejo de estas debe de encontrarse determinada por un proceso bien definido y lo bastante explícito para el mejor desenvolvimiento de las mismas y que permitan el logro del objetivo propuesto.

1.4.1. Objetivos de las pruebas de software

Dentro de los objetivos principales que se persiguen con las pruebas de software se encuentran los siguientes:

- Medir el grado en que el software cumple con los requisitos establecidos por el cliente.
  - ✓ Verificar que el software trabaje como fue diseñado.
  - ✓ Validar y probar los requisitos que debe cumplir el software.
  - ✓ Validar que los requisitos fueron implementados correctamente.
- Encontrar defectos en el software.
  - ✓ Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
  - ✓ Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.
  - ✓ Una prueba tiene éxito si descubre un defecto. Una prueba fracasa si hay defectos pero no los descubre.
- Asegurar la Calidad del Software.
  - ✓ Ofrecer un producto altamente confiable.

1.4. Evaluación del producto

La evaluación del producto es la actividad final de la estrategia de prueba trazada. Como su nombre indica, es una evaluación general emitida acerca del producto, teniendo en cuenta ciertos atributos de calidad que no deben faltar en ningún software. [11]

Para ello se ha establecido como estrategia el establecimiento de pruebas con el fin de medir los diferentes atributos de calidad, en el que a través de un análisis de los resultados obtenidos, se realizará la evaluación del sistema. Además se aplicará un conjunto de listas de chequeo que permitirán desglosar los diferentes indicadores del sistema, dándole una clasificación en dependencia de su comportamiento. El establecimiento de encuestas que permitirán determinar el grado de asimilación del producto por parte de los usuarios.

Por los elementos que definen al sistema se ha determinado que las características fundamentales a medir estarán dadas por los atributos de eficiencia, usabilidad, portabilidad y fiabilidad.
La fiabilidad está dada por la veracidad a la hora de obtener un reporte. Por su parte por la variedad de usuarios que pueden disponer de esta aplicación se hace necesaria la comprobación de la usabilidad de este. Por la velocidad de los tiempo de respuesta ante diferentes volúmenes de datos y por la diferencia de entornos en el que puede ser desplegado se establece rl atributo de eficiencia, así como la interacción de los gestores de Base de datos se establecen los atributos de rendimiento y portabilidad. Por la facilidad de instalación y el ambiente en que se despliega el sistema se encuentra el atributo de portabilidad.

1.5. Estrategia de pruebas

La estrategia de pruebas busca que el producto de software que se está construyendo o modificando, reúna los requerimientos de lógica del negocio que el cliente ha pedido realizar mediante el debido contrato de desarrollo de software. Son las líneas guía del equipo de pruebas de un proyecto de desarrollo de software. Reúne las ideas más representativas del proceso de pruebas que se llevará a cabo.
Una estrategia de prueba del software debe ser lo suficientemente flexible como para promover un enfoque personalizado. Al mismo tiempo, debe ser lo adecuadamente rígido como para promover una planeación razonable y un seguimiento administrativo del avance del proyecto.
Según Pressman: “Una 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”. [7]

1.6. Niveles de prueba

La prueba es aplicada con diferentes objetivos y en disímiles escenarios o niveles de prueba durante todo el ciclo de desarrollo del software. Estas deben fluir de forma orgánica y ascendente, logrando ir avanzando progresivamente comenzando por los componentes más simples y pequeños hasta probar el sistema en su conjunto. Es considerado que deben realizarse todas para garantizar la calidad de forma continua e incremental.
Se distinguen los siguientes niveles de pruebas:

- **Pruebas de unidad**

La prueba de unidad es la primera fase de las pruebas dinámicas y se realizan sobre cada módulo del software de manera independiente. El objetivo es comprobar que el módulo, entendido como una

unidad funcional de un programa independiente, está correctamente codificado. En estas pruebas cada módulo será probado por separado y lo hará, generalmente, la persona que lo creó.

Tanto las pruebas de caja blanca como las de caja negra han de aplicarse para probar de la manera más completa posible un módulo. Las pruebas de caja negra (los casos de prueba) se pueden especificar antes de que módulo sea programado, no así las pruebas de caja blanca.

Al tratar software orientado a objeto (OO) cambia el concepto de unidad. El encapsulamiento dirige la definición de clases objetos. Esto significa que cada clase e instancia de clase (objeto) empaqueta los atributos (datos) y las operaciones (también conocidas como métodos o servicios) que manipulan estos datos.

Por lo tanto, en vez de módulos individuales, la menor unidad a probar es la clase u objeto encapsulado. Una clase puede contener un cierto número de operaciones, y una operación particular puede existir como parte de un número de clases diferentes. Por tanto, el significado de prueba de unidad cambia ampliamente frente al concepto general visto antes. [15]

- **Pruebas de integración**

Aún cuando los módulos de un programa funcionen bien por separado es necesario probarlos conjuntamente: un módulo puede tener un efecto adverso o inadvertido sobre otro módulo; las sub-funciones, cuando se combinan, pueden no producir la función principal deseada; la imprecisión aceptada individuamente puede crecer hasta niveles inaceptables al combinar los módulos; los datos pueden perderse o malinterpretarse entre interfaces, etc.

Por lo tanto, es necesario probar el software ensamblando todos los módulos probados previamente. Y este constituye el objetivo principal de la pruebas de integración.

A menudo hay una tendencia a intentar una integración no incremental; es decir, a combinar todos los módulos y probar todo el programa en su conjunto. El resultado puede ser un poco caótico con un gran conjunto de fallos y la consiguiente dificultad para identificar el módulo que los provocó.

En contra, se puede aplicar la integración incremental en la que el programa se prueba en pequeñas porciones en las que los fallos son más fáciles de detectar. Existen dos tipos de integración incremental, la denominada ascendente y descendente.

Ascendente: Se empieza con los módulos de nivel inferior, y se verifica que los módulos de nivel inferior llaman a los de nivel superior de manera correcta, con los parámetros correctos.

Descendente: Se empieza con los módulos de nivel superior, y se verifica que los módulos de nivel superior llaman a los de nivel inferior de manera correcta, con los parámetros correctos.

Debido a que el software OO no tiene una estructura de control jerárquica, las estrategias convencionales de integración ascendente y descendente poseen un significado poco relevante en este contexto.

Generalmente se pueden encontrar dos estrategias diferentes de pruebas de integración en sistemas OO. La primera, prueba basada en hilos que integra el conjunto de clases necesario para responder a una entrada o evento del sistema. Cada hilo se integra y prueba individualmente. El segundo enfoque para la integración, es la prueba basada en el uso. Esta prueba comienza en la fase de construcción del sistema, integrando y probando aquellas clases (llamadas clases independientes) que usan muy pocas de estas. Después de probar las clases independientes, se comprueba la próxima capa de clases, llamadas clases dependientes, que usan las clases independientes. Esta secuencia de capas de pruebas de clases dependientes continúa hasta construir el sistema por completo. [15]

❖ **Pruebas del sistema**

Este tipo de pruebas tiene como propósito ejercitar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema (hardware, otro software, etc.) y que realizan las funciones adecuadas. Concretamente se debe comprobar que:

- Se cumplen los requisitos funcionales establecidos.
- El funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
- La adecuación de la documentación de usuario.
- Rendimiento y respuesta en condiciones límite y de sobrecarga.

Para la generación de casos de prueba de sistema se utilizan técnicas de caja negra.

Este tipo de pruebas se suelen hacer inicialmente en el entorno del desarrollador, denominadas Pruebas Alfa, y seguidamente en el entorno del cliente denominadas Pruebas Beta.

En el nivel de prueba del sistema, los detalles de las conexiones entre clases no afectan. El software debe integrarse con los componentes hardwares correspondientes y se ha de comprobar el funcionamiento del sistema completo acorde a los requisitos. Como en el caso del software convencional, la validación del software OO se centra en las acciones visibles del usuario y las salidas del sistema reconocibles por éste. Para asistir en la determinación de casos de prueba de sistema, el ejecutor de la prueba debe basarse en los casos de uso que forman parte del modelo de análisis.

Los métodos convencionales de prueba de caja negra, pueden usarse para dirigir estas pruebas. [15]

❖ **Pruebas de aceptación**

A la hora de realizar estas pruebas, el producto está listo para implantarse en el entorno del cliente. El usuario debe ser el que realice las pruebas, ayudado por personas del equipo de pruebas, siendo deseable, que sea el mismo usuario quien aporte los casos de prueba.

Estas pruebas se caracterizan por:

- Participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas.
- Están enfocadas a probar los requisitos de usuario, o mejor dicho a demostrar que no se cumplen los requisitos, los criterios de aceptación o el contrato. Si no se consigue demostrar esto el cliente deberá aceptar el producto
- Corresponden a la fase final del proceso de desarrollo de software.

Es muy recomendable que las pruebas de aceptación se realicen en el entorno en que se va a explotar el sistema (incluido el personal que lo maneje). En caso de un producto de interés general, se realizan pruebas con varios usuarios que reportarán sus valoraciones sobre el producto.

Para la generación de casos de prueba de aceptación se utilizan técnicas de caja negra.

La prueba de aceptación en un sistema OO es semejante a la prueba de aceptación en un software tradicional. El motivo es que el objetivo de este tipo de prueba es comprobar si el cliente está satisfecho con el producto desarrollado y si este producto cumple con sus expectativas, en términos de los errores que genera y de la funcionalidad que suministra. Al igual que las pruebas convencionales serán los clientes quienes realicen estas pruebas y suministren los casos de prueba correspondientes. [15]

❖ **Pruebas de regresión**

La regresión consiste en la repetición selectiva de pruebas para detectar fallos introducidos durante la modificación de un sistema o componente de un sistema. Se efectuarán para comprobar que los cambios no han originado efectos adversos no intencionados o que se siguen cumpliendo los requisitos especificados.

En las pruebas de regresión hay que:

- Probar íntegramente los módulos que se han cambiado.
- Decidir las pruebas a efectuar para los módulos que no han cambiado y que han sido afectados por los cambios producidos.

Este tipo de pruebas ha de realizarse, tanto durante el desarrollo cuando se produzcan cambios en el software, como durante el mantenimiento. [15]

1.7. Niveles de pruebas seleccionados

Los niveles de pruebas que se han seleccionado para llevar a cabo el proceso de evaluación de la calidad del GDR son:

- Pruebas de unidad.
- Pruebas de sistema.

En el proceso en que se encuentra enmarcado este trabajo, se tomó que el desarrollo de las pruebas de unidad permitirá valorar la eficiencia y complejidad del código. Por su parte las pruebas de sistema, valorarán el sistema como un todo. Otras de las razones por la cual se ha seleccionado este nivel de prueba están dadas por el conjunto de técnicas a desarrollar, las cuales van a estar determinadas por la comprobación del sistema de forma general.

1.8. Métodos de pruebas

Los avances en lenguajes de programación y la gran variedad de tipos de software, las pruebas de caja negra y caja blanca han tomado un lugar muy importante en el desarrollo de sistemas de cualquier tipo, tanto que sin dichas pruebas un sistema desarrollado carecería de garantías y credibilidad en sus funciones. Estos métodos proporcionan directrices sistemáticas para pruebas de diseño que 1) comprueben la lógica interna y las interfaces de todo componente del software y 2) comprueben los dominios de entrada y salida del programa para descubrir errores en su función, comportamiento y desempeño. [16]

❖ **Método de caja blanca**

A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa. Es decir, permite comprobar los caminos lógicos del software y examinar el estado del programa en varios puntos para determinar si el estado es real y coincide con el esperado.

Estas pruebas deben garantizar como mínimo que:

- Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Se ejerciten las estructuras internas de datos para asegurar su validez.[11]

➢ **La prueba del camino básico**

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

un conjunto básico de caminos de ejecución. Se basa en construir un caso de prueba por camino básico que se encuentre en el grafo de programa asociado al método de la clase que se desea someter a pruebas.

La idea de esta técnica es derivar casos de prueba a partir de un conjunto dado de caminos independientes, que no son más que aquellos que introduce al menos una sentencia de procesamiento que no estaba considerada. Para obtener el conjunto de caminos independientes se construirá el grafo de flujo asociado y se calculará su complejidad ciclomática. [5]

- **Prueba de condición**

Método que ejercita las condiciones lógicas contenidas en un módulo del programa. Una condición simple es una variable lógica o una expresión relacional. Esta prueba se concentra en la prueba de cada condición del programa para asegurar que no contiene errores. [5]

- **Prueba de flujo de datos**

El método de flujo de datos selecciona caminos de pruebas de un programa de acuerdo con la ubicación de las definiciones y el uso de las variables del programa. [5]

- **Prueba de bucle**

La prueba de bucle se concentra exclusivamente en la validez de la construcción de bucles. Se pueden definir cuatro tipos de bucles: simples, anidados, concatenados, no estructurados. [5]

❖ **Método de caja negra**

También conocidas como pruebas de comportamiento. Las pruebas de caja negra se centran fundamentalmente en lo que se espera de un sistema. 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.

Estas pruebas están especialmente indicadas en aquellos sistemas que poseen una interfaz que interactué con el usuario. Se apoyan en la especificación de requisitos funcionales de dicha aplicación verificando que estos se cumplan.

El problema con las pruebas de Caja Negra no suele estar en el número de funciones proporcionadas por el sistema; sino en los datos que se le pasan a estas funciones. El conjunto de datos posibles suele ser muy amplio así que la prueba se torna tediosa.

Lo fundamental en este tipo de pruebas es encontrar el subconjunto de todas las entradas posibles del programa esto es muy complejo, para conseguirlo se siguen diferentes criterios [11]:

- Métodos de prueba basados en grafos.

- Métodos de prueba partición de equivalencia.
- Métodos de prueba de análisis de valores límite.
- Métodos de prueba de la tabla ortogonal.
- Métodos de prueba adivinando el error.

➢ **Técnica de prueba basada en grafos**

En esta técnica se debe entender los objetos que se modelan en el software y las relaciones que conectan a estos, tales como objetos de datos, objetos de programa como módulos o colecciones de sentencias del lenguaje de programación. Una vez que se ha llevado a cabo esta operación, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas.

En este método:

- Se crea un grafo de objetos importantes y sus relaciones.
- Se diseña una serie de pruebas que cubran el grafo de manera que se ejerciten todos los objetos y sus relaciones para descubrir errores. [17]

➢ **Partición equivalente**

La técnica de prueba partición equivalente 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 mediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes detectar el error genérico. [17]

En otras palabras, este método intenta dividir el dominio de entrada de un programa en un número finito de clases de equivalencia. De tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada clase es equivalente, a una prueba realizada con cualquier otro valor de dicha clase. 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.

Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Típicamente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. [17]

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. Se toma un riesgo porque se escoge

no probar todo. Así que se necesita tener mucho cuidado al escoger las clases.
Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada. Típicamente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.
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. Se toma un riesgo porque se escoge no probar todo. Así que se necesita tener mucho cuidado al escoger las clases. [17]

- **Análisis de valores límite**

El análisis de valores límite (AVL) es una técnica de diseño de casos de prueba que completa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida.
Dentro del AVL se encuentran las condiciones sublímites o condiciones límites internas. Las condiciones límite normales son las más obvias de descubrir. Estas son definidas en la especificación o son evidentes al momento de utilizar el software. Algunos límites, sin embargo, son internos al software, no son necesariamente aparentes al usuario final, pero aún así deben ser probadas por el probador. Estas son conocidas como condiciones sublímites o condiciones límite internas. [17]

- **Prueba de la tabla ortogonal**

La prueba de la tabla ortogonal puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para posibilitar pruebas exhaustivas ya que hay aplicaciones donde el número de parámetros de entrada es pequeño y los valores de cada uno de los parámetros están claramente delimitados. Cuando estos números son muy pequeños, es posible considerar cada permutación de entrada y comprobar exhaustivamente el proceso del dominio de entrada. En cualquier caso, cuando el número de valores de entrada crece y el número de valores diferentes para cada elemento de dato se incrementa, la prueba exhaustiva se hace impracticable. [17]

De lo dicho anteriormente es que se puede decir que el método de prueba de la tabla ortogonal es particularmente útil al encontrar errores asociados con fallos localizados y permite proporcionar una buena cobertura de pruebas con menos casos de prueba que en la estrategia exhaustiva.

- **Adivinando el error**

Adivinando el error es una técnica que dado un programa particular, se conjetura, por la intuición y la experiencia, ciertos tipos probables de errores y se escriben casos de prueba para exponer esos errores. La idea básica es enumerar una lista de errores posibles o de situaciones propensas a error y después escribir los casos de prueba basados en la lista. [17]

1.9. Método de prueba seleccionado

Para el desarrollo de este trabajo será necesaria la utilización de las pruebas de caja blanca, así como las pruebas de caja negra. Una por su capacidad de interacción con el código, en la que a partir de la técnica de prueba de camino básico permitirá obtener una medida de la complejidad lógica de un diseño. Por otra parte las pruebas de caja negra es muy efectiva a la hora de probar la valides de las condiciones de entrada. Dentro de este método de prueba se utilizará la técnica de partición de equivalencia ya que se dirige a la definición de casos de prueba que descubren clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

1.10. Técnicas de pruebas

Tras la aplicación de las pruebas a un software se hace necesaria la utilización de varias técnicas que facilitan comprobación del funcionamiento del mismo. Existen diferentes técnicas de prueba de software, en las que cada una de ellas se desglosa en diferentes atributos. El conjunto de técnicas están dadas por:

- Pruebas de Funcionalidad.
- Pruebas de Usabilidad.
- Pruebas de Fiabilidad.
- Pruebas de Rendimiento.
- Pruebas de Soportabilidad.

- **Pruebas de funcionalidad.**

La prueba de funcionalidad se enfoca en requerimientos para verificar que se corresponden directamente a casos de uso o funciones y reglas del negocio. Los objetivos de estas pruebas son verificar la aceptación de los datos, el proceso, la recuperación y la implementación correcta de las reglas del negocio. Esta técnica de prueba se basa en el método de caja negra, y consiste en verificar la aplicación y sus procesos interactuando con la aplicación por medio de la interfaz de usuario y analizar los resultados obtenidos. El conjunto de pruebas pertenecientes a este grupo son:

- Prueba funcional.

- Prueba de seguridad.
- Prueba de volumen.

➢ **Prueba funcional**

La prueba funcional tiene como objetivo asegurar el trabajo apropiado de los requisitos funcionales; incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

Verifica el procesamiento, recuperación e implementación adecuada de las reglas del negocio y la apropiada aceptación de datos utilizando el método de caja negra.

➢ **Prueba de seguridad**

La prueba de seguridad y control de acceso se enfoca en dos áreas de seguridad: seguridad en el ámbito de aplicación, incluyendo el acceso a los datos y a las funciones de negocios y seguridad en el ámbito de sistema, incluyendo conexión, o acceso remoto al sistema.

La seguridad en el ámbito de aplicación consiste en que los usuarios finales están restringidos a funciones o casos de uso específicos o limitados, en los datos que están disponibles para ellos; por lo que tiene como objetivo verificar que un actor pueda acceder solo a las funciones o datos para los cuales su tipo de usuario tiene permiso.

La seguridad en el ámbito de sistema asegura que, solo los usuarios finales con derecho a acceder al sistema son capaces de acceder a las aplicaciones a través de los puntos de ingresos apropiados, por lo que tiene como objetivo verificar que solo los actores con acceso al sistema y a las aplicaciones, puedan acceder a ellos.

➢ **Prueba de volumen**

La prueba de volumen somete al software a grandes cantidades de datos para determinar si se alcanzan límites que causen la falla del software. La prueba de volumen identifica la carga máxima continua que puede manejar el software a prueba en un período dado.

Tiene como objetivo verificar que el software funciona correctamente con volúmenes de datos grandes:

- Máximo número de clientes conectados, o simulados, todos realizando la misma operación por un período de tiempo extenso.
- Máximo tamaño de base de datos y múltiples consultas ejecutadas simultáneamente.[18]

❖ **Pruebas de usabilidad**

La prueba de usabilidad consiste en realizar pruebas efectuadas con usuarios, con el objetivo de determinar si la organización de los contenidos y las funcionalidades que se ofrecen desde el sitio web son entendidas y utilizadas por los usuarios de manera simple y directa. Las pruebas tradicionales son:

- Prueba inicial
- Prueba de boceto web
- Prueba de interfaz de usuario

➢ **Prueba inicial**

La prueba inicial se hace para ver cómo funciona la organización de contenidos y elementos iniciales de diseño (botones, interfaces). El material con que se prueba es una imagen dibujada del sitio web.

➢ **Prueba de estructura web**

La prueba de estructura web tiene como objetivo entender la navegación y verificar si se pueden cumplir tareas y si el usuario entiende todos los elementos que se le ofrecen. El material con que se prueba es una maqueta web semifuncional.

➢ **Prueba de interfaz de usuario**

La prueba interfaz de usuario verifica que la interfaz del usuario proporcione al usuario el acceso y navegación a través de las funciones apropiadas. Además asegura que los objetos presentes en la interfaz de usuario se muestren como se espera y conforme a los estándares establecidos por la empresa o de la industria.

Esta prueba tiene como técnica crear o modificar pruebas para cada ventana verificando la navegación y los estados de los objetos para cada ventana de la aplicación y cada objeto dentro de la ventana. [19]

❖ **Pruebas de fiabilidad**

La fiabilidad de un software es la probabilidad de que un programa realice su objetivo satisfactoriamente, sin fallos, en un determinado periodo de tiempo y en un entorno concreto. Las pruebas de fiabilidad están constituidas por:

- Prueba de integridad.
- Prueba de estructura.
- Prueba de estrés.
- Prueba de falla y recuperación.

- **Prueba de integridad**

La prueba de integridad es la prueba que se le realiza a un software para comprobar la resistencia a los fallos. Debe estar conforme técnicamente en cuanto a lenguaje, sintaxis y uso de recursos de implementación.

- **Prueba de estructura**

Utilizan el método de Caja Blanca, son también conocidas como "pruebas basadas en código", donde se enfocan en probar cada una de las estructuras de código, para que su comportamiento sea el esperado. Son las pruebas donde se conoce la estructura interna del componente a probar, y se efectúa una prueba sobre dicha estructura. En el caso de una aplicación web también se revisa la estructura interna de los links y otros elementos.

- **Prueba de estrés**

La prueba de estrés es un tipo de prueba de performance implementada y ejecutada para encontrar errores cuando hay pocos recursos o cuando hay competencia por recursos. Poca memoria o poco espacio de disco pueden revelar fallas en el software que no aparecen bajo condiciones normales de cantidad de recursos. Otras fallas pueden resultar al competir por recursos compartidos como bloqueos de bases de datos o ancho de banda de red. La prueba de estrés también puede usarse para identificar el trabajo máximo que el software puede manejar.
Tiene como técnica usar las pruebas desarrolladas para performance y prueba de carga. Para probar recursos limitados, las pruebas se deben ejecutar en una sola máquina, y se debe reducir o limitar la memoria en el servidor. Para las pruebas de estrés restantes, deben usarse múltiples clientes, cualquiera que ejecute las mismas pruebas o pruebas complementarias para producir el peor caso de volumen de operaciones.
La prueba de estrés se puede ver reflejada cuando se quiere medir la resistencia del sistema, aumentando la carga simultánea de trabajo con la misma cantidad de recursos disponibles.

- **Prueba de falla y recuperación**

Las pruebas de fallas y recuperación aseguran que el software puede recuperarse de fallas de hardware, software o mal funcionamiento de la red sin pérdida de datos o de integridad de los datos. [8] La técnica es usar pruebas creadas para probar funcionalidad y ciclos de negocio para crear una serie de operaciones. En la prueba se incluyen los siguientes tipos de condiciones:

- Interrupción de energía al cliente.
- Interrupción de energía al servidor.
- Interrupción de comunicaciones mediante los servidores de la red.

- Interrupción de comunicación o pérdida de energía de los discos del servidor o con los controladores.
- Ciclos incompletos (procesos de filtro de datos interrumpidos, procesos de sincronización de datos interrumpidos).
- Punteros a la base de datos o claves inválidos.
- Elementos de datos en la base de datos inválidos o corruptos.[19]

❖ **Pruebas de rendimiento**

Las pruebas de rendimiento son las pruebas que se realizan, desde una perspectiva, para determinar lo rápido que realiza una tarea de un sistema en condiciones particulares de trabajo. También puede servir para verificar otros atributos de calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos. Existen diferentes tipos de pruebas de rendimiento como son:

- Prueba de contención.
- Prueba de carga.
- Prueba profile.

➢ **Prueba de contención**

Las pruebas de contención son aquellas que tienen como objetivo verificar que el elemento que se prueba maneja adecuadamente cuando muchos actores solicitan el mismo recurso, por ejemplo: registro de datos, memoria, entre otros.

➢ **Prueba de carga**

La prueba de carga somete los objetos a verificar a diferentes cargas de trabajo para medir y evaluar los comportamientos de performance y la habilidad de los objetos de continuar funcionando apropiadamente bajo diferentes cargas de trabajo. El objetivo es determinar y asegurar que el sistema funciona apropiadamente en circunstancias de máxima carga de trabajo esperada. Además evaluar las características de performance, como tiempos de respuesta, tiempos de transacciones y otros elementos sensitivos al tiempo.

La técnica que utiliza es el uso de pruebas desarrolladas para funciones o ciclos de negocios y modificar archivos de datos para aumentar el número de transacciones, o las pruebas para aumentar la cantidad de ocurrencia de transacciones.

➢ **Prueba profile**

En esta prueba se miden y evalúan los tiempos de respuesta, los tiempos de transacción y otros requerimientos sensitivos al tiempo. El objetivo de la prueba es verificar que se logren los

requerimientos de performance. Esta prueba tiene como técnica usar procedimientos de prueba desarrollados para verificar funciones o ciclos de negocio, modificar archivos de datos para aumentar el número de transacciones o los procedimientos de prueba para aumentar el número de iteraciones de ocurrencia de transacciones. Las pruebas se deben ejecutar en una y se debe repetir con múltiples usuarios. [19]

❖ **Pruebas de soportabilidad**

Las pruebas de soportabilidad son otras de las pruebas que se le pueden realizar al software para solucionar problemas de una aplicación. Existen diferentes tipos de pruebas de soportabilidad, las cuales son: prueba de configuración y prueba de instalación.

➢ **Prueba de configuración**

La prueba de configuración verifica el funcionamiento del software con diferentes configuraciones de software y hardware. Su técnica es usar las pruebas de funcionalidad.

- Abrir y cerrar varias sesiones de software que no son objeto de prueba, como parte de la prueba o antes de comenzar la prueba.
- Ejecutar operaciones seleccionadas para simular la interacción del actor con el software objeto de prueba y con el software que no es objeto de prueba.
- Repetir los procedimientos anteriores minimizando la memoria convencional disponible en la máquina cliente.

➢ **Prueba de instalación**

La prueba de instalación tiene dos propósitos. Uno es asegurar que el software puede ser instalado en diferentes condiciones (como una nueva instalación, una actualización, y una instalación completa o personalizada) bajo condiciones normales y anormales. Condiciones anormales pueden ser insuficiente espacio en disco, falta de privilegios para crear directorios, etc. El otro propósito es verificar que, una vez instalado, el software opera correctamente. Esto significa normalmente ejecutar un conjunto de pruebas que fueron desarrolladas para prueba de funcionalidad.

Tiene como técnica:

- Validar la condición de la máquina destino (nueva, nunca instalado, misma versión, versión anterior ya instalada).
- Realizar la instalación.
- Ejecutar un conjunto de pruebas funcionales ya implementadas para la prueba de funcionalidad.[19]

1.11. Técnicas de prueba seleccionadas

El conjunto de pruebas seleccionadas se compone fundamentalmente por:

- Pruebas de Estructura.
- Pruebas de Volumen.
- Pruebas de Carga y Estrés.
- Pruebas de Instalación.

Se aplicarán las pruebas de Estructura para determinar la complejidad del código, las pruebas de volumen permitirán analizar el comportamiento de la aplicación y de la BD con volúmenes de datos almacenados similares a los esperados en la explotación real del sistema. Por su parte las pruebas de carga y estrés determinan la comprobación del rendimiento del sistema soportando la cantidad máxima de usuarios conectados y su comportamiento al aumentar esta carga con los mismos recursos disponibles. Por último las pruebas de instalación determinará el comportamiento del sistema en diferentes entornos.

1.12. Herramientas para automatizar las pruebas

En la actualidad el uso de las herramientas se ha hecho tan imprescindible , para automatizar muchos procesos, es por ellos que se han desarrollado una serie de frameworks para automatizar el desarrollo de las pruebas de software, estos tienen como fin acelerar el proceso de pruebas, teniendo grandes ventajas frente a realizar las mismas manualmente. Se hizo imprescindible la investigación del uso de herramientas para mejorar la calidad y eficiencia del flujo de pruebas. Durante la investigación se dará a conocer algunas herramientas que permiten el desarrollo de las pruebas, específicamente el desarrollo de las pruebas de carga y estrés.

1.12.1. Herramientas para pruebas de carga y estrés.

El desarrollo de las pruebas de carga y estrés, tras su difícil realización de dichas pruebas manualmente, se hace necesario el uso de una herramienta capaz de automatizarlas, y lograr que se mida el rendimiento del sistema de la mejor manera posible. El establecimiento de la herramienta manualmente implicaría el uso de gran cantidad de recursos para lograr un comportamiento similar en el sistema y verificar que el desempeño es el esperado. Resulta extremadamente difícil simular de forma manual el comportamiento de 300 personas accediendo simultáneamente al sistema en cuestión. Por estas razones se realizó una investigación de algunas herramientas encargadas de automatizar este tipo de pruebas.

❖ **QALoad**

Herramienta que te permite ejecutar diferentes tipos de pruebas de desempeño a tus aplicativos (Web, SAP, Cliente Servidor, Citrix, .NET, Java, entre otros), simulando usuarios virtuales que acceden al servidor en concurrencia.

QALoad también posee la habilidad para simular una aplicación Cliente/Servidor sin comprometer al usuario final o al equipo de usuarios finales. Lleva a cabo la prueba de carga capturando el middleware y creando usuarios virtuales que ejecuta y simula usuarios físicos. QALoad genera gráficos y reportes que le ayudan a entender el desempeño de la base de datos, aplicación o servidor Web. [20]

Básicamente la herramienta proporciona:

- Pruebas escalables: Utilizando QALoad se puede emular la carga generada por cientos o miles de usuarios en la aplicación sin requerir la participación de los usuarios finales en sus equipos.
- Puede repetir fácilmente las pruebas de carga variando las configuraciones del sistema para alcanzar una aproximación óptima a los escenarios reales de uso. Con QALoad se configura un escenario de pruebas de carga para controlar las condiciones de las pruebas, crear los usuarios virtuales que se necesitan para simular la carga, iniciar y monitorizar las pruebas y se obtienen los informes de resultados.
- Fácil desarrollo de los scripts de prueba: QALoad ofrece módulos configurables que facilitan el desarrollo de los scripts de prueba. Estos módulos, llamados EasyScripts, permiten a los ingenieros de pruebas grabar el tráfico que genera la aplicación y convertirlo en un script de prueba. Los scripts de prueba resultantes reflejan el tráfico real generado por la aplicación y miden el tiempo empleado para completar las transacciones para garantizar que el sistema que sometido a las pruebas alcanza las especificaciones para las que ha sido construido.
- Análisis exhaustivo: Durante cada sesión de pruebas, QALoad ofrece las estadísticas de rendimiento en tiempo real, y permite a los ingenieros de prueba insertar puntos de control dentro de los scripts para identificar y revisar áreas específicas del rendimiento del sistema.
- QALoad muestra los resultados de las pruebas en una amplia variedad de informes y gráficas.
- Además, los datos de las sesiones de prueba se pueden exportar automáticamente a otros formatos para su revisión en detalle. [21]

❖ **LoadRunner**

LoadRunner es una herramienta para realizar pruebas de carga desarrollada por la empresa Mercury Interactive, que permite prever el comportamiento y el rendimiento del sistema. Además, permite poner a prueba toda la infraestructura corporativa para identificar y aislar los posibles problemas mediante la simulación de la actividad de miles de usuarios. Realiza pruebas en toda la infraestructura corporativa,

que comprende las soluciones e-business, ERP[1], CRM[2] y las aplicaciones personalizadas, simulando la actividad de miles de usuarios, con lo que los equipos de desarrollo de aplicaciones y sitios Web pueden mejorar el rendimiento de las aplicaciones. Es la herramienta de pruebas de carga más escalable que permite simular la actividad de miles de usuarios con los mínimos recursos de hardware. Se integra a la perfección con las herramientas de gestión del rendimiento de Mercury Interactive. Los mismos scripts creados durante las pruebas pueden volverse a utilizar para monitorizar la aplicación una vez terminada su implantación. Presenta una interfaz API abierta, con la que los usuarios y otros fabricantes pueden integrar LoadRunner en sus propios entornos.

- **JMeter**

Es una herramienta de software libre que ofrece la posibilidad de realizar pruebas de carga sobre diferentes aspectos de una aplicación Web, inicialmente diseñada para pruebas de estrés en aplicaciones Web, en la actualidad, su arquitectura ha evolucionado no sólo para llevar a cabo pruebas en componentes habilitados en Internet (HTTP), sino además en bases de datos, programas en Perl y prácticamente cualquier otro medio.

El JMeter muestra los resultados de las pruebas en una amplia variedad de informes y gráficas. Además facilita a una rápida detección de los cuellos de botella existentes debido al tiempo de respuesta excesivo. Todas estas herramientas pueden ser usadas para hacer las pruebas de eficiencia bajo carga intensiva, sin embargo hay algunas que poseen ventajas con respecto a las demás, por lo que son más óptimas a utilizar durante las pruebas en cuestión. En una aplicación es de vital importancia emplear algo de tiempo a preparar pruebas de eficiencia bajo carga y stress, antes de ser entregada. El tiempo invertido es recuperado con creces, ya que se detectaran los posibles efectos laterales y se podrá comprobar si esa nueva funcionalidad soporta la cantidad de usuarios concurrentes que se especificaban en los requisitos.

Objetivos específicos que se pueden tener [22]:

- Verificar que el sistema esté ajustado para soportar la máxima carga de trabajo posible usando la infraestructura actual.
- Asegurar que, ante una carga de trabajo determinada, las páginas a las que se accede responden en menos del intervalo de tiempo especificado por el grupo de diseñadores.
- Determinar el tiempo medio de respuesta que obtendrá el usuario.

[1] Planificación de Recursos Empresariales

[2] Administración de las relaciones con el cliente

- Determinar el número máximo de usuarios concurrentes que pueden acceder a una página específica, o transacciones por segundo que la aplicación es capaz de soportar.
- Identificar las páginas que responden más lentas y las más rápidas.
- Identificar las páginas con una mayor desviación típica en sus tiempos de respuesta.

Las pruebas de eficiencia bajo carga intensiva deben de ejecutarse idealmente sobre un entorno estable, lo más similar posible al entorno final de producción, siguiendo los objetivos antes mencionados, siempre pudiendo ser agregados otros en dependencia del interés que se persiga.

De las herramientas gratis, es la más completa y útil para el tipo de pruebas en cuestión.

Es una herramienta que sirve para realizar pruebas funcionales, pero también sirve para realizar pruebas de regresión en aplicaciones web, algo, que a veces es verdaderamente complicado, según la aplicación, pero que es casi imprescindible en el mantenimiento y evolución de las aplicaciones, si se quiere asegurar un nivel de capacidad adecuado en la entrega del producto.

Tiene una estructura en árbol que le da potencia, permitiendo que sea la imaginación de quien la use la que ponga los límites a la hora de diseñar el plan de prueba. Y brinda mayor cantidad de variantes para recoger los resultados obtenidos, que el resto de las herramientas gratis, lo que permiten hacer un análisis exhaustivo de las pruebas realizadas.

JMeter es fácilmente configurable y permite conocer los tiempos de respuesta experimentados por una aplicación cuando se tiene un número N de usuarios y el número real de transacciones procesadas por unidad de tiempo. Los resultados pueden ser visualizados en diferentes formatos, lo cual facilita las labores de análisis para el probador. [22]

Una ventaja de JMeter es que permite realizar pruebas de carga sobre cada una de las capas que conforman la aplicación a probar. De esta manera, en un momento dado, es sencillo localizar el foco que está generando contención y está afectando los tiempos de respuesta de un caso de uso específico.

1.13. Herramienta de prueba seleccionada

Tras el análisis realizado a cada una de las herramientas abordadas, es posible determinar que la herramienta más eficiente para la realización de este tipo de pruebas es JMeter, puesto que facilita las pruebas de carga, concurrencia y stress sobre una aplicación web.

Permite almacenar los resultados de la prueba y generar gráficos que representan los aspectos que se han probado. La posibilidad de contar una amplia documentación para el estudio de la utilización de la herramienta, supone una garantía en el éxito de la ejecución de las pruebas y de la realización de un manual que permita a otros proyectos de gestión, identificar puntos de control para la elaboración de

estas pruebas y la ejecución de las mismas, además de la recopilación y análisis de los errores encontrados.

Otro aspecto a favor de esta herramienta es que cumple con uno de los principales requisitos a cumplir en el marco de trabajo de nuestra Universidad, en el que se plantea que los productos a utilizar deben de encontrarse en el marco del software libre, que por las características que presentan existe un por ciento superior de confianza en la utilización de la misma ya que se puede contar con todo el código que genera a la herramienta. No es el caso de QALoad y de LoadRunner los cuales se encuentran desarrolladas por empresas propietarias fueras de este marco. Además, para la adquisición de este no es necesario la utilización de una licencia, ya que se permite bajar de forma gratuita desde la web.

Conclusiones parciales

Después de haberse realizado las investigaciones sobre los elementos teóricos que sustentan el problema científico y los objetivos del trabajo los objetivos y estrategias a seguir, así como los tipos de pruebas que existen, se ha adquirido el conocimiento necesario para realizar las pruebas a la aplicación.

En el proceso de pruebas para la evaluación del Generador se llegó a la conclusión que se estarán realizando las pruebas estructurales, de volumen, instalación y carga y estrés, por ser técnicas capaces de cubrir y evaluar las principales requisitos no funcionalidades del software, enmarcándolas en los niveles de unidad y de sistemas, utilizando para estos, los métodos de caja blanca y caja negra. Para automatizar el proceso de pruebas de carga y estrés se estará utilizando la herramienta JMeter, por sus particularidades de uso y ventajas.

CAPÍTULO 2

Diseño de casos de pruebas

Introducción

Este capítulo describe las características del GDR, del que se muestran los requisitos no funcionales, así como los casos de usos para cada uno de los módulos existentes del sistema. Detalla el proceso de diseño de las pruebas propuestas, partiendo de la estrategia elaborada para el desarrollo de las mismas.

2.1 Descripción del sistema a probar

El GDR es una herramienta multiplataforma, desarrollada con tecnologías web que posibilitan el diseño, edición y visualización de informes/reportes en diferentes formatos, extrayendo datos de una amplia gama de gestores de bases de datos, entre los que se encuentran PostgreSQL, MySQL y SQlite. Esta versión permite el diseño de tablas pivote y tablas cruzadas, la generación de gráficas dentro de los reportes y el soporte para el Sistema Gestor de Base de Datos (SGBD) Oracle y SQL-Server.

El GDR da la posibilidad de cargar en una vista estándar los modelos de bases de datos con los que se desea trabajar a través de una interfaz amigable, creada con el objetivo de brindar la mayor cantidad de opciones posibles para el diseño de los reportes, estos pueden ser personalizados dependiendo de las necesidades del informe deseado, dando la posibilidad de agregar gráficas e imágenes que muestren una representación más clara de los datos a valorar.

El Sistema para la Generación Dinámica de Reportes, está diseñado sobre una arquitectura modular y distribuida, que permite escalabilidad y flexibilidad. Los procesos se distribuyen entre varios componentes que permiten ser aplicados e integrados a soluciones personalizadas, en donde las funcionalidades son distribuidas en módulos. Dichas partes se complementan entre sí, haciendo más fácil la generación de reportes, puesto que aumentan la reutilización y reducen la redundancia de información.

Aunque le permite al usuario abstraerse en parte de los conocimientos relacionados con los gestores de bases de datos, es necesario poseer conocimientos básicos para el trabajo con el sistema. Su entorno de trabajo está estructurado de forma que es difícil guiarse en la creación y generación de reportes.

No presenta un entorno de comprobación de su seguridad, solo la determina, una vez integrado a algún sistema donde se establece un nivel de seguridad propio.

El conjunto de módulos que brindan funcionalidades para el trabajo con reportes son:

1- Diseñador de Modelos.
2- Diseñador de Reportes.
3- Diseñador de Consulta.
4- Visor de Reportes.
5- Administrador de Reportes.

2.1.1. Requerimientos

Para todo proceso de desarrollo de pruebas se hace necesario tener especificado el conjunto de requisitos que van a formar parte de la elaboración de las pruebas. Pues estos constituyen el pilar por el cual se van a basar para determinar si un producto presenta la calidad requerida.

Tras el desarrollo de los tipos de pruebas a aplicar, el conjunto de requisitos que se tendrán en cuenta se enumeran a continuación, todos ellos se definen en la planilla de *Especificación de requisitos de software* que se encuentra en el *expediente de proyecto* publicada en el repositorio del centro DATEC. Cada uno de ellos se encuentra definidos como requisitos no funcionales, es decir que representarán las cualidades o propiedades que debe de poseer el sistema.

- **Fiabilidad:**

1. El sistema deberá estar disponible las 24 horas del día, pudiendo estar fuera de servicio por un período de 72 horas máximo. La precisión y exactitud de las salidas del sistema se corresponden con la calidad y exactitud de la información contenida en las base de datos desde donde se extraigan los reportes.

- **Usabilidad:**

2. Permitir personalizar los reportes: Consiste en que un usuario pueda diseñar un reporte lo más ajustado posible a sus necesidades, pudiendo cambiar la tipografía de el texto, agregar imágenes, ubicar en el área de diseño los campos en la sección que el usuario desee, usando características de arrastrar y soltar.
3. El usuario debe poder localizar un reporte de manera rápida, si es posible tenerlo ubicado en alguna categoría que permita dicha ubicación y tener la posibilidad de buscarlo.
4. El usuario debe poder diseñar una consulta SQL de manera ágil y sencilla sin necesidad de ser un experto en el proceso de diseño de un reporte.
5. La eficiencia del sistema depende en gran medida de la velocidad de conexión a las base de datos donde se encuentre, así como del volumen de información contenido en las mismas.

❖ **Eficiencia:**

6. El tiempo máximo para la obtención de un reporte no debe sobrepasar los 5 minutos.
7. Como promedio un reporte deberá demorar alrededor de 10 segundos.
8. El sistema deberá permitir que existan al menos 100 usuarios conectados de forma simultánea.

❖ **Interfaz:**

9. El usuario deberá acceder a la aplicación a través del protocolo HTTP usando el navegador Firefox 2 o una versión superior.

❖ **Interfaces Hardware:**

10. Por su naturaleza, el sistema no interactúa con ningún tipo de interfaz de Hardware.

2.1.2. Casos de uso

Los Requisitos Funcionales del sistema en cuestión han sido agrupados en 19 Casos de Uso, distribuidos en los 5 módulos del sistema. Para el conocimiento de ellos se relacionan a continuación agrupándolos en correspondencia al módulo que pertenece cada cual. Todos ellos se definen en la planilla de *Identificación de escenarios X Casos de Uso* que se encuentra en el *expediente de proyecto* publicada en el repositorio del centro DATEC.

Módulos	Casos de Uso
Diseñador de modelo	▪ Diseñar modelo. ▪ Administrar modelo. ▪ Gestionar origen de datos.
Diseñador de reporte	▪ Diseñar reporte. ▪ Diseñar tipo de reporte tabular. ▪ Diseñar tipo de reporte con tabla pivote. ▪ Diseñar tipo de reporte con tabla cruzada. ▪ Seleccionar fuente de datos. ▪ Visualizar estructura de reporte. ▪ Gestionar grupo. ▪ Configurar parámetros.
Visor de reporte	▪ Visualizar reporte. ▪ Filtrar reporte.
Administrador de reportes	▪ Administrar reporte. ▪ Gestionar categorías de reportes.

Diseñador de consulta	▪ Diseñar consulta. ▪ Visualizar escritura SQL.
Casos de Uso Comunes	
▪ Generar vista previa de reporte en HTML. ▪ Obtener datos desde R-Server.	

Tabla 1: Relación de casos de usos por módulos

2.2 Estrategias de prueba

Para efectuar el proceso de evaluación del software, en la que el proceso de pruebas ocupa el papel principal en todo el desarrollo, se trazó una estrategia de prueba capaz de reflejar el conjunto de acciones a realizar en dicho proceso de manera organizada y secuencial. Para ello se integran un conjunto de actividades que describen los pasos que hay que llevar a cabo en el proceso de prueba (planificación, diseño de casos de prueba, ejecución y resultados), en la que se toma en consideración cuanto esfuerzo y recursos se van a requerir, con el fin de obtener como resultado la correcta construcción del software, para así, dar al cliente un mayor nivel de satisfacción del sistema desarrollado.

2.2.1. Proceso de prueba

Con el fin de obtener un proceso detallado y explicativo, se presenta una secuencia de pasos que permitirán llevar todo el proceso de evaluación:

- Selección de los niveles, técnicas y métodos de pruebas.
- Elaboración del plan de pruebas.
- Análisis de la documentación del sistema.
- Realización de pruebas exploratorias.
- Diseño de los casos de pruebas.
- Realización de pruebas de unidad a las funcionalidades críticas para el desempeño del sistema.
- Realización de pruebas de Instalación.
- Realización de pruebas de carga y estrés al software.
- Realización de pruebas de volumen.
- Documentar de los resultados de las pruebas y las no conformidades detectadas.
- Evaluación de las no conformidades detectadas por el equipo de desarrollo de Calisoft.

- Análisis de los resultados de las pruebas.
- Evaluación del producto.
- Recomendaciones.

Para un mejor entendimiento del proceso a llevar se detalla la siguiente figura:

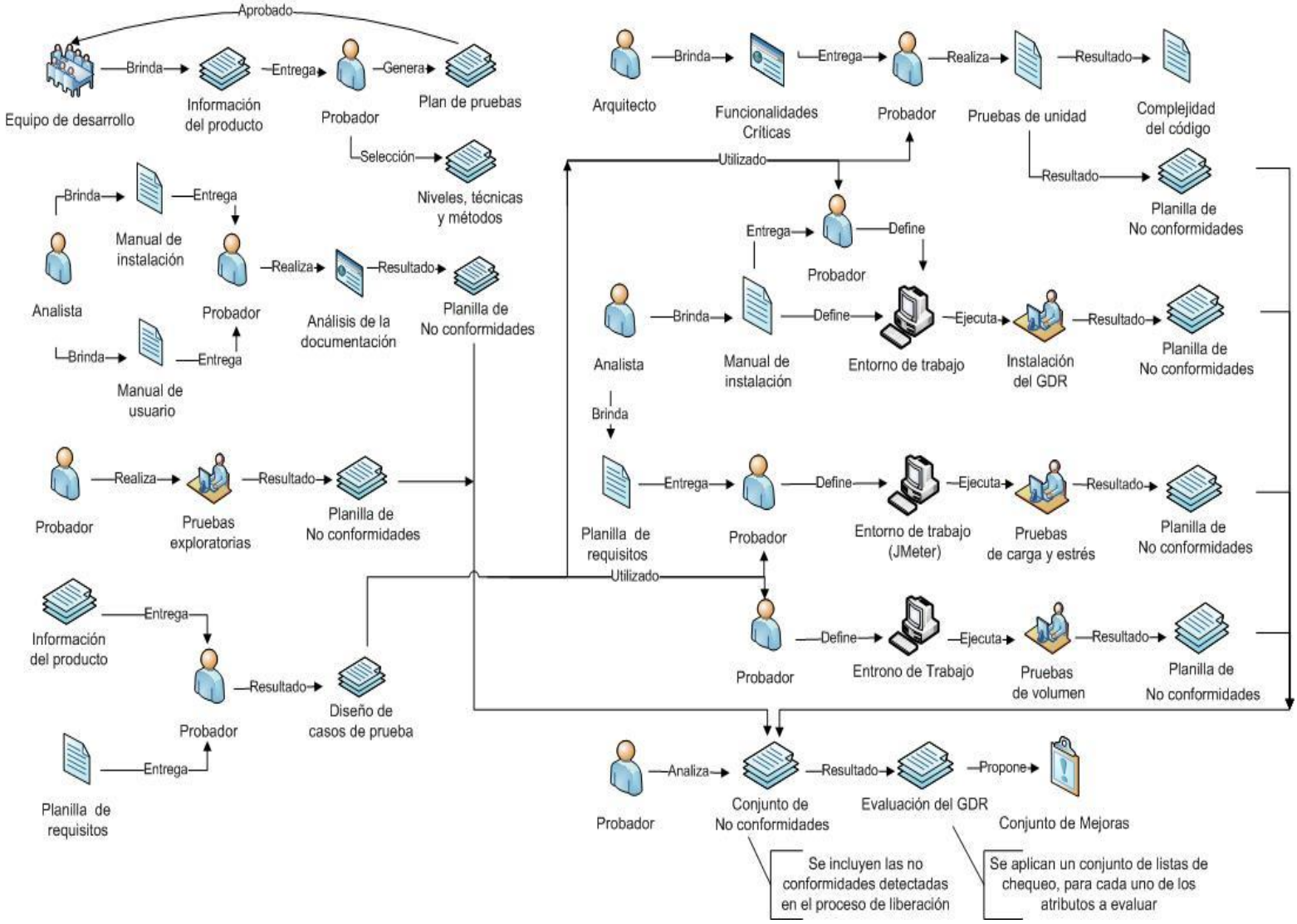


Figura 2: Proceso de Prueba

Para el proceso de ejecución de las pruebas el probador hará uso de los entregables, proporcionados por el equipo de desarrollo. Además se podrá ser uso de estrategias que el probador estime conveniente para facilitar la obtención de los objetivos.

Las pruebas se realizarán de acuerdo a la secuencia de pasos definida, en donde los fallos o errores encontrados serán almacenados en la plantilla de no conformidades, las que posteriormente serán analizadas para determinar la evaluación del sistema.

2.2.2. Niveles, técnicas y métodos de pruebas empleados.

El desarrollo de las pruebas de software, como se ha visto anteriormente, tiene como principal objetivo revelar defectos en un software. Estas miden características del sistema, en las que las que el proceso de detección de errores pueden estar dadas en líneas, métodos, clases, componentes, módulos, o el sistema en general. Es por ello que se hace necesario determinar niveles que permiten dar a conocer el alcance que se está obteniendo tras la ejecución de de las pruebas. Ahora bien, no basta con determinar hasta donde quiero llegar, sino que tipo de caso de prueba me encuentro ejecutando, si por una parte deseo obtener un examen de los detalles procedimentales del código o únicamente tener conocimiento las funcionalidad que el sistema debe realizar. Por su parte las técnicas responden que característica del software, estoy probando, que me permite determinar si se están cumpliendo tanto los requisitos funcionales, como los no funcionales determinados por el cliente. Es por ello que para un buen diseño y ejecución de las pruebas se hace necesario tener aparejado los niveles, técnicas y métodos de desarrollo.

Para el desarrollo de este trabajo y como se determinó en el capítulo 1 los niveles en los que se medirán en el sistema estarán dados por, las pruebas de unidad y pruebas de sistema.

Las pruebas de unidad por su parte permitirán evaluar el código, por lo que será necesario utilizar el método de caja blanca en la que a partir de las técnicas de camino básico, con la utilización de los grafos de flujo se podrá determinar la complejidad del código en cuestión.

Durante el nivel de pruebas de sistema se utilizará para el desarrollo de los casos de pruebas el método de caja negra. En el que se aplicarán pruebas tales como pruebas de volumen, carga y estrés e instalación.

2.2.3. Plan de prueba

El plan de pruebas constituye un artefacto de considerable importancia en el proceso de pruebas y es lo primero que se genera en este proceso, para planificar y trazar una estrategia, en la que proporciona el marco dentro del cual el equipo de probadores desarrolla las pruebas trabajando con los recursos y la planificación dada. Para su diseño se tuvieron en cuenta las características del software a liberar y las pruebas que se le aplicarían al mismo.

De manera general, suministra la siguiente información:

- Roles y responsabilidades.

  Descripción del equipo de probadores, por quienes está compuesto, responsabilidad de cada miembro.
- Escenario de pruebas.

  Descripción del escenario en el que se ejecutarán las pruebas, en los que se plantean el despliegue y los recursos del sistema.
- Requerimientos a probar.

  Listado del conjunto de requerimientos a probar.
- Estrategia de pruebas de aceptación.

  Se describe el flujo de trabajo que se utilizará para la ejecución de las pruebas.
- Evaluación de las pruebas.

  Se describen los criterios de evaluación para las pruebas.
- Cronograma.

  Cronograma de ejecución de las pruebas.

Para determinar los elementos que aquí se explicaron, remitirse a la planilla *Plan de prueba* en los *materiales complementarios*.

2.3 Planificación de pruebas

La planificación de las pruebas se realizará conjuntamente con el equipo de desarrollo, a medida que este le proporciona los recursos necesarios, artefactos, entregables, entre otros aspectos de importancia para poder establecer el desarrollo de las pruebas. Las pruebas se ejecutarán por un probador.

2.4 Realización de pruebas a la documentación

Para el desarrollo de este conjunto de pruebas se estableció la utilización de las listas de chequeo. Se realizará un análisis minucioso de todos los aspectos a tener en cuenta para la confección con la calidad requerida de la documentación de la aplicación. Para ello se han establecido que el manual de usuario y de instalación, serán los seleccionados para su revisión, pues se han establecido como las más relevantes a la hora de la liberación del producto, ya que constituirán los artefactos imprescindibles para poner en funcionamiento el sistema.

2.5 Realización de pruebas exploratorias

El término “Pruebas Exploratorias” fue introducido por Cem Kaner, se refiere a ejecutar las pruebas a medida que se piensa en ellas, sin gastar demasiado tiempo en preparar o explicar las pruebas, confiando en los instintos [23].

Las pruebas exploratorias son las primeras que se realizan al sistema por parte del equipo de pruebas, con el fin de encontrar errores superficiales que puedan existir. Estas son el resultado del estudio y aplicación del sistema en el contexto adecuado. Constituyen una guía para las pruebas venideras en base a los resultados que se van obteniendo durante su realización. Por su sencillez no se utilizan métodos para su ejecución, lo que no significa que sean menos importantes o se puedan ignorar. Por la poca complejidad que esta conlleva se decidió que para su ejecución se estimará solo un día de trabajo.

2.6 Diseño de los casos de prueba

“El objetivo del proceso de diseño de casos de prueba es crear un conjunto de casos de pruebas que sean efectivos descubriendo defectos en los programas y muestren que el sistema satisface sus requerimientos”. [24]

Como bien se plantea, el diseño de casos de prueba, tiene como finalidad conseguir gran confianza en el producto que se prueba. Si bien se conoce, que es casi imposible desarrollar casos de pruebas para todas las funcionalidades del sistema, la idea fundamental para el diseño de casos de prueba consiste en elegir algunas de ellas que, por sus características, se consideren representativas del resto. Así, se asume que, si no se detectan defectos en el software al ejecutar dichos casos, se puede tener un cierto nivel de confianza en que el programa no tiene defectos.

En fin, los casos de prueba fueron diseñados con el propósito de especificar una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, los resultados esperados y las condiciones bajo las que ha de probarse.

El diseño de los casos de pruebas se realizó apoyándose en las especificaciones de casos de uso y de requisitos que se encuentran en el expediente del proyecto del GDR.

Para el GDR se diseñaron los casos de prueba en dependencia de las técnicas que se iban a utilizar.

2.7 Realización de pruebas de unidad

Con el desarrollo de esta prueba se pretende determinar la efectividad y complejidad del código. La gran cantidad de código que existe en la implementación de la aplicación hace que sea muy difícil determinar la efectividad del código de manera general, por lo que se tomó como estrategia determinar

los casos de usos arquitectónicamente significativos del sistema, y de estos a su vez, obtener las funcionalidades críticas para su desempeño. A estas funcionalidades se le aplicarán las pruebas de camino básico para obtener una medida de la complejidad lógica del diseño procedimental.

Para el logro de los objetivos se estará desarrollando a partir de la estrategia que se plantea a continuación:

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

- **Notación del grafo de flujo.**

El grafo de flujo representa el flujo de control lógico mediante la notación que se ilustra en la Figura 3. Cada círculo, denominado nodo del grafo, representa una o más sentencias procedimentales. Las flechas del grafo, denominadas aristas o enlaces, representan el flujo de control. Una arista debe terminar en un nodo, aunque el nodo no represente ninguna sentencia procedimental. Las áreas delimitadas por las aristas y nodos se denominan regiones. Cuando se contabiliza las regiones se incluye el área exterior del grafo, contando como una región más.

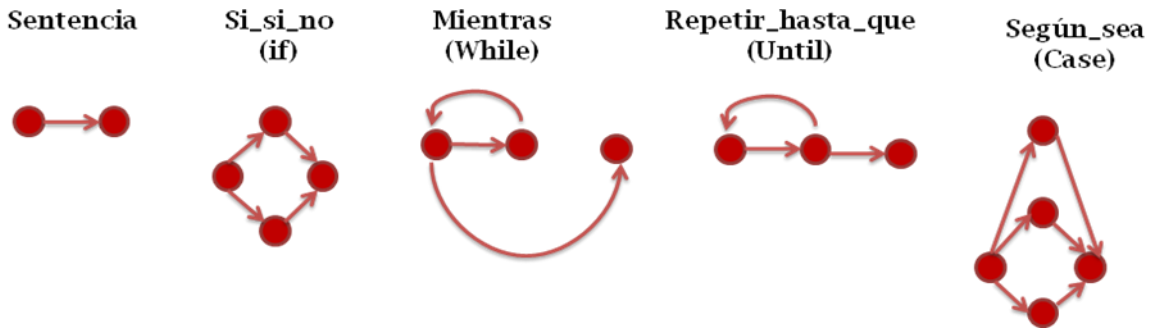


Figura 3: Notación de grafo de flujo

- **Cálculo de la complejidad ciclomática**

Para calcular la complejidad ciclomática del grafo en cuestión, se estará desarrollando a partir de las diferentes formas planteadas por Pressman en su 4ta edición. Estas son:

1- El número de regiones del grafo coincide con la complejidad ciclomática, V (G).
2- La complejidad ciclomática, V (G), de un grafo de flujo G se define como V (G) = Aristas – Nodos + 2
3- La complejidad ciclomática, V (G), de un grafo de flujo G se define como V (G) = Nodos Predicado + 1

Por ser considerada la más fácil y con vista a acelerar el proceso de cálculo, se ha establecido que la forma a utilizar será la 2da propuesta.

2.8 Realización de pruebas de instalación

Se pretende determinar el comportamiento del sistema en diferentes entornos. Para ello se establecerá un conjunto de pasos para proceder a la ejecución de las pruebas en cuestión:

1- Como precondiciones el equipo de desarrollo proporcionará las planillas de especificación de requisitos tecnológicos de instalación y del manual de Instalación del sistema.
2- Se procederá a la instalación de la aplicación a partir de la secuencia de pasos definida en el manual de instalación.
3- Se procederá a la instalación del sistema, en diferentes entornos para determinar el comportamiento en su proceso de instalación. En el proceso se contará con máquinas que carezcan de algunos de los paquetes necesarios para su instalación.

2.9 Realización de pruebas de carga y estrés.

Una prueba de carga a un sistema informático se realiza para observar el comportamiento de éste bajo una cantidad determinada de peticiones. La carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga. Esta prueba puede calcular los tiempos de respuesta de todas las transacciones importantes de la aplicación.

La realización de esta prueba posibilita determinar probabilidades de encontrar errores de rendimiento en el software y verificar que funciona eficientemente bajo condiciones anormales. Para la realización de estas pruebas se utilizó la herramienta JMeter para la automatización de las mismas.

Para el desarrollo de las mismas se detallan un conjunto de estrategias para alcanzar el logro de los objetivos.

1. Búsqueda de Requisitos No Funcionales.

Se buscan los Requisitos No Funcionales de dicha aplicación los cuales son cantidad de usuarios máximo a conectar, tiempo máximo de demora en responder el servidor, requisitos de rendimiento, requisitos de usabilidad, requisitos de soporte, requisitos hardware y requisitos software.

2. Grabar los escenarios críticos.

En este paso se graban todos aquellos escenarios que son críticos para la aplicación, dichos escenarios no son más que probar detalladamente los casos de uso críticos de la aplicación.

3. Incluir Elementos de Pruebas.

En este punto se incluyen elementos de pruebas los cuales facilitarán un mejor entendimiento de dicha prueba. Por ejemplo: Grupo de Hilos, Informe Agregado, Ver Árbol de Resultados.

4. Ejecución de las Pruebas y almacenar los Resultados.

En este punto se ejecuta la prueba, para ello:

Se determina el número de usuarios que se desea intervengan a la vez en un/os camino/s crítico/s.

Se van simulando incorporaciones de usuarios que ejecutan el mismo camino, todos al mismo tiempo.

Los resultados se almacenarán en el Informe Agregado para su posterior análisis.

5. Analizar resultados.

Se analizan los resultados en cuanto a los valores de los parámetros arrojados por el Informe de resultado.

Para el desarrollo de las mismas se hace necesario tener un conjunto de requisitos para su entorno de prueba, estos son:

- Herramienta JMeter 2.3.
- Maquina virtual de Java: Java Virtual Machine 1.3 o superior.
- Navegador: Internet Explorer o Mozilla.
- Sistema Operativo Windows XP Profesional o Linux (Ubuntu).

De ellas, se determinó que para las pruebas se estarán utilizando la máquina virtual en su versión 6, como navegador se utilizará Internet Explorer y Windows XP Profesional como sistema operativo.

Por parte del equipo de desarrollo, quedó estipulado que el módulo Visor de Reporte constituye el más eficiente para la realización de este tipo de pruebas, pues este por su alta conectividad que presenta con la base de datos, hace que este sea el que se encuentre más vulnerable a fallas en el rendimiento del sistema. Asimismo se estará comprobando dos casos de usos pertenecientes a dos de los módulos del sistema, que permitirán evaluar el funcionamiento normal del sistema, a pesar de que no tengan una alta vulnerabilidad ante el fallo. Por lo que queda conformado, que los casos de usos que se probarán con el JMeter son:

- Visualizar reporte.
- Exportar reporte.
- Administrar reporte.

Para el proceso de desarrollo de estas pruebas es necesario que queden declaradas explícitamente las especificaciones de rendimiento. Estas permiten brindar conocimientos sobre características de rendimiento que se supone deba tener la aplicación en un ambiente real de ejecución. Los requisitos que se tendrá en cuenta serán los 6,7 y 8 de los mencionados en el epígrafe 2.1.1.

Se realizarán tres iteraciones interactuando con el sistema, probando todos los escenarios de cada caso de uso implícito. Se establecen un período de subida de 1 segundo, bajo diferentes condiciones, que a continuación se relacionan:

1. 50 usuarios en 2 bucles, o sea, 50 conexiones en un primer momento, luego las próximas 50, con la totalidad de 100 conexiones.
2. 100 usuarios en 2 bucles, o sea, 100 conexiones en un primer momento y luego las próximas 100, llegando así a la totalidad de las 200.
3. 300 usuarios en 1 bucle, o sea, 300 conexiones desde un primer momento.

Los resultados totales de las diferentes peticiones se van guardando en el Informe Agregado y el Árbol de Resultados.

A continuación, se describen los aspectos mostrados en la tabla de la Plantilla de Reporte de los Resultados:

Muestras: Cantidad de páginas (Hilos) que simulan la cantidad de usuarios que están interactuando con el sistema desde la misma URL.

Media: Media de tiempo en milisegundos en que las páginas que se cargaron de manera satisfactoria.

Mediana: Tiempo promedio que han tardado en cargarse las páginas.

Min: Tiempo mínimo que ha demorado en cargarse una página.

Max: Tiempo Máximo que ha tardado en cargarse una página.

Línea 90 %: 90 por ciento del tiempo en el que las páginas que se cargaron de manera satisfactoria.

%Error: Por ciento de error de las páginas que no se llegaron a cargar de manera satisfactoria.

Kb/Seg: Velocidad de carga de las páginas.

Rendimiento: Representa el número de muestras por unidad de tiempo.

En el Listener ”Árbol de Resultado”, se puede observar el instante en el que se van cargando las peticiones http de manera factible o no, el Resultado del Muestreador, así como la respuesta de sus códigos. Para los casos en los cuales ha sido fallo la petición http se mostrarán en rojo dicha peticiones y para el caso en que sean cargadas satisfactoriamente se mostrarán en verde.

2.10 Realización de pruebas de volumen

El desarrollo de estas pruebas permite analizar el comportamiento de la aplicación y de la base de datos con volumen de información almacenada similar a los esperados en la explotación real del sistema. A pesar que el generador trabaja con diferentes SGBD, la realización de estas pruebas es de gran importancia, pues esta permitirá encontrar errores que no solo determinen la carga máxima continua que puede manejar el software a prueba en un período dado, sino que permite detectar otros tipos tales como consultas lentas por mal diseño, así como problemas en los procedimientos de almacenados.

Las pruebas estarán enfocadas en determinar el comportamiento del sistema tras la inserción de diferentes volúmenes de datos, para ello se efectuarán la obtención de reportes para la realización de los conjuntos de escenarios que se establecen.

Para la comprobación de las pruebas se utilizará Firebug un complemento de Mozilla. En la figura siguiente se ilustra su interfaz.

Figura 4: Interfaz del Firebug

A continuación, se describen los aspectos mostrados en la interfaz, donde se visualizarán los resultados:

URL: URL utilizada y método HTTP usado.

Status: Estado de la respuesta recibida.

Domain: Dominio de la petición ejecutada.

Size: Tamaño de la respuesta recibida.

Timeline: Información detallada de los tiempos de petición y respuesta.

Los resultados de las pruebas estarán almacenados en la planilla de pruebas de volumen que se encuentra en los materiales complementarios. En esta se mostrará el conjunto de valores que se estuvieron teniendo en cuenta para la realización de la prueba. En caso de resultar insatisfactorio el resultado que arrojen éstas, se procede a llenar la tabla de registro de las No Conformidades.

Para la documentación de las pruebas se hizo necesaria la realización de una planilla que dejara vigente los elementos tenidos en cuenta, la razón estuvo dada a partir de la no existencia de un

documento que mostrará de forma precisa los elementos a tomar en cuenta para la realización de estas pruebas.

La realización de la planilla fue realizada a partir de las características que presenta el sistema, por lo que no es aplicable a cualquier tipo de productos, solo aquellos que presenten semejanzas con el producto en cuestión.

Para su mejor entendimiento se describirán brevemente a continuación cada una de los parámetros por lo que está compuesta la matriz de datos.

Sección	Escenario de la sección	Descripción	Cantidad de tuplas	Filas x Columnas	Otros elementos	Volumen de datos	Tiempo de carga	Criterio de aceptación

Tabla 2: Matriz de datos

Sección: Nombre del módulo que se estará probando.

Escenario: Nombre del escenario que se probará.

Descripción: Descripción del escenario en cuestión.

Cantidad de tuplas: Número de tuplas que posee el reporte.

Filas x Columnas: Número de filas y columnas que presenta el reporte.

Volumen de datos: Valor que determina el tamaño de los datos que existen en la petición realizada.

Otros elementos: Elemento que determina si en el reporte existen valores adicionales además de las tablas.

Tiempo de carga: Tiempo que demora en realizar la petición.

Criterio de aceptación: Elemento que determina si la prueba fue satisfactoria.

Los escenarios a llevar a llevar a cabo estarán dados por los que poseen mayor interacción con la base de datos. Tras una selección, se obtuvo que los escenarios implicados en el proceso estarán dados por: Generar, Visualizar, Exportar, Imprimir, Mover y Copiar Reporte.

El conjunto de pasos a llevar a cabo son:

- Selección y llenado de las tablas a ser pobladas a partir de datos válidos.
- Diseño de reportes a partir de las tablas pobladas.
- Ejecución de los escenarios para determinar el comportamiento del sistema.

2.11 Pruebas funcionales

El desarrollo de estas pruebas no formará parte del desarrollo de la gama de pruebas a realizar por parte del probador. La razón viene dada por la aplicación de este conjunto de pruebas tras el proceso de liberación en que estuvo enmarcado el producto, en el que el departamento de calidad de la Universidad pudo realizar, las que responden a los requisitos funcionales en la que a partir de casos de pruebas a través del método de caja negra se pretende mostrar los errores que presenta el sistema. En el proceso de evaluación, se hace necesario tener conocimiento sobre el conjunto de estas pruebas, en la que se obtendrá un reporte del conjunto de no conformidades detectadas tras la conclusión del proceso de liberación. Este permitirá obtener un reporte de cuáles son las funcionalidades que mayor dificultad han presentado, así como medir por cada iteración el número de no conformidades detectadas y la comparación del porcentaje de erradicación.

Conclusiones parciales

En este capítulo se expuso una breve descripción del sistema, donde se detallaron los casos de uso y los requerimientos no funcionales del mismo. Se describió los aspectos fundamentales del plan de prueba, así como la planificación y desarrollo de la estrategia de pruebas a realizar para el establecimiento de la evaluación del producto. Se diseñaron los procedimientos a desarrollar para cada una de las pruebas, los cuales permitieron una mejor comprensión del funcionamiento y ejecución de los casos de pruebas.

CAPÍTULO 3

Aplicación de las pruebas y análisis de los resultados

Introducción.

En este capítulo se muestran y se analizan los resultados obtenidos luego de ejecutar las pruebas pertinentes al proceso en que se encuentra enmarcado el desarrollo del trabajo. Una vez terminado se procura analizar las no conformidades registradas de los defectos y fallos del sistema durante todo el proceso con el fin de proporcionar una evaluación del producto que se prueba. Se establece una valoración de acuerdo a parámetros de calidad como la eficiencia, portabilidad, usabilidad y confiabilidad del sistema, basándose en los resultados obtenidos. Además se propone un conjunto de mejoras que permitirán elevar el nivel de la aplicación.

3.1 Análisis de la documentación de la aplicación.

Como bien fue planteado en el capítulo anterior, se le han desarrollado un conjunto de pruebas a los documentos Manual de Usuario y de Instalación, las que van a estar definidas por la aplicación de listas de chequeo, con vista a obtener las irregularidades que puedan detectarse.

Las listas de chequeo arrojaron los siguientes problemas en el manual de usuario:

Ha podido ser demostrada que en la estructuración de la documentación no existe una adecuada formulación de la planilla, al existir falta de secciones que pretenden ser importantes para establecer una plantilla entendible y bien conformada. Esto repercute en la falta de informaciones relevantes que brindan datos adicionales al usuario.

A pesar de no existir una planilla específica que determine la confección de estos dos documentos, se hace necesario que para la elaboración de estos se cumpla con los parámetros imprescindibles en cada una de las planillas que permiten obtener una mejor comprensión por parte del usuario, asegurando que se establezca una mayor calidad de las mismas. La elaboración de elementos como la introducción, que es un aspecto que muchas veces es obviado en tales documentaciones, permite determinar los objetivos que se persiguen y elementos que son de importancia para el entendimiento del contenido.

Las listas de chequeo para el manual de instalación arrojaron los siguientes problemas:

Existen conjuntos de pasos que para un mejor entendimiento de lo que se desea trasmitir, la demostración con imágenes sería una opción factible y así se obtendría una mejor aceptación por

parte del cliente.

Existe una falta de explicación en algunos de los procedimientos a realizar. Hay que destacar que el manual va a ser utilizado por usuarios de los que no se conoce como será su conocimiento sobre los temas referentes. Elementos como los pasos 1, 2 y 3 del epígrafe 1.1 del Manual de usuario, no se especifican de la mejor manera posible, dejando a consideración de los usuarios lo que se desea realizar.

De forma general, la evaluación dada quedó de la siguiente manera:

- Para el manual de usuario se obtuvo una evaluación de bien.
- Para el manual de instalación se obtuvo una evaluación de regular.

Para obtener más detalles de las listas de chequeo aplicadas remitirse a *materiales complementarios.*

Para una confección aceptada del manual de usuario se ha decidido detallar un conjunto de pasos, que permitirán elaborarlo de forma correcta y así mejorar la calidad:

1. Portada: ¿De qué se trata el documento y quién lo elaboró? Además se recogerá el nombre del producto y del proyecto al cual pertenece, así como la versión que se estará analizando.
2. Introducción: Describe el uso del documento: ¿para qué sirve y de qué habla?
3. Análisis y requerimientos del sistema: ¿qué se ocupa para poder instalarlo y usarlo?
4. Explicación del funcionamiento: Se debe de poner paso a paso y con pantallas bien explicadas cómo funciona el programa.
5. Glosario
   - Debe ser escrito de tal manera, que cualquier persona pueda entenderlo con la menor dificultad posible.
   - Es recomendable, detallar todos aquellos pasos que se llevan a cabo para usar el programa.
   - Especificar los alcances y las limitaciones que tiene el programa.
   - Un buen punto de partida para un manual de usuario, es hacer de cuenta que las personas que lo van a leer no tienen el más mínimo conocimiento sobre computadores. [25]

3.2 Ejecución y análisis de las pruebas exploratorias a la aplicación.

Las pruebas Exploratorias, como se especificó en capítulos anteriores, constituyen la primera iteración entre el grupo de probadores y el sistema, buscando fallas superficiales que puedan existir en las interfaces de la aplicación.

El GDR cuenta con un conjunto de módulos a través de las cuales el usuario interactúa con el sistema.

Los resultados de estas pruebas no arrojaron ningún problema en los vínculos y lógica entre las interfaces, mostrándose únicamente la información que realmente necesita el usuario. A pesar de esto, fue posible demostrar que el sistema presenta algunos problemas en el tiempo de carga. Tales errores serán valorados en las pruebas de carga y estrés.

3.3 Ejecución y análisis de las pruebas de unidad a la aplicación.

Para la obtención del riego que pueden presentar los métodos implementados, se referenció en el trabajo desarrollado por McCabe (Director Principal de las normas de facilitación en el Instituto Nacional Americano de Estándares(ANSI)) que atribuye valores de referencia para la obtención de los riegos del código, estos son los que se detallan a continuación:

- 1-10, métodos sencillos, sin mucho riesgo.
- 11-20, métodos medianamente complejos, con riesgo moderado.
- 21-50, métodos complejos, con alto riesgo.
- 51 o más, métodos inestables, de altísimo riesgo.

Tras la selección de funcionalidades representativas del GDR y tras la elaboración del conjunto de grafos de flujo constituirá a determinar la medida del grado de riesgo de la codificación del sistema a partir del cálculo de la complejidad ciclomática.

Para obtener la codificación cuyos resultados será cada uno de los posteriores grafos de flujo, remitirse al Anexo 1.

1) Funcionalidad del módulo: *Diseñador de Modelo.*	**2) Funcionalidad del módulo: *Diseñador de Reportes.***
Clase: Class buildModelAction extends abstractBuildModelAction	**Clase:** class addReportAction extends gdrAction
Cálculo de complejidad ciclomática:	Cálculo de complejidad ciclomática:
V (G)= Número de Aristas – Número de Nodos + 2	V (G)= Número de Aristas – Número de Nodos + 2
V (G)= 12-11+2 =3	V (G)= 17-16+2 =3

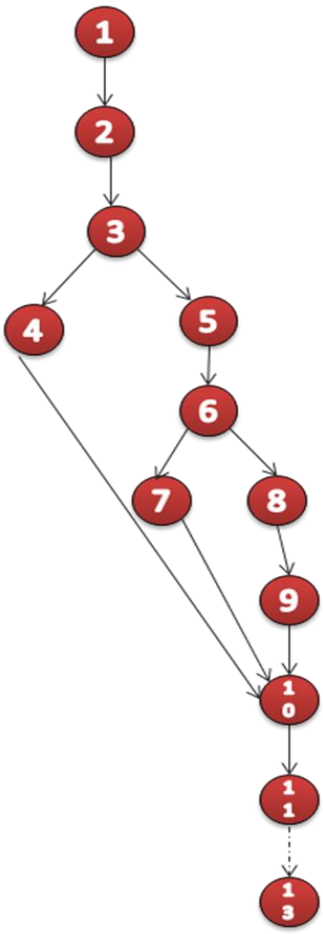


Figura 5:Grafo de Flujo

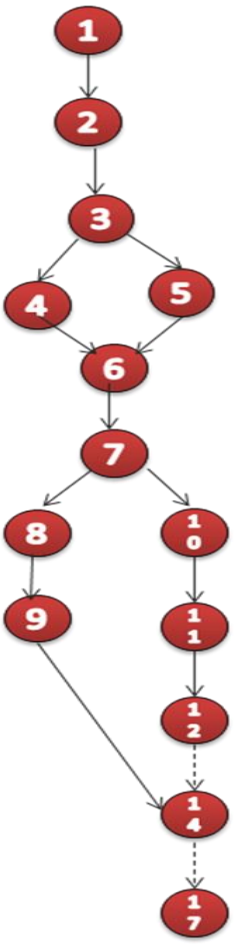


Figura 6:Grafo de Flujo

3) **Funcionalidad del módulo: *Administrador de Reportes.***

Clase: class ResultQueryAction extends sfAction

Cálculo de complejidad ciclomática

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

V (G)= 11-10+2 =3

4) **Funcionalidad del módulo: *Diseñador de Reportes.***

Clase: class existReportAction extends sfAction

Cálculo de complejidad ciclomática

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

V (G)= 7-6+2 =3

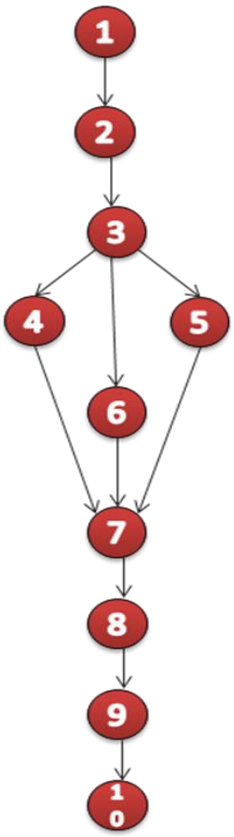


Figura 7:Grafo de Flujo

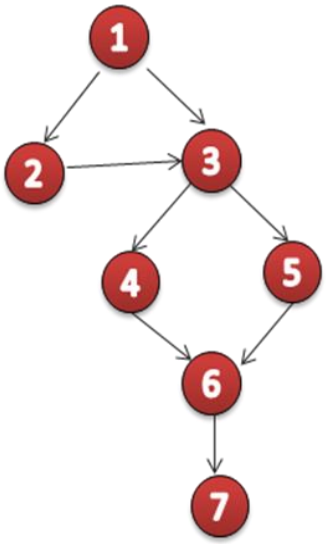


Figura 8:Grafo de Flujo

Después del cálculo de la complejidad al conjunto de clases seleccionadas se obtiene como resultado, que todos los valores obtenidos se presentan en el rango cuyos métodos son sencillos y sin muchos riesgos, lo que se establece como conclusión que no se indican problemas de diseño, además de no indicar problemas en el cual su codificación sea poco comprensible (y por ello costoso de cambiar), por tanto no se considera complejo de probar.

Como estrategia para una mejor valoración de la codificación se empleó una lista de comprobación, en las que a partir de un conjunto de preguntas se definió como percibe el equipo de desarrollo el estado de la codificación del sistema. Este tiene como objetivo unificar los criterios obtenidos de los participantes en el equipo de trabajo, para de esta forma obtener cuales son los principales problemas presentados tras el proceso de implementación del sistema y de esta forma trazar estrategias que permitan erradicarlos para futuros proyectos.

Para determinar el conjunto de preguntas realizadas y la puntuación dada por parte de los entrevistados, remitirse al Anexo 2.

Tras la elaboración de la entrevista a los miembros del equipo de desarrollo, mayormente dados por implementadores del sistema, permitió determinar los diferentes criterios existentes sobre la codificación de la aplicación. Se obtuvo muy buena semejanza entre los resultados.
Como elemento irregular es posible determinar que existen interrogantes cuyos valores están dados por una puntuación de 3 y 2 puntos. Se hace necesario mencionar que los valores proporcionados, están dados por las malas prácticas de desarrollo. Las interrogantes fueron contestadas por los mismos implicados en el proceso de confección del producto, lo que demuestra que aunque se conocen los posibles defectos que se pueden obtener en el desarrollo del mismo, no se aplica una mitigación de errores por parte del propio programador. Esta razón viene dada por la falta de un proceso detallado y genérico que permita a los implicados mejorar las estrategias de implementación.
Es necesario destacar que muchos de los errores que presentan hoy en día están dadas por la no existencia de buenas prácticas de programación lo que hace que para futuras modificaciones de los sistemas desarrollados no se entienda lo realizado con anterioridad.

Para determinar el conjunto de preguntas elaboradas remitirse a *Materiales complementario*s en *Listas de comprobación del código*.

3.4 Ejecución y análisis de las pruebas de instalación a la aplicación.

Tras el proceso de instalación que se llevó a cabo debe destacarse que para emprender la instalación y configuración del producto es necesario el dominio del SO Linux, esta razón complejiza mucho el proceso, pues el dominio de este es un poco engorroso y gran parte del desarrollo es a través de comandos.
A pesar de que el manual de instalación presenta algunos problemas en el entendimiento de los pasos para su progreso, fue posible instalar satisfactoriamente la aplicación, luego de su desarrollo se emprendió a ejecutar un conjunto de casos de pruebas a las funcionalidades principales de la aplicación para determinar si el sistema trabajaba satisfactoriamente.
De estas se pudo determinar que al cargar un modelo de datos o generar o visualizar un reporte, mientras esta aún no termina de ejecutarse, y se agota el tiempo límite para la conexión, no es posible obtener el resultado deseado.
Pudo determinarse tras su proceso de desinstalación que:
Los paquetes establecidos no pueden ser eliminados en ningún momento, ejemplo:
La no incorporación del paquete php5 no permite ejecutar los ficheros con extensión php.
Apache2 tras ser eliminado no admite que se visualice la aplicación.

Debe verificarse que el paquete apache2 tenga instalado todos sus paquetes adyacentes, pues al eliminarle uno de estos, no es posible mostrar la aplicación.

Par obtener más información sobre las pruebas realizadas remitirse a la planilla de *Casos de Pruebas de Instalación,* que se encuentra en *Materiales complementarios*.

Tras finalizado su proceso de instalación se llevó a desarrollar un conjunto de pruebas funcionales con el fin de medir el cumplimiento de sus requisitos para determinar la existencia de errores en sus ejecución.

Se concluye que el sistema funciona totalmente tras su proceso de instalación. Los resultados obtenidos demuestran que el proceso se realiza de la forma en que se encuentra especificada en el manual de instalación.

3.5 Ejecución y análisis de las pruebas de carga y estrés a la aplicación.

Para el desarrollo de este tipo de pruebas se comenzó a probar de forma funcional la aplicación para determinar de forma visual la rapidez en que se cargan los datos. Se arrojó que aunque se muestran los resultados satisfactoriamente, existen interfaces que se demoran en su proceso de carga.

Como fue especificado en el capítulo anterior el desarrollo de estas pruebas se estarán desarrollando con la herramienta JMeter. Para la realización de estas se inició con el establecimiento de 100 usuarios (hilos) en un período de subida de un segundo.

Las características del entorno donde fue desarrollada las pruebas estuvo determinada por:

PC Cliente: Pentium(R) 4, 3.0Ghz, Memoria RAM de 1.0GB, Disco Duro de 15GB.

Servidor de BD: Pentium(R) 4, 3.0Ghz, Memoria RAM de 1.0GB.

Funcionamiento de la red: 100 Mbps de velocidad de enlace.

Características de la BD: Modelo_SIGE: 38 tablas de las que se encontraban 6 nomencladores (datos fijos), 13 funciones y 9 secuencias.

Estas pruebas fueron desarrolladas a través de la realización de un reporte con una tabla de 300 tuplas.

Tras la ejecución de las pruebas a los diferentes escenarios especificados, fue posible determinar qué:

Para la ejecución de las pruebas en la que se vincularon los escenarios visualizar, exportar y mover reporte teniendo vinculación con la página principal, fue posible exponer que existe un elevado porciento en los niveles de rendimiento como se muestra en la tabla siguiente.

Secciones	# Muestras	Media	Mediana	Mín.	Máx.	% Error	Rendimiento	Kb/sec
Aplicación	9200	4339	63	0	180687	27.14%	21.9/sec	220.5

Tabla 3: Resumen del Informe Agregado obtenido para 100 usuarios.

Por ello se tomó como estrategia seleccionar cada uno de los escenarios de forma independiente para determinar cómo se establece su comportamiento. En siguiente tabla se muestran los valores que resultaron tras la realización de las pruebas a los escenarios exportar y filtrar reporte, pertenecientes al módulo visor de reporte y mover reporte para el módulo administrar.

Secciones	# Muestras	Media	Mediana	Mín.	Máx.	% Error	Rendimiento	Kb/sec
Visor de reporte	3800	3942	94	0	42138	62.97%	24.1/sec	25.1
Administrador de Reportes	3300	4162	94	0	20750	69.45%	22.5/sec	21.7

Tabla 4: Resumen del Informe Agregado obtenido para 100 usuarios.

Como se muestra en la tabla anterior es posible concluir que los valores del rendimiento aún continúan presentando elevado nivel. En la ejecución de esta prueba fue posible determinar tras obtener el árbol de resultado, que las principales peticiones que presentaban errores estaban dadas por los elementos que referenciaban a la interfaz inicial de la aplicación. Es por ello, que se propuso una nueva iteración para la comprobación de los escenarios sin la interacción con esta página.

Una vez ejecutada la realización de las pruebas, los resultados presentados quedaron de la siguiente manera:

Para 100 usuarios:

Secciones	# Muestras	Media	Mediana	Mín.	Máx.	% Error	Rendimiento	Kb/sec
Visor de reporte	1300	10617	4703	0	46313	7.69%	8.9/sec	46.3
Administrador de Reportes	1600	11835	14547	0	25062	12.50%	7.6/sec	1.2

Tabla 5: Resumen del Informe Agregado obtenido para 100 usuarios.

Para 200 usuarios:

Secciones	# Muestras	Media	Mediana	Mín.	Máx.	% Error	Rendimiento	Kb/sec
Visor de reporte	2600	21046	10078	0	176781	9.42 %	8.6/sec	39.9
Administrador de Reportes	3200	20193	20016	0	297980	12.50%	7.9/sec	1.2

Tabla 6: Resumen del Informe Agregado obtenido para 200 usuarios.

Para 300 usuarios:

Secciones	# Muestras	Media	Mediana	Mín.	Máx.	% Error	Rendimiento	Kb/sec
Visor de reporte	3900	26538	10969	0	339573	9.82%	9.0/sec	41.7
Administrador de Reportes	4800	27308	22609	0	333947	13.69%	8.1/sec	1.4

Tabla 7: Resumen del Informe Agregado obtenido para 100 usuarios.

Como bien se muestra los tiempos de respuesta disminuyeron significativamente, considerándose apropiados.

Par obtener más información sobre las pruebas realizadas remitirse a la planilla de *Casos de Pruebas de Carga y Estrés,* que se encuentra en *Materiales complementarios*.

Se llevó a cabo la verificación de los códigos respuesta dadas en el árbol de resultado de los casos mostrados en rojo, en la que fue posible obtener:

Para el administrador de reporte se obtuvo que dos fueron las peticiones http que no pudieron cargarse de manera factible, una con código de respuesta 404, el cual está dado por un recurso no encontrado, lo que indica que el cliente fue capaz de comunicarse con el servidor, pero que no pudo encontrar lo que se solicitó. La otra con un código de respuesta 500, lo que responde a situaciones de error ajenas a la naturaleza del servidor web.

Se hace necesario destacar que los errores encontrados coincidieron para cada prueba desarrollada con los distintos grupos de usuarios. Para ver el árbol de resultado remitirse a anexo 3.

Para verificar la existencia del mayor conjunto de errores en la página principal se estableció la elaboración de la prueba para este escenario únicamente, en la que pudo mostrarse que existe un alto índice del valor del rendimiento.

Secciones	# Muestras	Media	Mediana	Mín.	Máx.	% Error	Rendimiento	Kb/sec
Página Inicio	2300	1121	31	0	21354	94.91%	80.6/sec	112.0

Tabla 8: Resumen del Informe Agregado obtenido para 100 usuarios.

Para la página de inicio se concluyó que la cantidad de peticiones http que no pudieron cargarse de manera factible fueron 12, todos ellas obtuvieron un código de respuesta 404.

Con este resultado fue posible comprobarse la veracidad de lo antes expresado. Pues pudo evidenciarse las diferencias de valores entre los distintos criterios evaluados.

Como resultado de la realización de la prueba puedo comprobarse que el sistema presenta algunos

errores en el tiempo de carda de la aplicación, principalmente en la muestra de su página inicial.
Por otro lado, tras la consulta con clientes del generador pudo determinarse que los tiempos de carga para las secciones del visor y administrador de reportes pueden considerarse aceptables, no es así cuando este interactúa con su página principal.
Por estas razones se puede considerar que el rendimiento del sistema no se encuentra en las mejores condiciones, pero no por ello deja de ser aceptable su condición, pues a pesar de que los tiempos se consideran elevados, no son considerados críticos.

3.6 Ejecución y análisis de las pruebas de volumen a la aplicación.

El desarrollo de las pruebas de volumen arrojó que la aplicación presenta la dificultad, que los reportes cuyos volúmenes de dados son de gran tamaño, el tiempo en presentar los resultados pueden demorar con respecto a otros.
En la tabla se muestran algunos de los resultados de las pruebas aplicadas.

Nombre del escenario	Escenario de la sección	Cantidad de tuplas	Otros elementos	Volumen de datos	Tiempo de carga	Criterio de aceptación
EC1: Visualizar un reporte.	EC1.1: Visualizar Reporte.	34	no	1.5KB	1.75s	Aceptable
		1105	no	47.3KB	8.33s	Aceptable
		8073	no	45KB	20.83s	Aceptable
EC 3: Administrador de Reportes.	EC 3.1: Eliminar reporte.	5	no	34B	265ms	Aceptable
		3	si	34B	234ms	Aceptable
EC 4: Diseñar Reporte	EC 4.1: Generación de reporte estático.	6	no	476B	1.02s	Aceptable
		43	no	865B	3.61s	Aceptable
		1105	no	21.5KB	7.92s	Aceptable
		8073	no	45KB	20.16s	Aceptable

Tabla 9: Resultado de las pruebas de volumen.

Para obtener el conjunto de pruebas desarrolladas remitirse a *Pruebas de Volumen que se encuentra en los Materiales complementarios.*
Es válido destacar que la aplicación tras la posibilidad que brinda de poder integrarse a diferentes sistemas gestores de base de datos así como a una base de datos determinada, puede encontrase afectada tras el gestor que se utilice, es por ello que para un rendimiento y mejor facilidad de uso de la

aplicación se hace necesario un servidor que cumpla con las expectativas dadas, así como un gestor integral y bien diseñado.

3.7 Análisis de las pruebas funcionales de la aplicación.

El desarrollo de las pruebas funcionales fue ejecutado por el equipo de pruebas del departamento de Calisoft, en el proceso de liberación en que se encuentra el producto. Hasta el momento han sido desarrolladas 2 iteraciones. En su primera iteración el porciento de no conformidades detectadas ocupó un elevado por ciento con respecto a su segunda evaluación. Figura 9

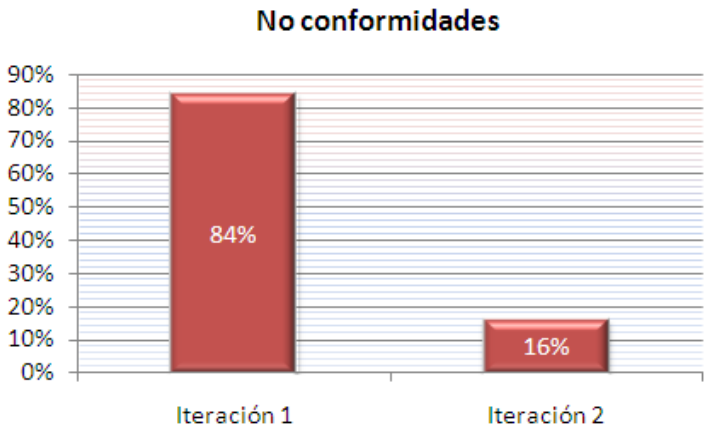


Figura 9: Porciento de no conformidades

De forma general se determinó que los principales errores detectados tras su primera iteración estuvieron enmarcado en errores funcionales, en donde el módulo *Diseñar reporte* ocupó la mayor puntuación.

Tras analizar los resultados, fue posible comprobar que a pesar de haber erradicado los problemas existentes en la primera iteración, mostrando un elevado descenso de las no conformidades detectadas, es necesario destacar el gran número de errores obtenidos.

El desarrollo de un proceso debe encontrarse enmarcado por la aplicación de metodologías que guíen todo el ciclo de vida de la aplicación. En el cual la ejecución de pruebas constituye una parte importante en todo su recorrido. Tras detectar que el conjunto de no conformidades se encuentran en los elementos funcionales del sistema es posible deducir que existen grandes problemas también en las pruebas de unidad, evidencia de ello es la no incorporación de las pruebas de unidad en el

departamento, como un punto imprescindible tras el comienzo de la implementación. Con las pruebas unitarias, la mayoría de los errores de programación se detectan durante la propia etapa de programación, lo que tiene gran valor, ya que mientras más tiempo permanezca un error en el sistema, más tiempo requerirá eliminarlo y más repercusiones acarrearán en otras secciones del programa. Es por ello que se ha establecido un conjunto de elementos que se hace necesario para la elaboración de las pruebas de unidad y la comprobación eficiente de la mayor parte de la codificación del sistema:

- Se prueba la interfaz del módulo para asegurar que la información fluye apropiadamente hacia dentro y hacia fuera de la unidad de programa sujeta a prueba.
- Se examinan las estructuras de datos locales para asegurar que los datos temporales mantienen la integridad durante todos los pasos de la ejecución de un algoritmo.
- Se recorren todos los caminos independientes en toda la estructura para asegurar que todas las instrucciones de un módulo se hayan ejecutado por lo menos una vez.
- Se prueban todos los caminos de manejo de errores. [26]

3.8 Evaluación de la aplicación.

Una vez realizado el proceso de pruebas del software se llevó a cabo la evaluación del sistema teniendo en cuenta los atributos de calidad siguientes: eficiencia, usabilidad, portabilidad y fiabilidad.
Para efectuar dicho proceso se utilizó una lista de chequeo como mecanismo conducente al control de riesgos.
Esta lista contiene una serie de preguntas cuyas respuestas consisten en una evaluación que se le confiere a cada uno de estos parámetros de calidad por separado. Dicha evaluación se concede en dependencia del grado de disponibilidad de las propiedades medidas, según las preguntas.
Una puntuación de 1 del parámetro evaluado significa que la propiedad se encuentra en su estado crítico, es decir, un parámetro mal implementado. Una puntuación de 2 representa que aún sigue habiendo algunos fallos. Una puntuación de 3 representa una disponibilidad parcial de la propiedad, es decir, en parte son satisfactorios los resultados observados pero sigue habiendo fallos en algunos casos. En cambio una calificación de 4 describe un parámetro bien implementado. El 5 es para la excelencia, aquel parámetro que no tenga ningún señalamiento negativo y no presente ningún fallo.
Una vez aplicadas las preguntas al software, se sumaron los valores asignados y se dividió por la cantidad total de estas; así se obtuvo un valor general para cada atributo evaluado, que este ocupó un lugar en un rango determinado, mostrando la eficiencia y validez del software (Ver Tabla 10):

Evaluación	Clasificación
1-1.9	No disponible
2.0-3.9	Parcialmente disponible
4-4.9	Disponible
5	Muy bien implementada

Tabla 10: Cálculo de disponibilidad del Software.

3.8.1 Eficiencia

En el GDR se definieron para este atributo 2 preguntas. Luego de su aplicación arrojó un valor de 3.5 puntos tras promediar los resultados, llegando a la conclusión de que este atributo se encuentra parcialmente disponible. Tal calificación estuvo dada por la aplicación de las pruebas de carga y estrés en la que se puedo determinar que existen algunos problemas en su tiempo de carga, A pesar de que no es de elevada relevancia los resultados obtenidos, si tiene influencia a la hora de establecer un criterio final. Los valores obtenidos de las interrogantes se detallan en la siguiente tabla.

Característica de calidad: Eficiencia							
Total de Preguntas	Evaluadas de:					Promedio	Evaluación según rango
	1	2	3	4	5		
2	-	-	1	1	-	3.5	Parcialmente disponible

Tabla 11: Valores del atributo eficiencia.

Para obtener mayor información sobre la aplicación de las listas de chequeo para este atributo, remitirse a *"Listas de chequeo para atributos de calidad",* que se encuentra en los *Materiales complementarios.*

Calificación de 3:

1. ¿La velocidad de los tiempos de respuesta y el tratamiento y las tasas de rendimiento se realizan en función de los objetivos?

Calificación de 4:

2. ¿La cantidad de recursos utilizados y la duración de esos usos en el desempeño de su función es buena?

3.8.2. Fiabilidad

Para este atributo se definió un total de 5 preguntas, las que arrojó un promedio de 4.4 puntos, por lo que es posible establecer que la **aplicación es fiable**. En la siguiente tabla se detallan los valores.

Característica de calidad: Fiabilidad							
Total de Preguntas	Evaluadas de:					Promedio	Evaluación según rango
	1	2	3	4	5		
5	-	-	-	3	2	4.4	Disponible

Tabla 12: Valores del atributo fiabilidad.

Para obtener mayor información sobre la aplicación de las listas de chequeo para este atributo, remitirse a *"Listas de chequeo para atributos de calidad",* que se encuentra en los materiales complementarios.

Calificación de 4:

1. ¿La recuperación ante fallas se realiza en el tiempo requerido para la aplicación?
2. ¿Se recupera el software ante fallas?
3. ¿Se carga correctamente la aplicación?

Calificación de 5:

1. ¿La aplicación realmente muestra lo que pretende?
2. ¿Los especialistas pueden establecer análisis a partir de los resultados que ofrece la aplicación?

3.8.3. Portabilidad

En el GDR se definió para este atributo 1 pregunta, la cual arrojó un valor de 4.0 puntos. Con ello se llega a la conclusión de que este atributo a pesar de no haber adquirido la máxima puntuación la aplicación se encuentra **portable**. Lo antes explicado se detalla en la siguiente tabla.

Característica de calidad: Portabilidad							
Total de Preguntas	Evaluadas de:					Promedio	Evaluación según rango
	1	2	3	4	5		
1	-	-	-	1	-	4.0	Disponible

Tabla 13: Valores del atributo portabilidad.

Para obtener mayor información sobre la aplicación de las listas de chequeo para este atributo, remitirse a *"Listas de chequeo para atributos de calidad"*, que se encuentra en los materiales complementarios.

Calificación de 4:

1. ¿Existe independencia del ambiente de software? (sistema operativo, lenguaje de programación)

3.8.4. Usabilidad

En el GDR se definieron para este atributo 25 preguntas. Luego de su aplicación no se obtuvo la máxima puntuación, al promediar arrojó un valor de 4,4 puntos, manteniéndose en el rango máximo, llegando a la conclusión de que este atributo está disponible, aunque no en un 100% de especificidad, se considera una aplicación usable**.** Lo antes explicado se detalla en la siguiente tabla.

Característica de calidad: Usabilidad							
Total de Preguntas	Evaluadas de:					Promedio	Evaluación según rango
	1	2	3	4	5		
25	-	1	2	6	16	4.4	Disponible

Tabla 14: Valores del atributo usabilidad.

Para obtener mayor información sobre la aplicación de las listas de chequeo para este atributo, remitirse a *"Listas de chequeo para atributos de calidad",* que se encuentra en los materiales complementarios.

Calificación de 2:

1. ¿Existen atajos de teclado bien hechos?

Calificación de 3:

1. ¿Se permite al usuario personalizar la interfaz?
2. ¿Se reconoce todas las tareas del software en cualquier momento?

Calificación de 4:

1. ¿Resulta fácil instalar el software?
2. ¿Resulta fácil especificar un objeto o una acción?

3. ¿Resulta fácil entender el resultado de una acción?
4. ¿Actúa el sistema en la prevención de errores?
5. ¿Permite distintos niveles de uso del producto para usuarios con distintos niveles de experiencia?
6. ¿Existen diferentes niveles de ayuda?

Calificación de 5:

1. ¿Es simple el vocabulario utilizado?
2. ¿Se proporciona tiempo suficiente para realizar las entradas por teclado?
3. ¿Se posibilita la opción de incrementar el tamaño de los caracteres?
4. ¿Hay algún tipo de asistencia para los usuarios que hacen uso del sistema por primera vez?
5. ¿Se entienden la interfaz y su contenido?
6. ¿Está diseñada la interfaz para facilitar la realización eficiente de las tareas de la mejor forma posible?
7. ¿Es fácil de utilizar el sistema en la realización de tareas?
8. ¿Se muestran todas las imágenes y marcos correctamente?
9. ¿Es apropiada la retro alimentación presentada por el sistema?
10. ¿Actúa el sistema en la información de los errores?
11. ¿Se usan adecuadamente los modos de trabajo de los usuarios en el software?
12. ¿Se utiliza mensajes y textos descriptivos?
13. ¿Permite deshacer las acciones e informar el estado?
14. ¿Permite una cómoda navegación dentro del producto y una fácil salida de éste?
15. ¿Se proporciona información visual de dónde está el usuario, qué está haciendo y qué puede hacer a continuación?

16. ¿Se presenta al usuario la información que sólo necesita?

Muchas de las preguntas destacadas en la lista de chequeo para el atributo de usabilidad, fueron utilizadas en la realización de una encuesta a personal de nuestra universidad, en la que todas ellas han interactuado con el Generador, mayormente a través de productos que lo utilizan para su desempeño. El objetivo de esta, fue determinar la asimilación que ha presentado el sistema en los usuarios, y cuáles son las principales dificultades que se presentan. De esta forma se pretende elevar la calidad de este teniendo en cuenta los resultados obtenidos, y de esta forma mejorar la satisfacción de los usuarios en posteriores versiones.

Algunos de las no conformidades emitidas por los encuestados fueron:

- Se evidencia más el logo del centro.
- El logo no identifica al producto No brinda una ligera idea de ¿Qué es el producto?
- Cuando se navega por los diferentes módulos se pierde el logo y solo se vuelve a observar si se va al Inicio de la aplicación.
- Los mecanismos para ponerse en contacto con la empresa lo ofrece pero no de forma intuitiva, teniéndose que recorrer toda la página en busca del correo electrónico, que a su vez no es muy visible.
- El trabajo se vuelve lento cuando se utiliza esta herramienta.
- Le faltan muchas funcionalidades, como por ejemplo, poner varios grupos que no estén uno dentro de otro, poner tablas resúmenes, llamar a varias funciones en un reporte, ponerle rayas horizontales a los reportes, ajustar el reporte al tamaño de la letra si se le dice no truncar, realizar subtotales donde el subtotal esté encima y no debajo, en la integración los parámetros de entrada se envían por la url penando la seguridad, entre otras funcionalidades que deberían mejorar.
- Se desea la adición de:
  1- Una mejor forma de instalación.
  2- Una ayuda insertada en la aplicación.
  3- Un buscador para cuando se estén seleccionando las tablas de la Base de datos necesarias para trabajar.

Fueron muchos los criterios emanados a partir de la realización de la encuesta, donde pudo reflejarse que aún existen irregularidades que afectan la completa conformidad de los usuarios con respecto al sistema, los cuales manifestaron las inconformidades que poseen, pero no por ello puede determinarse que el sistema fue de rechazo total por parte del usuario, pues la mayor parte de las preguntas desarrolladas obtuvieron la aceptación de los encuestados. Es por ello que la calificación dada

coincide con la propiciada tras la aplicación de la lista de chequeo, quedando como resultado final que la aplicación es considerada **usable.**

Para tener conocimientos sobre los resultados de las encuestas realizadas, remitirse a *Materiales complementarios* en *Encuesta de Usabilidad del GDR*.

Haciendo un análisis de los resultados expuestos anteriormente se puede llegar a las siguientes conclusiones:

- ✓ El sistema GDR es una aplicación parcialmente eficiente.
- ✓ El sistema GDR es una aplicación fiable.
- ✓ El sistema GDR es una aplicación portable.
- ✓ El sistema GDR es una aplicación usable.

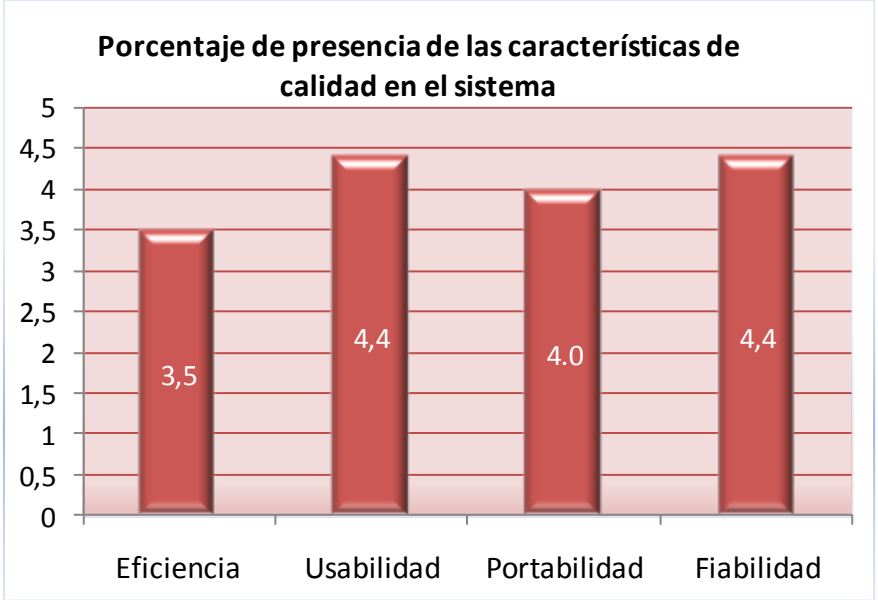


Figura 10: Presencia de las características de calidad en el sistema.

3.9 Conjunto de mejoras.

Estas son tan solo una serie de sugerencias a usar en el entorno de trabajo con vista a mejorar la calidad del Generador Dinámico de Reportes en futuras versiones, tras el desarrollo de los epígrafes anteriores fueron brindados de igual forma un conjunto de consejos a tener en cuenta.

- Análisis
  - Perfeccionar los requisitos no funcionales de la aplicación, incluyendo requisitos de volumen, características de servidores y rendimiento del sistema.

- Diseño
  - Elaborar un logotipo más representativo a lo que hace el producto.
  - Elaborar encuestas en las que permitan determinar los diferentes criterios de los clientes sobre las principales funcionalidades que desean obtener, y a partir de estas elaborar el proceso de diseño.

- Codificación
  - Llamar a las cosas por su nombre: nombres de variables y funciones fáciles, cortas y legibles.
  - Comenta tanto como necesites: suelen ser mensajes a otros desarrolladores, y formas de recordatorio si se desea interactuar con el código después de un tiempo trabajando en otras cosas).
  - Revisar el código y el diseño del trabajo realizado semanalmente. Las revisiones deben incluir comprobaciones de compilador, pruebas unitarias, documentación del diseño y documentación de resultados.
  - Incorporar revisores del código como parte del equipo de trabajo, los que constituirán personal ajeno a los desarrolladores del sistema.
  - Debe dedicar a la revisión la mitad del tiempo que ha tardado en escribir el código o en planear el diseño originalmente. Permitir que los revisores controlen la revisión, donde sus comentarios controlen la revisión.
- Pruebas
  - Contar con una suite de pruebas de unidad para mantener la salud del código.
  - Incorporar un proceso de medición de código, incluyendo la utilización de herramientas para su desarrollo.
  - Se hace mención de algunos de los procesos a tener en cuenta, con algunas de las herramientas que se utilizan:
    - Realización de pruebas unitarias automatizadas: JUnit, PHPUnit, etc.
    - Medir la cobertura de código (de líneas y desviaciones): Spike PHPCoverage.
    - Utilizar una herramienta de análisis estático de código como PMD.
- Instalación

- Diseñar un instalador para el Generador de Dinámico de Reportes, cumpliendo con la mayoría de los factores que facilitan la instalación del producto. Este permitiría obtener un producto fácilmente instalable por un usuario final ya sea un usuario común o experimentado.

❖ Documentación

- Elaborar un manual de instalación en la que especifiquen de forma clara y detallada cada uno de los procedimientos a llevar a cabo en todo el proceso.

Conclusiones parciales

Con este capítulo concluye el proceso de pruebas diseñado y aplicado al Generador Dinámico de Reportes. El desarrollo del mismo permitió obtener el resultado de las pruebas que se aplicaron mostrando la efectividad de estas. Fueron aplicadas un conjunto de listas de chequeo permitieron desglosar los diferentes indicadores del sistema, obteniendo una clasificación en dependencia de su comportamiento. Así como el establecimiento de encuestas que permitieron determinar el grado de asimilación del producto por parte de los usuarios. Dado los resultados obtenidos fue posible determinar que el producto a pesar de que presenta algunas inconformidades, puede determinarse como un producto usable.

CONCLUSIONES

Con la realización de este trabajo se logró desarrollar una evaluación del producto GDR en su versión 1.7.0, para el cual se llevó a cabo una revisión técnica de la documentación, se diseñaron y ejecutaron un conjunto de pruebas que permitieron obtener un software con un número reducido de defectos, a partir de los resultados obtenidos se pudo establecer una valoración de cada uno de ellos, la cual ayudó a evaluar al software a través de diferentes atributos de calidad.

Con este trabajo de manera general se arribaron a las siguientes conclusiones:

- Se diseñaron todos los casos de prueba a partir de las diferentes técnicas a aplicar, logrando con estos probar los requerimientos no funcionales con los que cuenta el software.
- Se aplicaron el conjunto de pruebas tanto para la documentación como para el sistema, mostrando algunos errores por ambas partes:
  - En el caso de la revisión técnica de la documentación se pudo obtenerse pequeños errores en la redacción y estructuración de la misma.
  - Mientras que en el sistema, los principales errores estuvieron enmarcado en el tiempo de carga del sistema.
- La evaluación realizada permitió obtener la medida del estado del sistema, en la que se estableció como conclusión que el sistema puede considerarse usable. Los defectos detectados podrán ser analizados posteriormente por el equipo de desarrollo que evitará la permanencia de estos en futuras versiones. Además fue propuesto un conjunto de mejoras que permitirán elevar la calidad del producto.

Con el estudio realizado y la aplicación de las pruebas que se han llevado a cabo dentro del proceso evaluación del producto, se ha cumplido el objetivo propuesto de desarrollar un proceso de pruebas que permita establecer una evaluación del sistema con vistas a tener conocimiento del estado en que se encuentra el GDR, erradicando los problemas existentes en futuras versiones y por consiguiente obtener mayor calidad.

RECOMENDACIONES

Luego de haber concluido con el proceso de evaluación del producto GDR, y haber demostrado la importancia de la detección de errores a tiempo y la aplicación de las pruebas como vía imprescindible en el desarrollo del software, se recomienda:

- Establecer un equipo de aseguramiento de la calidad, en la que establezca un control de esta para todas las fases del software.
- Incorporar la ejecución de pruebas de unidad a través de los grafos de flujo, que permitirán determinar 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.

BIBLIOGRAFÍA

1. **Anna** C., **Grimán** M. P. (2005). Estrategia de Pruebas para Software OO que garantiza Requerimientos No Funcionales. Caracas, Venezuela: Departamento de Procesos y Sistemas. Universidad Simón Bolívar.
2. **Antonio Olsina,Luis.** Tesis Doctoral" Metodología Cuantitativa para la Evaluación y Comparación de la Calidad de Sitios Web", Facultad de Ciencias Exactas, Universidad Nacional de La Plata – Argentina, Noviembre de 1999. http://gidis.ing.unlpam.edu.ar/home/downloads/pdfs/Web-site_QEM_VF.pdf
3. Breves notas sobre la Medición de los Atributos Externos del Software. http://www.sc.ehu.es/jiwdocoj/mmis/externas.htm.
4. **Brito, Irina Reyes.** Las pruebas de software, su aplicación a Config. CASE. Ciudad de La Habana.2003.
5. **Caballano Alcántara, José Luis**. *El prisma*. Gestión de la Calidad. http://www.elprisma.com/apuntes/administracion_de_empresas/gestiondelacalidad/.
6. **Cristancho, José Alberto.** Evaluación de la calidad del software educativo bajo el estándar ISO 9126. Instituto Universitario de Tecnología Región los Andes. http://ecotropicos.saber.ula.ve/db/ssaber/Edocs/pubelectronicas/evaluacioninvestigacion/vol2num1/articulo3.pdf.
7. **Díaz J. G.** Introducción al Proceso de Pruebas. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla., Sevilla.
8. Diccionario. *Diccionario Informático.* http://www.lawebdelprogramador.com/diccionario/.
9. **Escobar, L. M**. La calidad del software y su importancia en el mercado. 2003
10. **Fernández Carrasco, Oscar M., García León, Delba and Beltrán Benavides, Alfa**. Un enfoque actual sobre la calidad del software. http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.
11. **Fernández Sanz, Luis, Alarcón Rodríguez, Miren Idoia**. Necesidades de medición en la gestión y el aseguramiento de calidad del software. http://www.sc.ehu.es/jiwdocoj/remis/docs/aseguracal.htm.
12. **Gónzalez C.** Un plan de Pruebas Exitoso. 2002. http://www.americaxxi.cl/modules.php?name=News&file=article&sid=20.
13. **Guzmán, Cortés and Hernando, Oscar.** Aplicación práctica del diseño de pruebas de software a nivel de programación. abril/2004. http: //www.scribd.com/doc/36530170/oguzman-diseno-pruebas, Scribd.
14. **Hugo Vázquez, Roberto**. Taller de Calidad de Software: Introducción a la Calidad de Software.

http://gridtics.frm.utn.edu.ar/docs/Introduccion%20a%20la%20Calidad%20de%20Software%20Vazquez.pdf.

15. **IEEE**. Computer Dictionary. Computer Society. 1990.
16. **Ingeniería del Software II**. Curso 2010 – 2011. Tema 2: Fases de Construcción. Semana 10 y 11. Materiales Básicos y Complementarios.
17. **Vinicio León, Luis, Castillo, Abraham, Ruelas, Elena and Moreno, Aaron.** Software Guru. noviembre-Diciembre de 2006. http://www.sg.com.mx/content/view/347/.
18. **Mauricio.** DISEÑO DE CASOS DE PRUEBA. 2006. http://my.opera.com/pelican0/blog/show.dml/564829.
19. **Natalia Juristo, A. M.**. TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 2007.
20. **Nielsen, J.** "Usabilidad. Diseño de sitios Web". Prentice Hall. 2000.
21. **Ortega, M. Pérez, M. and Rojas, T.** UN ENFOQUE SISTÉMICO PARA EVALUAR EL PRODUCTO DE SOFTWARE. http://www.lisi.usb.ve/publicaciones/02%20calidad%20sistemica/calidad_27.pdf.
22. Revista Española de Innovación, Calidad e Ingeniería de software, Vol. 4, No. 2, 2008, "La Norma ISO/IEC 25000 y el proyecto KEMIS para su automatización con software libre".
23. **Rojas, Johanna and Barrios, Emilio.** *Investigación sobre estado del arte en diseño y aplicación de pruebas de software*. 2007. http://www.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html.
24. **Pressman Roger S**. Ingeniería del software. *Un enfoque práctico.* Cuarta Edición. McGraw-Hill. 1998
25. **Pressman, Roger S**. Ingeniería del Software. *Un enfoque práctico.* Quinta edición. McGraw-Hill. 2002.
26. **Saavedra López, Lesdey and Pupo Blez, Yohandris**. Diseño y aplicación de pruebas al producto Registro Cubano de Discapacitados. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, 2007.
27. **Sánchez Almenares, Liudmila**. Cómo Realizar Pruebas de Carga y Estrés en JMeter. Manual, 08/05/2008, UCI.
28. **Sánchez Almenares, Luidmila**. Prueba Automática de Carga y Estrés en el Proyecto CICPC. Trabajo de Diploma para optar por el título en Ciencias Informáticas. Universidad de las Ciencias Informáticas, Ciudad de la Habana, 2008.
29. **Sommerville**, **I**. Ingeniería del software. Séptima Edición. Madrid: Pearson Educación S.A. 2005.
30. **Torbio Vicente, Jose María**. Tutorial JMeter. 2005. http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jmeter.

31. Universidad Simón Bolívar. Ingeniería de Software 3.*El plan de Prueba*. Enero-Marzo de 2001. http://www.ldc.usb.ve/~teruel/ci4713/clases2001/planPruebas.html.

REFERENCIAS BIBLIOGRÁFICAS

1. **Fernández Carrasco, Oscar M., García León, Delba and Beltrán Benavides, Alfa**. “Un enfoque actual sobre la calidad del software”, septiembre-diciembre de1995. http://bvs.sld.cu/revistas/aci/vol3_3_95/aci05395.htm.
2. **Hugo Vázquez, Roberto**. Taller de Calidad de Software: Introducción a la Calidad de Software. http://gridtics.frm.utn.edu.ar/docs/Introduccion%20a%20la%20Calidad%20de%20Software%20Vazquez.pdf.
3. **Cortés Guzmán, Hernando Oscar**. *Scribd*. Aplicación práctica del diseño de pruebas de software a nivel de programación. abril de 2004. http: //www.scribd.com/doc/36530170/oguzman-diseno-pruebas
4. **Mancini, Pablo**. *Educar*. Sociedad de la Información. Programa sobre la informatización de la sociedad cubana. marzo de 2004. http://www.educ.ar/
5. **Crosby, Philip**. La calidad del Software, julio de 2010, Javierpello. http://www.softqanetwork.com/philip-crosby-y-la-calidad-del-software.
6. *Calidad de Software,* enero de 2011. http://www.ecured.cu/index.php/Calidad_de_Software.
7. **Pressman, Roger S.**, Ingeniería del Software. Un enfoque práctico. 5ta edición.2002.
8. Revista Española de Innovación, Calidad e Ingeniería de software, Vol. 4, No. 2, 2008, “La Norma ISO/IEC 25000 y el proyecto KEMIS para su automatización con software libre”.
9. Proyecto de Ingeniería Web, Procesos de las Aplicaciones Web sobre la “Calidad de las Aplicaciones Web”. http://zarza.usal.es.
10. **Cueva Lovelle, Juan Manuel**. Conferencia. Grupo GIDIS. Universidad Nacional de la Pampa: ¨Calidad del Software¨.21 de octubre de 1999. http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF.
11. *Cig_Labs*. The Home of Groundbreaking Software Quality. 2002. http://:www.cigitallabs.com/resources/definitions/software_testing.html.
12. **IEEE** Standard Glossary of Software Engineering Terminology, 1990.
13. **Alonso Amo, F. and Martínez, Normand**. Loïc. Introducción a la ingeniería del software. http://books.google.com.cu.
14. *Buenas tareas*, Pruebas De Software, 02 de 2011. http://www.buenastareas.com/ensayos/Pruebas-De-Software/1539716.html.
15. **Brito, Irina Reyes**. Las pruebas de software, su aplicación a Config. CASE. Ciudad de La Habana.2003.
16. *Técnicas de prueba de software.* http://www.cetic.guerrero.gob.mx/pics/art/articles/113/file.TiposPruebasSoftware.pdf

17. **Rojas Johanna and Barrios, Emilio**. Grupo ARQUISOFT. Métodos de prueba de caja negra. 2007. http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node28.html.
18. **Blank, Isabel, Herrera, Larissa and Ortiz, Miguel. Pruebas** de Funcionalidad. mayo de 2005. http://carolina.terna.net/ingsw3/datos/Pruebas_Funcionales.pdf.
19. **Vasallo Artigas, Onaysi, Ramírez Camejo, Yislén Dolores**. Proceso de Pruebas de Liberación al Sistema de Manejo de Datos de Ensayos Clínicos Cubano. Trabajo de Diploma para optar por el título en Ciencias Informáticas. Universidad de las Ciencias Informáticas, Ciudad de la Habana, 2009.
20. *Sguia*, "QALoad", 2009. http://www.sg.com.mx/guia/node/422
21. **Sánchez Almenares, Luidmila**. Prueba Automática de Carga y Estrés en el Proyecto CICPC. Trabajo de Diploma para optar por el título en Ciencias Informáticas. Universidad de las Ciencias Informáticas, Ciudad de la Habana, 2008.
22. *Scrum-QA*, Pruebas de Stress/Jmeter. http://scrum-qa.blogspot.com/2010/09/pruebas-de-stressjmeter.html.
23. **Pérez B., Pittier A. and Travieso M, Wodzislawski M.** "Testing exploratorio en la práctica." http://www.ces.com.uy/documentos/JIISIC-2007.pdf.
24. **Sommerville, Ian**. Ingeniería del software, cap 23. 2005. http://books.google.com.cu/.
25. Elaboración Manual De Usuario. http://www.mitecnologico.com/Main/ElaboracionManualDeUsuario.
26. Estrategias de pruebas del software. 2011. http://www.dataprix.com/ca/node/5683.

ANEXO

Anexo 1:

1) **Funcionalidad del módulo: *Diseñador de Modelo.***	2) **Funcionalidad del módulo: *Diseñador de Reportes.***

class buildModelAction extends abstractBuildModelAction
{
public function execute($request)
{
1 $error = null;
try
{
2 $ds = $this->getUser()->getAttribute('dataSource');
2 $error = $this->testDS($ds);
3 if ($error != NULL)
4 return $this->renderText("{success:false, errors:{ Message:'$error'}}");
5 $xml = $this->createXmlSchema($request);
5 $model = new Model();
5 $model->setIddatasource($ds->getIddatasource());
5 $model->setXmlmodel($xml);
5 $name = $request->getParameter('name');
5 $model->setName($name);
5 $error = ModelMessages::$ServidoDeBdNoEncendido;
6 if (!Model::validateName($name))
7 return $this->renderText("{success:false, errors:{ Message:'Ya existe un modelo llamado
	class addReportAction extends gdrAction {
	public function execute($request) {
	1 $error="";
	try {
	2 $overwrite=$this->getUser()->getFlash('OVERWRITE');
	2 $iduser = $this->getUser()->getAttribute('iduser', 6);
	2 $title = $this->getUser()->getFlash('TITLE');
	2 $path = $this->getUser()->getFlash('XML_PATH');
	2 $query = $this->getUser()->getFlash('SQL');
	2$dsName = $this->getUser()->getFlash('DSNAME');
	2 $xmltemplate = $this->getUser()->getFlash('TEMPLATE');
	2 $category = $this->getUser()->getFlash('CATEGORY');
	2 $idmodel = $this->getUser()->getFlash('IDMODEL');
	3 if ($overwrite) {
	4 $rep=Report::getReportByName($title);
	} else {

```
$name.'}}');
        $model->save();
        $this->getUser()->setAttribute('dataSource',
null);
    $this->getUser()->setAttribute('schema', null);
        return $this->renderText('{success:true}');
      }    catch (Exception $e)
     {
        return $this->renderText("{success:false,
errors:{ Message:'" . $error . "'}}");
      }
    }
}
```

```
            $rep=new Report();
          }
            $con=Propel::getConnection();
        if ($iduser==NULL){
            $error="Identificador del usuario
no v&aacute;lido";
            return $this-
>renderText("{success:false, msg:'".$error."',
type:'error'}");
          }
        $rep->setCreatebyid($iduser);
            $rep->setModifiedbyid($iduser);
        $xml = $this-
>normalize_to_utf8(file_get_contents($path));
        $rep->setXmlreport($xml);
      unlink($path);
       $rep->setQuery($query);
        $rep->setTitle($title);
        $rep->setParentid(0);
        $rep->setModifiable('FALSE');
        $rep->setTemplate($xmltemplate);
        $error="Error al salvar el reporte";
        $rep-
>salvar($dsName,$con,$overwrite,$iduser,$categ
ory,$idmodel);


return $this-
>renderText("{success:true,msg:'Reporte
guardado satisfactoriamente.',
```

```
type:'information'}");
   14          }catch (Exception $e){

   15     return $this->renderText("{success:false,
msg:'".$e->getMessage()."'}, type:'error'");
   16        }
   17   }
   }
```

3) Funcionalidad del módulo: *Administrador de Reportes.*

```
class showReportAction extends sfAction
{
        public function execute($request)
        {
            1   $this->getResponse()-
>setHttpHeader('Content-Type', 'application/json;
charset=UTF-8');
              try
              {
            2  $id = $request->getParameter('node');

            2  $actions = explode('#',$id);
            2  $index = count($actions) - 1;
            2  $category = $actions[$index - 1];
            3  switch ($actions[$index])
               {
            4     case "first":
               $categories = $this->Categories();

               $reports = $this-
```

4) Funcionalidad del módulo: *Diseñador de Reportes.*

```
class existReportAction extends sfAction
{
 public function execute($request)
 {
   1   if ($request->hasParameter('name'))
   2     $title = $request->getParameter('name');

   3     if (Report::ExistName($title))
   {
   4   return $this->renderText("{success:false,
msg:'Ya existe un reporte con ese nombre.
¿Desea sobrescribirlo?', type:'information'}");
   }
   else
   {
   5   return $this->renderText("{success:true}");
   6   }
   7 }
}
```

```
>reportsByCategory($category);
                $result =
array_merge($categories,$reports);
                return $this-
>renderText(json_encode($result));

        5   case "category":
                $categories = $this->Categories(false,
$category); //Subscateorias de la solicitada
                $reports = $this-
>reportsByCategory($category);    //Reportes de la
categoria solicitada
                $result =
array_merge($categories,$reports);
                return $this-
>renderText(json_encode($result));

        6               default :
                return $this->renderText("{failure: true}");
                    }
        }
        7   catch(Exception $e)
        {
        8 return $this->renderText("{success:false,
errors:{ Message:'". $e->getMessage()."'}}");
        9 }
  }     10
}
```

Anexo 2:

Lista de Comprobación del Código

Sistema de Información de Gobierno

Generador Dinámico de Reportes

V 1.7.0

Consultantes:

#	Nombre y Apellido	Ocupación	Categoría	Fecha Realizada
1	Yadrian Serrano	Desarrollador	Estudiante	07/04/2011
2	Miguel Lescano	Desarrollador	Profesor	07/04/2011
3	Aurelio Rodriguez	Desarrollador	Profesor	08/04/2011
4	Yoander Iñiguez	Desarrollador	Profesor	13/04/2011
5	Yasmany Hernández	Arquitecto	Profesor	13/04/2011
6	Orelvi Gazquez	Desarrollador	Estudiante	28/04/2011

Una puntuación de 1 del parámetro evaluado significa que la propiedad se encuentra en su estado crítico, es decir, un parámetro mal implementado. Una puntuación de 2 representa que existen fallos en algunos elementos. Una puntuación de 3 representa una disponibilidad parcial de la propiedad, es decir, en parte son satisfactorios los resultados observados pero sigue habiendo fallos en algunos casos. En cambio una calificación de 4 describe un parámetro correctamente implementado. El 5 es para la excelencia, aquel parámetro que no tenga ningún señalamiento negativo y no presente ningún fallo.

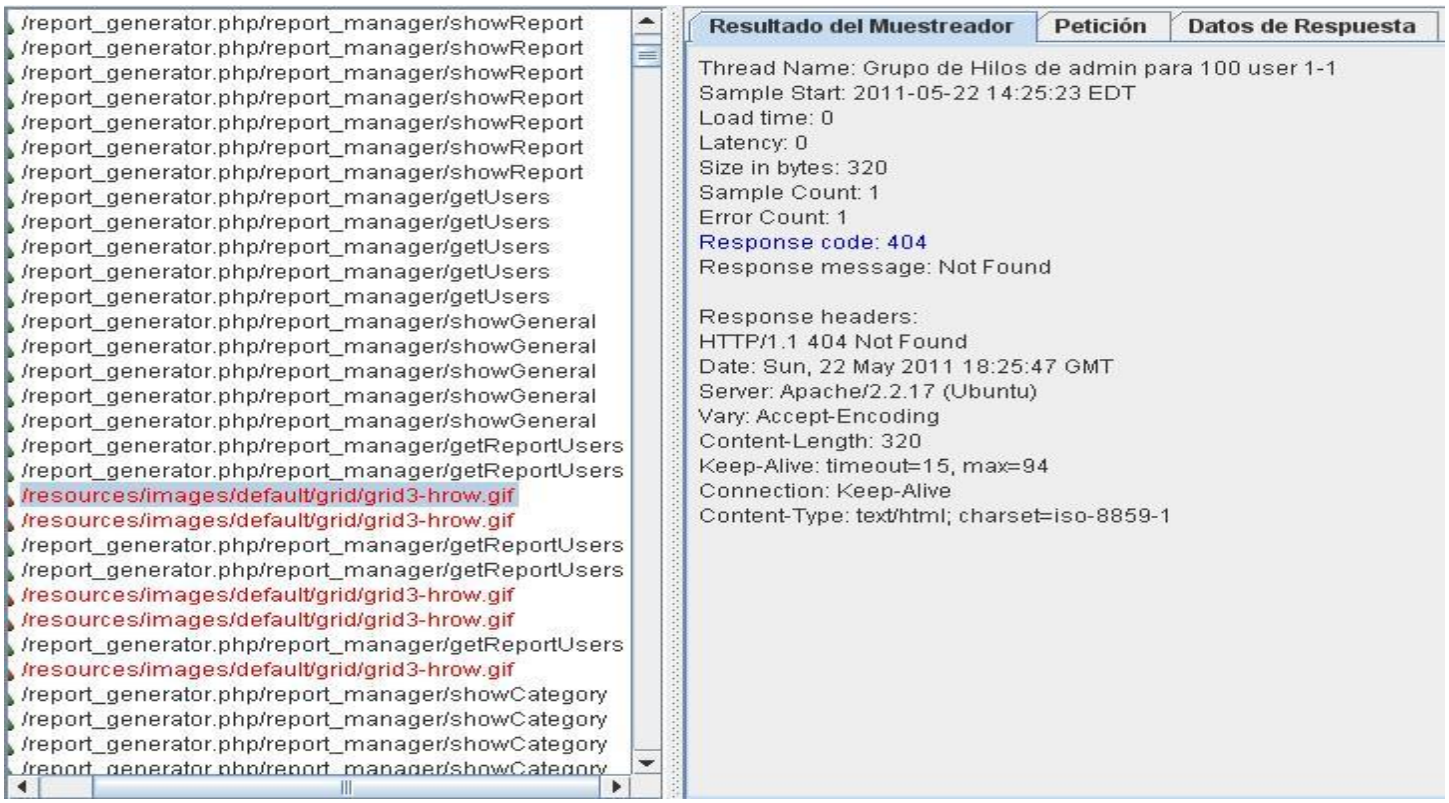
Indicadores a evaluar	Evaluaciones					
	1	2	3	4	5	6

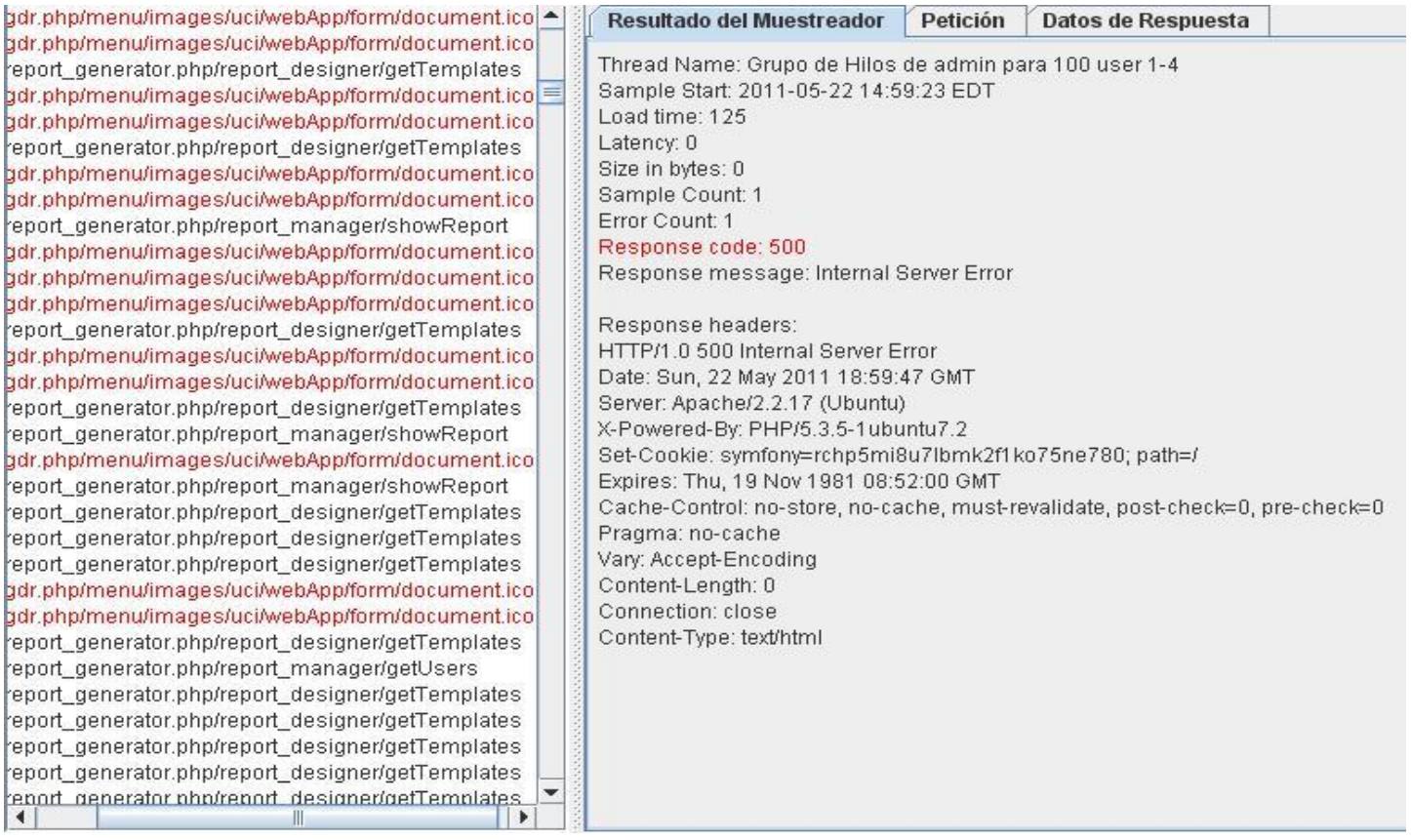
1. ¿Es correcta la lógica del programa?	5	5	5	4	4	5
2. ¿Está completa la lógica del programa?, es decir, ¿está todo correctamente especificado sin faltar ninguna función?	5	5	5	4	3	5
3. ¿Es igual el número de parámetros recibidos por el módulo a probar al número de argumentos enviados?, además, ¿el orden es correcto?	5	5	5	4	5	5
4. ¿Los atributos (por ejemplo, tipo y tamaño) de cada parámetro recibido por el módulo a probar coinciden con los atributos del argumento correspondiente?	5	5	5	5	4	5
5. ¿Coinciden las unidades en las que se expresan parámetros y argumentos? Por ejemplo, argumentos en grados y parámetros en radianes. ¿Altera el módulo un parámetro de sólo lectura?¿Son consistentes las definiciones de variables globales entre los módulos?	5	5	4	4	5	4
6. ¿Se ha hecho el formateado principalmente para iluminar la estructura lógica del código?	5	5	3	4	2	4
7. ¿Son los nombres de los tipos de datos lo suficientemente descriptivos para ayudar a documentar las declaraciones de los datos? ¿Se usan específicamente para ese propósito?	4	3	1	4	4	5
8. ¿Se han nombrado bien las variables?	4	4	4	4	4	5
9. ¿Son las estructuras de datos simples de tal forma que minimizan la complejidad?	5	4	4	4	4	5
10. ¿Es complicado el acceso a los datos a través de rutinas de acceso abstractas (tipos abstractos de datos)?	5	5	5	4	3	4
11. ¿Son las estructuras de control simples de tal modo que minimizan la complejidad?	5	4	4	4	4	4

12. ¿Se ha minimizado el número de anidamientos?	5	5	3	4	4	4
13. ¿Puede alguien coger el código y comenzar a entenderlo inmediatamente?	5	3	3	4	3	4
14. ¿Son los comentarios claros y correctos?	2	3	3	3	2	3
15. ¿Se ha comentado cada variable global en el lugar donde se ha declarado?	2	2	2	3	2	3
16. ¿Se ha documentado cada variable global cada vez que se usa, bien con una convención de nombrado o con un comentario?	2	2	1	2	2	3
17. ¿Se ha comentado cada sentencia de control?	2	2	2	3	2	3
18. ¿Se han comentado los finales de estructuras de control largas o complejas?	2	2	2	2	2	3

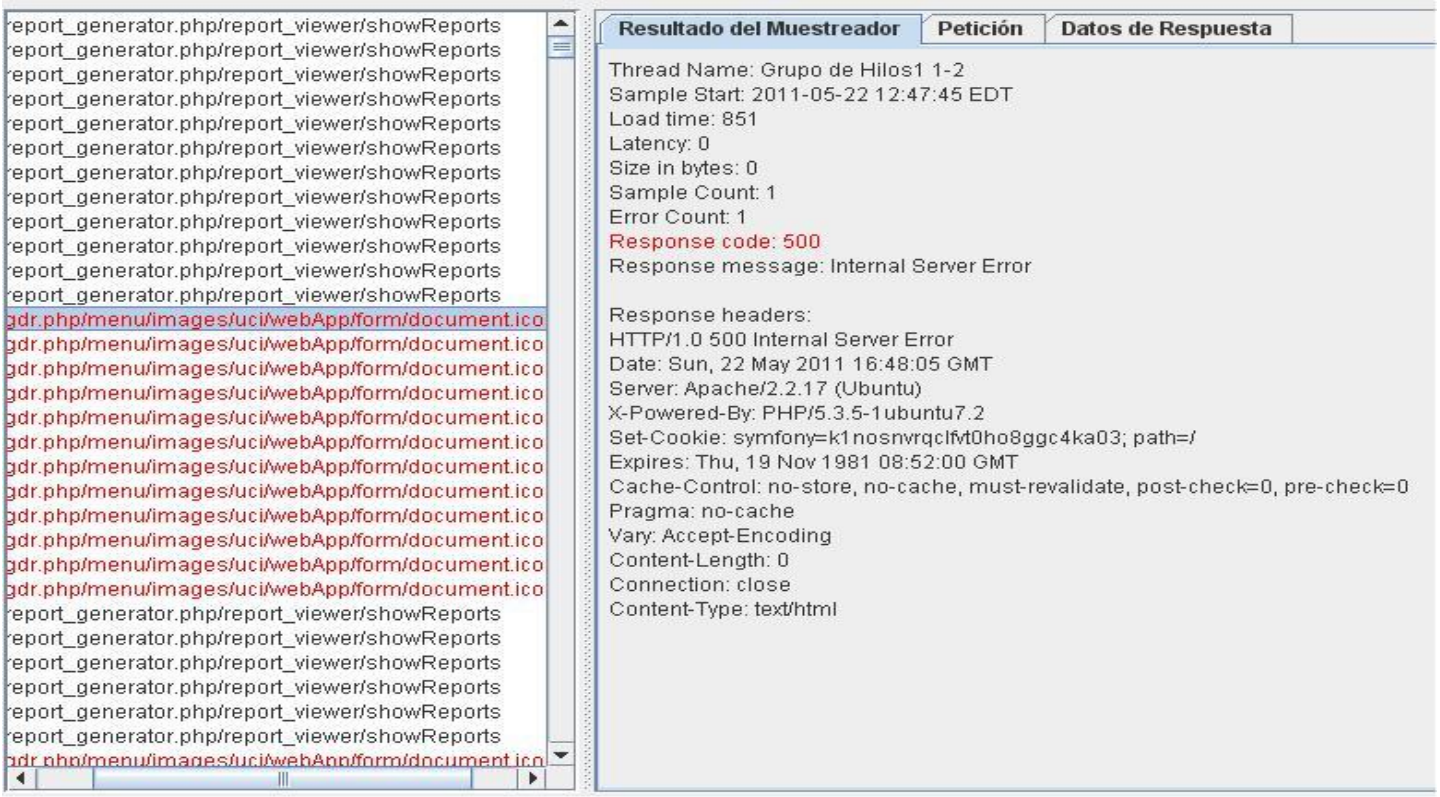
Anexo 3:

Árbol de resultado para Administrador de reporte


/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/showReport
/report_generator.php/report_manager/getUsers
/report_generator.php/report_manager/getUsers
/report_generator.php/report_manager/getUsers
/report_generator.php/report_manager/getUsers
/report_generator.php/report_manager/getUsers
/report_generator.php/report_manager/showGeneral
/report_generator.php/report_manager/showGeneral
/report_generator.php/report_manager/showGeneral
/report_generator.php/report_manager/showGeneral
/report_generator.php/report_manager/showGeneral
/report_generator.php/report_manager/getReportUsers
/report_generator.php/report_manager/getReportUsers
/resources/images/default/grid/grid3-hrow.gif
/resources/images/default/grid/grid3-hrow.gif
/report_generator.php/report_manager/getReportUsers
/report_generator.php/report_manager/getReportUsers
/resources/images/default/grid/grid3-hrow.gif
/resources/images/default/grid/grid3-hrow.gif
/report_generator.php/report_manager/getReportUsers
/resources/images/default/grid/grid3-hrow.gif
/report_generator.php/report_manager/showCategory
/report_generator.php/report_manager/showCategory
/report_generator.php/report_manager/showCategory
/report_generator.php/report_manager/showCategory
Resultado del Muestreador
Petición
Datos de Respuesta
Thread Name: Grupo de Hilos de admin para 100 user 1-1
Sample Start: 2011-05-22 14:25:23 EDT
Load time: 0
Latency: 0
Size in bytes: 320
Sample Count: 1
Error Count: 1
Response code: 404
Response message: Not Found
Response headers:
HTTP/1.1 404 Not Found
Date: Sun, 22 May 2011 18:25:47 GMT
Server: Apache/2.2.17 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 320
Keep-Alive: timeout=15, max=94
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1


gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_designer/getTemplates
gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_designer/getTemplates
gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_manager/showReport
gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_designer/getTemplates
gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_manager/showReport
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_manager/showReport
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_designer/getTemplates
gdr.php/menu/images/uci/webApp/form/document.ico
gdr.php/menu/images/uci/webApp/form/document.ico
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_manager/getUsers
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_designer/getTemplates
eport_generator.php/report_designer/getTemplates
Resultado del Muestreador
Petición
Datos de Respuesta
Thread Name: Grupo de Hilos de admin para 100 user 1-4
Sample Start: 2011-05-22 14:59:23 EDT
Load time: 125
Latency: 0
Size in bytes: 0
Sample Count: 1
Error Count: 1
Response code: 500
Response message: Internal Server Error
Response headers:
HTTP/1.0 500 Internal Server Error
Date: Sun, 22 May 2011 18:59:47 GMT
Server: Apache/2.2.17 (Ubuntu)
X-Powered-By: PHP/5.3.5-1ubuntu7.2
Set-Cookie: symfony=rchp5mi8u7lbmk2f1ko75ne780; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html

Árbol de resultado para Visor de reporte



GLOSARIO DE TÉRMINOS

- **API**: Application Programming Interfaz (interfaz de programación de aplicaciones). conjunto de funciones residentes en bibliotecas, que permiten que una aplicación corra bajo un determinado sistema operativo.
- **Casos de uso (CU):** Conjunto de secuencia de acciones que un sistema ejecuta y que produce un resultado observable para un actor.
- **Cliente Servidor:** Es originalmente aplicado a la arquitectura de software que describe el procesamiento entre dos o más programas: una aplicación y un servicio soportante.
- **GHz:** gigahercio. Unidad de medida informática.
- **Gigabyte (GB):** Unidad de medida informática.
- **Complejidad Ciclomática:** Es el número de caminos ejecutables en el código fuente.
- **CPU:** *Central Processing Unit* (unidad de proceso central). La CPU es el cerebro del ordenador.
- **CRM:** *Customer relationshíp Management* (administración de las relaciones con el cliente) Es una estrategia de negocios que consiste en focalizar los recursos de las empresas basados en un conocimiento real de todas interacciones de la compañía con el cliente y las respuestas de este a cada estimulo.
- **ERP:** *Enterprise resource planning* (Planificación de recursos empresariales) Conjunto de aplicaciones software que agrupan la empresa (finanzas, fabricación, ventas, RRHH) en una base de datos coherente.
- **GNU/Linux:** Términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux, que es usado con herramientas de sistema GNU.
- **HTTP:** Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto), es el protocolo usado en cada transacción de la Web (WWW).
- **Módulo:** Es una parte de un programa de ordenador. De las varias tareas que debe realizar un programa para cumplir con su función u objetivos, un módulo realizará una de dichas tareas (o quizá varias en algún caso).
- **.NET:** (network, internet) Es un dominio de Internet genérico que forma parte del sistema de dominios de Internet.
- **PDF:** Portable Document Format, (formato de documento portátil). Es un formato de almacenamiento de documentos.
- **Requisitos funcionales:** Son las capacidades o condiciones que el sistema tiene que cumplir para su correcto funcionamiento.
- **Requisitos no funcionales:** Definen propiedades y restricciones del sistema,

- **RAM:** Random Access Memory (memoria de acceso aleatorio) es la memoria desde donde el procesador recibe las instrucciones y guarda los resultados. Es el área de trabajo para la mayor parte del software de un computador.
- **SGBD:** Sistema Gestor de Bases de Datos. Es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones.
- **SQL:** *Structured Query Language* (lenguaje de consulta estructurado).
- **Software:** Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.
- **Servidor web:** Almacena documentos HTML, imágenes, archivos de texto, escrituras, y demás material Web compuesto por datos (conocidos colectivamente como contenido), y distribuye este contenido a clientes que la piden en la red.
- **Servidores:** Es una computadora que, formando parte de una red, provee servicios a otros denominados clientes.
- **TCP/IP:** Conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre redes de computadoras. Significa "Protocolo de control de transmisión/Protocolo de Internet"
- **Usuarios:** Persona que utiliza normalmente el software.
- **URL:** *Uniform Resource Locator*, es decir, localizador uniforme de recurso y se refiere a la dirección única que identifica a una página web en Internet.

- **PHPUnit:** Framework para PHP que nos facilita la creación de juegos de tests y la ejecución de estos y analiza sus resultados. Proporciona un marco que hace que la escritura de pruebas fáciles, así como la funcionalidad para ejecutar fácilmente las pruebas y analizar sus resultados. PhpUnit nos facilita la creación de pequeños scripts que nos ayudan a testear nuestras aplicaciones y a analizar los resultados. Se integra con varias aplicaciones de test. Es independiente del sistema operativo y un marco de código abierto.
- **PMD:** Herramienta para inspección o análisis de código. Un proyecto de código abierto mantenido por un gran número de desarrolladores y que además funciona con muchos lenguajes de programación. PMD exporta un archivo XML con los resultados del análisis realizado, y nuestra aplicación convierte estos archivos XML en HTML, formato que puede interpretarse con cualquier navegador web.
- **Spike PHPCoverage:** Herramienta para medir e informar sobre la cobertura de código proporcionado por el conjunto de pruebas de una aplicación PHP. Puede registrar la información de cobertura de la línea para todos los scripts PHP en tiempo de ejecución. También proporciona un

mecanismo extensible con un estándar de informe HTML implementado fuera de la caja. El informe por defecto muestra la información resumida sobre la cobertura de código para una aplicación y también muestra la información detallada acerca de un archivo, incluyendo las líneas que fueron ejecutados en realidad y con qué frecuencia. Es posible especificar los directorios y archivos que deben ser incluidos y / o excluidos de la medición de la cobertura. Tiene como licencia Software Libre (Open Source) y es independiente del sistema operativo.

- **JUnit:** Se trata de un conjunto de bibliotecas de la familia xUnit desarrolladas por Erich Gamma y Kent Beck con el fin de dar soporte a la realización de pruebas unitarias en las aplicaciones Java.