Universidad de las Ciencias Informáticas
Facultad 6.

Título: “Pruebas de liberación, su aplicación al sistema informático: Silenciamiento de Genes (siRNA)”.

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

Autores: Yanet Pérez Castellón.
Rafael Leyva Hernández.

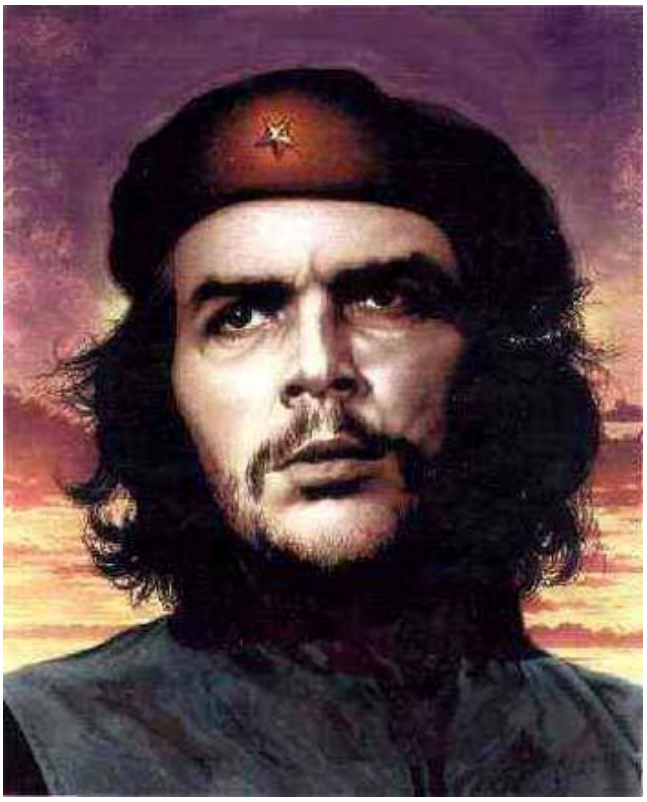
Tutores: Ing. Haylin Corujo Numa.
Lic. Maybel Anido Bada.

Ciudad de la Habana, Cuba,
Junio del 2009

“... LA LUCHA POR LA CALIDAD DEL PRODUCTO ES UNA LUCHA REVOLUCIONARIA Y DE VANGUARDIA...”

Ernesto Guevara de la Serna

3 de enero de 1962



DECLARACIÓN DE AUTORÍA.

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de __junio__ del año ___2009____.

Yanet Pérez Castellón. Rafael Leyva Hernández.

_____ _____

Firma del Autor. Firma del Autor.

Ing. Haylin Corujo Numa. Lic. Maybel Anido Bada.

_____ _____

Firma del Tutor. Firma del Tutor.

DATOS DE CONTACTOS

Tutora: Ingeniera Haylin Corujo Numa Correo electrónico: hcorujo@uci.cu

Tutora: Licenciada Maybel Anido Bada Correo electrónico: anido@uci.cu

AGRADECIMIENTOS.

A nuestros padres y familiares por habernos apoyado constantemente y por haber confiado con tanta plenitud en nosotros.

A nuestras tutoras Haylin Corujo y Maybel Anido por su incondicional ayuda y su preocupación por llevar a adelante este trabajo.

A nuestros profesores que nos han acompañado durante estos 5 años dándonos su apoyo.

Un agradecimiento especial para los profesores Yosdenis Urrutia y Abdel García por habernos apoyado tanto en todo momento.

A todos los que de una forma u otra contribuyeron a nuestra formación profesional.

DEDICATORIA.

A mis abuelos del alma Aída y José que con mucho trabajo, esfuerzo y cariño han hecho posible este sueño, ya que ellos son la razón por la cual me he esforzado día a día por ser mejor.

A mi esposo Dasvier al cual amo con todas las fuerzas de mi corazón, él siempre ha estado a mi lado para guiarme y ayudarme a ser cada día mejor. A él y mis abuelos les debo cada día de mis 5 años en esta escuela.

A mis hermanos Lázaro, Jean Carlos y Guillermo que son la cosita más linda de mi vida y con su cariño han logrado que en los momentos difíciles salga adelante.

A mis padres Yamilet y Raúl, que fueron los máximos responsables que hoy yo exista, y me han apoyado durante toda mi vida.

A mis abuelos paternos Mirella y Evaristo a los cuales quiero mucho, y me han ayudado a ser quien soy.

A mis amigos del alma Esneil, Nino, Maikel, Toledo, Kuki y El Rega los cuales me han ayudado cada día y han sido mi familia.

A mi amigo Alberto Garnache, que me ha servido siempre de guía y me dado esperanza y fuerzas para llevar la carrera adelante.

A mis hermanas Aleida Perera y Bárbara Mairene Carmona, las cuales han sido para mí más que simples amigas de universidad. Ellas siempre me han ayudado en todos los momentos buenos y malos.

A mis tres amigas Onaysi, Lorena y Rodaisy, por estar todo este curso soportándome, además me han apoyado en todo lo que he necesitado.

A mis antiguos compañeros de aula del grupo 6, los cuales fueron mi sostén cuando entre a esta escuela.

Yanet Pérez Castellón.

Dedicatoria.

A mi abuelo, le dedico mi tesis especialmente a mi abuelo que aunque no pudo verme hoy aquí yo sé que es lo que más hubiese querido por eso la primera persona en mi dedicatoria es él.

A mi abuela y a mis padres que sin ellos se me hubiera hecho imposible llegar hasta aquí hoy gracias por hacer lo posible y lo imposible porque este día llegara.

En general se la decido a toda mi familia, a mis amigos que han estado todos estos años al lado mío y que me han ayudado en buenos y malos momentos a Dismey, a Daine, a Saidel, al Chino, a Mayito, al Leo, a Karina, a Yania por aguantarme, a mis amigos del aula con quienes he pasado estos 5 años y me han ayudado en lo que han podido, a los pelú en fin a todos.

Rafael Leyva Hernández.

RESUMEN.

La calidad de software actualmente es un tema de interés a nivel mundial, esta adquiere cada día mayor importancia en el mercado debido a la exigencia de los clientes y la competencia existente entre las empresas.

El que un producto tenga calidad, implica que debe cumplir con ciertas exigencias entre las que están las requeridas por los clientes y la carencia de errores en el producto final. Las pruebas de software constituyen un elemento crítico para definir la calidad, además de ser una de las fases más importantes en el desarrollo de un sistema.

Entre los proyectos desarrollados por estudiantes y profesores de la Universidad de las Ciencias Informáticas (UCI) se halla el proyecto Silenciamiento de genes (siRNA) el cual es solicitado por: El Centro de Ingeniería Genética y Biotecnología (CIGB).

Dada la necesidad de que este proyecto tenga la calidad requerida para el cliente se le aplicó un grupo de pruebas de software fundamentando los objetivos, así como las técnicas para el desarrollo de las mismas. Además se realizó una planificación de estas pruebas teniendo en cuenta el nivel al cual pertenece dicha prueba, los métodos de pruebas, técnicas y herramientas utilizadas para llevar a cabo las mismas.

PALABRAS CLAVE.

Aplicación, Calidad, Informática, Pruebas, Software.

TABLA DE CONTENIDOS.

INTRODUCCIÓN.

En los últimos años se ha incrementado considerablemente la producción de software a nivel mundial ya que a través de su desarrollo ha surgido una nueva etapa en el mundo "La Era de la Informática y las Comunicaciones". Cada vez se hace más necesario el uso de software para el desarrollo de las empresas, pues estos brindan la automatización necesaria para facilitar el trabajo a todo el personal.

En Cuba la producción de software se torna bastante modesta con respecto al mundo, y una de las causas fundamentales es que el país no es muy conocido a nivel mundial como productor de software. Esto no quiere decir que no luche por penetrar en el mercado internacional, ya que esto sería una fuente de ingreso para el país. Dentro de su modesta producción ha creado disímiles software vinculados a tres áreas fundamentales del país como es la salud, la educación y el deporte. Es necesario citar que para que Cuba pueda tener mayor auge en el mercado debe tener presente la calidad del software lo que implica que los productos realizados deben ser confiables, precisos, rápidos de utilizar, flexibles y además deben estar bien documentados, lo cual influye en la competencia y en la aceptación de los mismos por parte de los clientes.

Para garantizar la calidad de los productos se hace necesario que en el proceso de desarrollo de software se tenga en cuenta un plan de aseguramiento de la calidad a lo largo del ciclo de vida del software, ya que es inevitable que las personas involucradas, no cometan errores por lo cual se hace indispensable el control de todas sus funcionalidades.

Es por ello que si se le aplican las pruebas requeridas a los productos en su etapa de desarrollo se logra que estos salgan al mercado con un mínimo de errores y tengan una buena aceptación por parte de los clientes.

En el marco de la naciente industria cubana del software la UCI juega un rol fundamental y ha establecido convenios de colaboración con los centros científicos del polo del oeste. Dentro de estos convenios de colaboración se desarrolla el proyecto Silenciamiento de Genes mediante RNA pequeños de interferencia (siRNA) el cual se plantea como una poderosa herramienta terapéutica en el futuro.

De este se conoce que: "Hace aproximadamente nueve años, se descubrió la existencia de genes que permitirían el silenciamiento de otros genes; es decir, que serían capaces de frenar su acción en el organismo. Ahora se sabe que estos genes están contenidos en casi todos los organismos, por lo que puede utilizarse como una forma de terapia génica: si cierta enfermedad es el efecto del incremento en

la expresión de un gen específico y se conoce su identidad, se podría “apagar” su expresión por medio de inyecciones de RNA pequeños de doble cadena y curar así la enfermedad”. [1]

“Existe un gran número de científicos y especialistas que realizan estudios sobre el diseño de siRNA y su aplicación en el silenciamiento de genes sobre la base de herramientas online publicadas en Internet. Uno de los problemas de la utilización de tales herramientas es que no se conocen los algoritmos que se han utilizado para la predicción de siRNA y no son configurables para el uso de algoritmos propios. No dan la posibilidad de hacer un análisis con el por ciento de exactitud que se estime conveniente de acuerdo al estudio que se esté realizando y no existe una unidad en las ideas expuestas en estos sitios, razón que provoca que haya diversidad sobre el mismo tema”. [1]

El CIGB conjuntamente con la UCI han creado una aplicación capaz de dar solución a algunos de estos problemas, esta aplicación informática se encuentra actualmente en su fase final por lo que se hace necesario realizar el procedimiento de liberación para garantizar que el producto tenga calidad, para esto debe contar con una perfecta documentación y con un buen funcionamiento de la aplicación pues de eso depende la aceptación del producto por parte de los clientes.

Por lo tanto se identifica como **problema científico de la investigación** ¿Cómo verificar la calidad en la aplicación informática siRNA?

El **objeto de estudio** es: El proceso de pruebas de Calidad en el desarrollo del Software.

El **Campo de acción** se enmarca en: Pruebas de liberación a la aplicación informática siRNA en el laboratorio de Calidad de la Facultad 6

El **objetivo general de la investigación** es Desarrollar el proceso de pruebas de liberación a la aplicación informática siRNA.

Para dar cumplimiento al objetivo general se tuvieron en cuenta los siguientes **objetivos específicos:**

- Estudiar el proceso que se utiliza para llevar a cabo el desarrollo de las pruebas.
- Diseñar los casos de pruebas que serán aplicados al Software.
- Realizar pruebas a la documentación de dicho Software.
- Realizar pruebas de Caja negra a la aplicación.
- Realizar pruebas de Carga y de Stress.

- ❖ Documentar los resultados obtenidos.

Para dar cumplimiento a los objetivos se trazaron las siguientes **tareas**:

1. Estudio Bibliográfico sobre los siguientes temas:

    - Calidad del Software.
    - Pruebas de calidad que más se aplican, objetivos y pasos.
    - Confección de la documentación del proceso de pruebas.
    - Pruebas de Carga y de Stress.

2. Elaboración del plan de pruebas.

3. Revisión de los documentos pertenecientes al proyecto:

    - Manual de usuario.
    - Modelo de casos de usos.
    - Diccionario de datos.
    - Especificación de Requisitos.

4. Elaboración de los diseños de casos de pruebas.

5. Realización de pruebas caja negra.

6. Realización de pruebas de carga y de stress.

7. Elaboración del informe de no conformidades.

El documento está estructurado de la siguiente manera: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, anexos y glosario de términos.

En el Capítulo 1: Fundamentación Teórica. Se reflejan temas de interés sobre los procesos de desarrollo de software y unido a esto la calidad de los mismos. Se hace un análisis de las técnicas de pruebas de software, objetivos de las pruebas y estrategias para su aplicación así como los niveles de pruebas y los métodos conocidos para la ejecución de las mismas. Además se hace referencia a los

artefactos y el rol existente en el flujo de trabajo de pruebas así como al estado actual de las pruebas en nuestro país. Se define la herramienta con la cual se le realizarán las pruebas a la aplicación.

En el Capítulo 2: Diseño y aplicación de las pruebas al Software. Se dará una descripción del proyecto que se liberará, además se presentará el plan de pruebas general y los diseños de pruebas para la aplicación informática así como se describirá la estrategia de las pruebas realizadas, las técnicas, los métodos y las herramientas con las cuales se realizarán las pruebas de carga y de stress a dicha aplicación.

En el Capítulo 3: Evaluación de los resultados. Se mostrarán los resultados obtenidos después de haber realizado las pruebas a la aplicación informática, así como una evaluación de los mismos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

Introducción.

En este capítulo se refleja el estado del arte de este tema. Teniendo en cuenta los objetivos trazados se analizan algunos temas como: la calidad de un software, los objetivos de pruebas y estrategias a seguir para su aplicación, las técnicas de pruebas que existen, como se llevan a cabo las mismas, los niveles a los que pertenecen estas pruebas, herramientas a utilizar, además se describe el rol del probador, se refleja la importancia de estas pruebas y se dan a conocer las pruebas especificas para esta aplicación.

Es necesario decir que el término calidad ha ido evolucionando a lo largo de la historia adaptándose a diferentes interpretaciones y manifestando en todo momento su carácter subjetivo. Partiendo de los conceptos clásicos de calidad en los procesos productivos del siglo XX, el nacimiento del concepto actual de calidad se puede situar en la adopción de la “Calidad Total” propuesta por Joseph M. Juran y W. Edwards Deming. La calidad daba así un gran salto, pasando de la mera inspección a la mejora de los procesos de una organización incluyendo a las personas involucradas en dichos procesos.

Actualmente, se ha superado el planteamiento inicial que relegaba a la calidad del software como una mera actividad, pasando a ser una política de gestión empresarial, donde el compromiso de todos los niveles organizativos es un requisito para el éxito. “El objetivo ya no es la ausencia de defectos sino la adecuación a los requisitos de negocio y del cliente”. [2]

Las pruebas en el software son un punto importante para garantizar la calidad y estas representan una revisión detallada de las especificaciones, del diseño y de la codificación. Las pruebas forman parte del desarrollo de cualquier software y son un tema significativo dentro de la ingeniería de los mismos.

1.1 Calidad de Software.

La calidad está presente en todas las actividades del ciclo de vida de un proyecto de software. Para ello, se hace imprescindible disponer de una serie de pilares sobre los que se sustentan las actividades de calidad como por ejemplo: la infraestructura de soporte, un grupo de personas especializada en esta disciplina y procesos alineados con los objetivos de negocio que permitan una mayor industrialización en el desarrollo y mantenimiento del software.

La calidad está orientada a las facilidades que ofrece el software una vez que este esté terminado, él mismo debe poseer las siguientes características para su buen funcionamiento:

Fiabilidad: Capacidad de operar sin errores.

Modificable: Capacidad de hacer los cambios necesarios de una forma sencilla.

Comprensible: Capacidad de comprender el software operativo, de cara a un cambio o arreglo.

Rendimiento: Velocidad y compacidad del software.

Utilizable: Capacidad de uso sencillo del software.

Probable: Capacidad de construir y ejecutar fácilmente los diseños de casos de prueba.

Portable: Capacidad de mover el software fácilmente de un entorno de trabajo a otro.

La calidad debe ser especificada, planificada, administrada, medida y certificada. Esto implica una visión integral que arroja la comprobación del software, con el fin de lograr un mayor grado de satisfacción y confianza del cliente hacia la organización productora de software. “Constituye entonces las pruebas del software, tarea de alta prioridad para las empresas productoras”. [3]

1.2 Aseguramiento de Calidad del Software.

El aseguramiento de la calidad, se puede definir como el esfuerzo total para plantear, organizar, dirigir y controlar la calidad en un sistema de producción con el objetivo de dar al cliente productos con la calidad adecuada. Este es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto va a satisfacer los requisitos dados de calidad. Este se diseña para cada aplicación antes de comenzar a desarrollarla y no después de haberla terminado, muchos autores presumen que más que aseguramiento la calidad es garantía pero esto puede traer consigo confusión debido a que:

– Garantía, puede ser confundida con garantía de productos.

– Aseguramiento pretende dar confianza en que el producto tiene calidad.

El aseguramiento de calidad del software está presente en:

– Métodos y herramientas de análisis, diseño, programación y prueba.

– Inspecciones técnicas formales en todos los pasos del proceso de desarrollo del software.

– Estrategias de prueba multiescala.

– Control de la documentación del software y de los cambios realizados.

– Procedimientos para ajustarse a los estándares.

– Mecanismos de medida.

– Registro de auditorías y realización de informes.

1.3 Pruebas de Calidad de Software.

Las pruebas son una actividad primordial en el proceso de desarrollo del software, estas permiten al desarrollador determinar si el producto generado satisface las expectativas establecidas por el cliente. Estas detectan errores que pudieran generar comportamientos inapropiados durante la ejecución del producto.

Un concepto más específico dado por algunos desarrolladores de software es que las pruebas son: "Cualquier intento de demostrar que el software tiene propiedades por debajo de la calidad requerida."[4] Diciéndolo de esta forma, las pruebas permiten encontrar aquellas imperfecciones que pueda tener el software, las cuales "no" garantizan que este pueda ser óptimo.

La prueba de software es una actividad de evaluación del producto, esta es realizada para saber en qué condiciones se encuentra el software, basándose en este criterio se puede tomar en consideración el concepto emitido por Pressman en su edición de 1998 donde destaca que: "La prueba del software es un elemento crítico para la garantía de calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación". [5]

Las pruebas de un sistema se definen como el proceso de ejercitar o evaluar el sistema por medios manuales o automáticos, para verificar que satisface los requerimientos o, para identificar diferencias entre los resultados esperados y los que produce el sistema.

A partir de los criterios anteriores se puede definir que la prueba de software es una actividad en la cual el sistema es ejecutado bajo condiciones específicas para demostrar que no tiene la madurez necesaria para ser implantado.

Estas deficiencias que permiten que no tenga la madurez necesaria pueden ser problemas en el código, en el modelado o en la documentación, en dependencia del tipo de pruebas que se le apliquen al software.

1.3.1 Objetivos de las pruebas de software.

Dentro de los principales objetivos que se llevan a cabo para realizarle las pruebas al software se encuentran:

- Mostrar la mayor cantidad posible de fallas o errores.
- Ofrecer un mayor nivel de seguridad en los productos que se van generando.
- Elevar la calidad del producto final.
- Lograr diseños de casos de pruebas con alta probabilidad de mostrar un error no descubierto hasta entonces.
- Ejecutar las pruebas, Registrar resultados y Analizar resultados.

El objetivo de toda prueba según Pressman en su edición de 1998 es:" que saquen a la luz diferentes clases de errores con la menor cantidad de tiempo y espacio". [5]

Estos objetivos muestran la necesidad que existe de realizar las pruebas, pues si estas se llevan a cabo con éxito se descubrirán errores en el software, dándole a este mayor integridad.

Es importante citar que "las pruebas de software no pueden asegurar la ausencia de defectos, solo pueden demostrar que existen defectos en el software". [6]

1.3.2 Estrategia de pruebas de software.

"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"[4]. Una estrategia de prueba del software proporciona un plano a seguir para el responsable del desarrollo del software, a la organización de control de calidad y al cliente. Por tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de las pruebas y la agrupación y evaluación de los datos resultantes.

La primera prueba a realizar es la prueba de unidad la cual se lleva a cabo en cada módulo del software, específicamente en el código de la aplicación. La prueba avanza hasta llegar a la prueba de integración, donde el centro de atención es el diseño y la construcción de la arquitectura del software. Luego se realizan las pruebas al sistema, en la que se prueba como un todo el software, y otros elementos del sistema. Finalmente se realiza la prueba de aceptación donde se comprueba que el sistema está listo para desplegar.

Tom Gilb plantea que se deben abordar los siguientes puntos si se desea implementar con éxito una estrategia de prueba del software:

- Especificar los requisitos del producto de manera cuantificable mucho antes de que comiencen las pruebas.
- Establecer los objetivos de la prueba de manera explícita.
- Comprender qué usuarios va a tener el software y desarrollar un perfil para cada categoría de usuario.
- Desarrollar un plan de prueba que haga hincapié en la prueba de ciclo rápido.
- Construir un software «robusto» diseñado para probarse a sí mismo.
- Usar revisiones técnicas formales efectivas como filtro antes de la prueba.
- Llevar a cabo revisiones técnicas formales para evaluar la estrategia de prueba y los propios diseños de casos de prueba.

- Desarrollar un enfoque de mejora continua al proceso de prueba. Debería medirse la estrategia de prueba.

De esta forma queda construida una buena estrategia de pruebas para llevar a cabo.

1.3.3 Niveles de prueba de software.

La Prueba es aplicada para varios objetivos, estas son usadas en diferentes escenarios o niveles de trabajo.

- Prueba de Desarrollador.
- Prueba Independiente.
- Prueba de Unidad.
- Prueba de Integración.
- Prueba de Sistema.
- Prueba de Aceptación.

Prueba de Desarrollador: Esta es diseñada e implementada por el equipo de desarrollo. Estas pruebas han sido consideradas solo para la prueba de unidad, aunque en algunos casos pueden ejecutar pruebas de integración. Es recomendable que estas pruebas cubran más que las pruebas de unidad.

Prueba Independiente: Es diseñada e implementada por alguna persona independientemente del grupo de desarrolladores. Su objetivo es proporcionar una perspectiva diferente y un ambiente más rico que los desarrolladores. Una vista de la prueba independiente es la prueba independiente de los stakeholder, que son pruebas basadas en las necesidades y preocupaciones de los stakeholders.

Prueba de Unidad: Es la prueba orientada a los módulos de código o subsistemas del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a Caja Blanca.

Antes de realizar otra prueba es necesario probar el flujo de datos de la interfaz del componente.

Si estos no reaccionan correctamente, todas las demás pruebas no tienen sentido.

El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente.

Normalmente es realizada por el programador y no por el probador, ya que requiere un conocimiento detallado del interior y del diseño de programa y del código. No es siempre fácil realizarse a menos que la aplicación disponga de un buen diseño de arquitectura con código muy estricto.

Prueba de Integración: Es ejecutada para asegurar que los componentes en el modelo de implementación maniobren correctamente cuando son mezclados para ejecutar un caso de uso. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Estas pruebas arrojan errores o incompletitud en las especificaciones de las interfaces de los paquetes.

Esta prueba debe ser responsabilidad de desarrolladores y de independientes. Es el proceso de combinar y probar diversos componentes juntos. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Prueba de Sistema: Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados.

En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos bien formados de comportamiento de caso de uso son implementados.

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

En un proceso iterativo deben ser ejecutadas todas las pruebas de unidad antes de pasar a los próximos niveles. “Una mejor estrategia es identificar las pruebas de unidad, integración y sistema que ofrecen mayor potencial para encontrar errores y entonces implementarlas y ejecutarlas”. [7]

A partir de este estudio se ha llegado a la conclusión de que las pruebas realizadas a la aplicación informática siRNA serán desarrolladas en el nivel de prueba de sistema, donde se probará la aplicación como un todo, pues esta aplicación cuenta con un solo módulo. Además no se pueden

desarrollar las pruebas de unidad pues estas son realizadas por el desarrollador del sistema, no por el probador.

1.3.4 Técnicas de prueba de software.

Las técnicas de pruebas de forma general ayudan a asegurar las buenas funcionalidades del software, su buen uso, la fiabilidad de los mismos, el rendimiento que este tiene y el soporte que lo mantiene. Existen técnicas de pruebas para verificar los requerimientos funcionales y no funcionales que debe tener la aplicación. Cada técnica de prueba posee un objetivo específico y una forma de usarse. Entre estas técnicas de pruebas se encuentran:

Función: Estas técnicas centran su atención en la validación de las funciones, métodos, servicios y caso de uso.

Seguridad: Asegura que los datos y el sistema solo pueden ser accedido por los actores deseados.

Volumen: Se enfoca en verificar las habilidades de los programas para manejar amplias cantidades de datos, tanto de entrada, salida o residente en la Base de Datos.

Usabilidad: Técnica enfocada a factores humanos, estéticos, consistencia en la interfaz de usuario, ayuda sensitiva al contexto y en línea, asistente documentación de usuarios y materiales de entrenamiento.

Integridad: Orientada a ser fuerte, resistente a fallos.

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

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

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

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

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

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

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

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

Dado el estudio se decidió utilizar las técnicas funcionales, de carga y de stress, pues la técnica funcional verificará que los casos de uso se correspondan con las funcionalidades del sistema así como las técnicas de carga y de stress serán las encargadas de verificar el rendimiento del mismo.

1.3.5 Métodos de prueba de software

Existen varios métodos para ejecutar las pruebas de software, entre los más elementales se encuentran la prueba de Caja Negra, prueba de Caja Blanca.

La prueba de Caja Blanca es la mejor de su tipo para verificar que se recorran todos los caminos y detectar un mayor número de errores. La Caja Negra brinda la posibilidad de cubrir la mayor parte de las combinaciones de entradas y lograr con ello un juego de pruebas más eficaz.

Las pruebas de Caja Negra se llevan a cabo sobre la interfaz del software. El objetivo fundamental es demostrar que las funciones del software son operativas, también las entradas de datos deben ser

aceptadas de forma adecuada y así se produce un resultado correcto, además debe mantenerse la integridad de la información externa.

Estas pruebas se centran principalmente en los requisitos funcionales del software. Permitiendo encontrar funciones incorrectas o ausentes que no se hayan notado, así como errores de interfaz, errores en las estructuras de datos o en accesos a las Bases de Datos externas, errores de rendimiento y errores de inicialización y terminación.

En las pruebas de Caja Blanca se comprueban los caminos lógicos del software. Se examina el estado del programa en varios puntos para determinar si el estado real coincide con el esperado. Estas pruebas son realizadas al código de dicha aplicación.

Estas pruebas requieren del conocimiento de la estructura interna del programa. Deben garantizar como mínimo que se ejerciten por lo menos una vez todos los caminos independientes para cada módulo en caso de que exista más de uno además se ejercitan todas las decisiones lógicas en sus vertientes verdaderas y falsa, se ejecutan todos los bucles en sus límites y con sus límites operacionales y se ejercitan también las estructuras internas de datos para asegurar su validez.

Las pruebas mencionadas permiten probar cada una de las condiciones existentes en el programa, identificar claramente las entradas, salidas y estudiar las relaciones que existen entre ellas, permitiendo así maximizar la calidad de las pruebas y en dependencia del resultado se contará con un sistema más estable y confiable.

1.3.5.1 Prueba de Caja Blanca.

Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se observa así el código, el cual debe ser probado completamente sin tener en cuenta los aspectos de rendimiento del software.

En este caso puede ser impracticable realizar una prueba exhaustiva de todos los caminos de un programa. Por esto se han definido diversos criterios de cobertura lógica, los cuales permiten decidir qué sentencias o caminos se deben examinar con los diseños de casos de prueba. Uno de estos criterios es:

Cobertura de Caminos: En este se escriben diseños de casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Entendiendo camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

Este criterio será el ampliado pues es uno de los más usados, teniendo en cuenta la efectividad del mismo.

La aplicación de este criterio de cobertura asegura que los diseños de casos de prueba diseñados permitan que todas las sentencias del programa sean ejecutadas al menos una vez y que las condiciones sean probadas tanto para su valor verdadero como falso.

Una de las técnicas empleadas para aplicar este criterio de cobertura es la Prueba del Camino Básico. Los pasos a realizar para aplicar esta técnica son:

- Representar el programa en un grafo de flujo.

- Calcular la complejidad ciclomática.

- Determinar el conjunto básico de caminos independientes.

- Derivar los diseños de casos de prueba que fuerzan la ejecución de cada camino.

Representar el programa en un grafo de flujo:

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

Nodos:  Aristas:  Regiones: 

Nodos predicados: Cuando en una condición aparecen uno o más operadores lógicos (AND, OR, XOR) se crea un nodo distinto por cada una de las condiciones simples. Cada nodo generado de esta forma se denomina nodo predicado. En esta figura se muestra un ejemplo de condición múltiple.

```
IF a OR b
    THEN
        x
    ELSE
        y
ENDIF
```

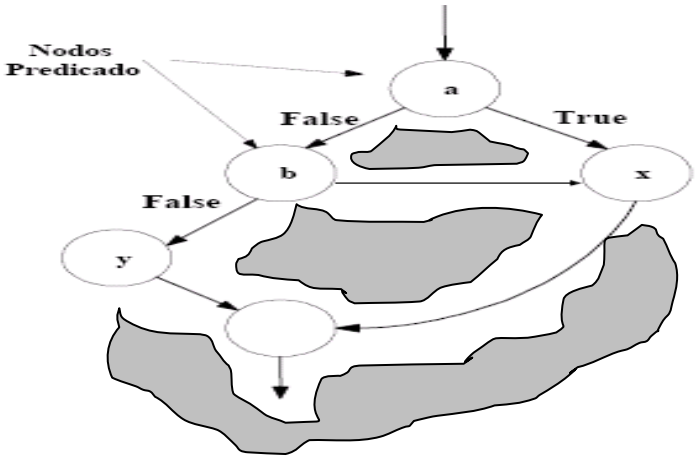


Fig.1.1 Representación gráfica de una sentencia de código.

Calcular la complejidad ciclomática:

Existen varias formas de calcular la complejidad ciclomática de un programa a partir de un grafo del flujo:

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

La complejidad ciclomática es la encargada de decir cuantos diseños de casos de pruebas deben ejecutarse para garantizar que todas las sentencias de código se hayan ejecutado.

Determinar el conjunto básico de caminos independientes

Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una condición, respecto a los caminos existentes. El conjunto de caminos independientes de un grafo no es único. No obstante, a continuación, se mostrará algunas heurísticas para identificar dichos caminos:

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

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

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

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

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

Derivar los diseños de casos de prueba que fuerzan la ejecución de cada camino.

Este último paso es construir los diseños de casos de pruebas que fuerzan la ejecución de cada camino, una forma de hacerlo es como se muestra en la tabla.

Número del Camino	Caso de Prueba	Resultado Esperado

Fig.1.2 Diseños de Casos de Pruebas por Caminos.

1.3.5.2 Pruebas de Caja Negra.

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.

Para confeccionar los diseños de casos de pruebas de Caja Negra existen distintos criterios. Algunos de estos son:

- Particiones de Equivalencia.
- Análisis de Valores Límite.
- Métodos Basados en Grafos.

Los cuales serán ampliados a continuación:

Partición equivalente:

"La partición equivalente es un método de prueba de Caja Negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar los casos de prueba" [6]. Un diseño de caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de diseños de casos de prueba que descubran clases de errores, reduciendo así el número total de diseños de casos de prueba que hay que desarrollar.

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 la partición equivalente es reducir el posible conjunto de diseños 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.

En el diseño de casos de prueba para partición equivalente se procede en dos pasos:

Se identifican las clases de equivalencia. Las clases de equivalencia son identificadas tomando cada condición de entrada y repartiéndola en dos o más grupos.

Es de notar que dos tipos de clases de equivalencia están identificados: las clases de equivalencia válidas representan entradas válidas al programa, y las clases de equivalencia inválidas que representan el resto de los estados posibles de la condición (es decir, valores erróneos de la entrada).

Se define los diseños de casos de prueba. El segundo paso es el uso de las clases de equivalencia para identificar los diseños de casos de prueba. El proceso es como sigue: se asigna un número único a cada clase de equivalencia. Hasta que todas las clases de equivalencia válidas han sido cubiertas por los diseños de casos de prueba, se escribe un nuevo diseño de caso de prueba que cubra la clase de equivalencia válida. Y por último hasta que los diseños de casos de prueba hayan cubierto todas las clases de equivalencia inválidas, se escribe un diseño de caso de la prueba que cubra una, y solamente una, de las clases de equivalencia inválidas descubiertas.

Análisis de valores límite:

Los errores tienden a darse más en los límites del campo de entrada que en el centro. Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba. El análisis de valores límite lleva a una elección de diseños de casos de prueba que ejerciten los valores límite.

El análisis de valores límite 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 diseños de casos de prueba en los extremos de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene diseños de casos de prueba también para el campo de salida.

Condiciones sublímite. 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. Una condición sublímite común es la tabla de caracteres ASCII, por ejemplo, si se está evaluando una caja de texto que acepta solamente los caracteres A-Z y a-z, se debe incluir los valores en la partición inválida justo «debajo de» y «encima de» esos caracteres de la tabla ASCII, [", y {.

Métodos de prueba basados en grafos:

En este método se debe entender los objetos que se modelan en el software y las relaciones que conectan a estos objetos. Una vez que se ha llevado a cabo esto, el siguiente paso es definir una serie de pruebas que verifiquen que todos los objetos tienen entre ellos las relaciones esperadas. 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.

Algunos modelados para pruebas de comportamiento que pueden hacer uso de los grafos son:

Modelado del flujo de transacción. Los nodos representan los pasos de alguna transacción, y los enlaces representan las conexiones lógicas entre los pasos.

Modelado de estado finito. Los nodos representan diferentes estados del software observables por él, y los enlaces representan las transiciones que ocurren para moverse de estado a estado.

Modelado de flujo de datos. Los nodos objetos de datos y los enlaces son las transformaciones que ocurren para convertir un objeto de datos en otro.

Gráfica Causa-efecto. La gráfica Causa-efecto representa una ayuda gráfica en seleccionar, de una manera sistemática, un gran conjunto de diseños de casos de prueba. Tiene un efecto secundario beneficioso en precisar estados incompletos y ambigüedades en la especificación.

Este método fue el escogido para desarrollar las pruebas a la aplicación pues verificará todas las funcionalidades del sistema, además se hará uso de este método utilizando el criterio de Partición Equivalente el cual permite minimizar la cantidad de diseños de casos de pruebas. Dicho método se basa en garantizar que la interfaz funcione adecuadamente.

1.3.6 Herramientas utilizadas en las Pruebas de software.

Existe una gran variedad de herramientas para automatizar de cierta forma el proceso de desarrollo de las pruebas de software. Entre estas herramientas de pruebas se puede encontrar para Carga y Stress, para la automatización de pruebas funcionales y para pruebas de código fuente, entre otras.

En este caso se utilizan las pruebas de carga y stress, y dentro de esta específicamente la herramienta JMeter.

Herramientas para pruebas de Carga y Stress.

OpenLoad Tester.

OpenLoad Tester es una herramienta de optimización de rendimiento basada en navegador para pruebas de carga y stress de sitios Web dinámicos. Esta permite elaborar escenarios de ejecución y ejecutarlos de forma repetida, simulando la carga de un entorno de producción real de la aplicación como si múltiples usuarios estuvieran usándola.

Benchmark Factory

Benchmark Factory es una herramienta de prueba de carga y capacity planning, capaz de simular el acceso de miles de usuarios a sus servidores de bases de datos, archivos, internet y correo, localizando los posibles cuellos de botella y aislando los problemas relacionados con sobrecargas del sistema.

JMeter

Es una herramienta de Apache que sirve para realizar pruebas de estrés a aplicaciones web. Con esta herramienta se puede atacar aquellos puntos que se quieran probar con respecto a concurrencia de usuarios y velocidad de carga en un sitio. Además esta herramienta es un Software Libre.

Para ello generalmente se elabora un plan de trabajo y se definen los parámetros de la o las peticiones que se atacarán en ese plan de prueba. Para ello deben configurarse las peticiones que se harán y también deberán agregarse las formas de recuperar la información para obtener los resultados de las pruebas. Estas formas de recuperar información pueden ser gráficos, tablas, etc.

JMeter no se limita solamente a pruebas de carga para aplicaciones web. Pues esta también se utiliza para realizar pruebas de estrés sobre drivers para base de datos, por lo que las pruebas deben enfocarse para medir el nivel de fiabilidad del driver con respecto a la base de datos.

Esta herramienta se utilizará, puesto que facilita de manera simple las pruebas de carga, concurrencia y stress sobre dicha aplicación web.

1.3.7 Metodología.

Es importante escoger una buena metodología si se pretende reducir costos y retrasos de proyectos así como mejorar la calidad del software, pues estas cobran gran importancia en proyectos empresariales ya que al no utilizarla adecuadamente se puede desembocar en la frustración del equipo de desarrollo y en la insatisfacción de los clientes. Por tanto el uso de una metodología es necesaria para controlar el ciclo de vida de un proyecto. El proyecto siRNA se desarrolló utilizando la metodología OpenUp la cual es un proceso de desarrollo iterativo del software que es mínimo, completo, y extensible. En este solamente es incluido el contenido fundamental y necesario. OpenUP como metodología de desarrollo es conducida por el principio de colaboración para alinear intereses y para compartir su comprensión.

1.3.7.1 Rol de Probador.

Este es el responsable durante las actividades principales de las pruebas, el cual incluye la conducción de las pruebas necesarias y el registro de resultado de la prueba. Ejecuta las pruebas diseñadas y anota los resultados obtenidos. Este rol es responsable de las actividades principales del esfuerzo de las pruebas. Estas actividades incluyen identificar, definir, implementar y dirigir las pruebas necesarias, como también verificar los resultados de las pruebas y analizar los resultados.

1.3.7.1.1 Tareas que desempeña este rol.

Este rol es responsable de las siguientes tareas:

- Identificar las pruebas que se requieren llevar a cabo.
- Identificar el acercamiento más apropiado para implementar una prueba dada.
- Implementar pruebas individuales.
- Preparar y ejecutar las pruebas.
- Estudiar resultados y verificar que las pruebas hayan sido ejecutadas.
- Analizar y recuperar los errores de ejecución.
- Comunicar los resultados de las pruebas al equipo.

1.3.7.1.2 Habilidades que debe tener este Rol.

Una persona para desempeñar este rol debe contar con las siguientes habilidades:

- Conocimiento de tendencias y técnicas de pruebas.
- Habilidades para el diagnóstico y solución de problemas.
- Conocimiento del sistema o aplicación que está siendo probada.
- Conocimiento de redes y arquitectura de sistemas.

Para automatizar estas pruebas, se deben tener estas cualidades adicionales:

- Entrenamiento en el uso apropiado de herramientas de automatización de pruebas.
- Experiencia usando herramientas de automatización de pruebas.
- Habilidades de programación.
- Habilidades en depuración y diagnóstico.

El requerimiento de habilidades específicas varía dependiendo del tipo de pruebas que se este dirigiendo. Por ejemplo, las habilidades necesarias para tener éxito en el uso de sistemas cargados con herramientas de automatización de pruebas son diferentes de aquellas necesarias para la automatización de pruebas de sistemas funcionales.

1.3.8 Importancia de las pruebas de software.

En la mayoría de los diseños de casos las pruebas de software son un escalón eliminado, esto se debe a que muchos equipos de desarrollo piensan que estas no son necesarias o en ocasiones se hace necesario dejarlas a un lado por problemas de atrasos en el producto. De esta forma por un motivo o por otro las pruebas son reducidas casi al máximo en el desarrollo del software.

Por su puesto, en la mayoría de los casos a todo el personal de desarrollo le conviene más vender el producto a tiempo en el mercado antes que los demás competidores y luego en pocos meses sacar las próximas versiones con los errores corregidos, pero esto tiene sus inconvenientes pues a medida que avanza el ciclo de desarrollo del software el trabajo de encontrar y corregir un simple error crece enormemente. Es cierto que en la mayoría de las ocasiones las pruebas son costosas y muy molestas pero estas son las únicas que certifican que el producto este “libre” de errores.

Dada esta afirmación es necesario decir que el software no tiene que ser 100% libre de errores, porque esto es casi imposible de hacerse, pero lo que si es cierto es que debe tener la menor cantidad posible ya que este se ha revisado y probado detalladamente.

La mejor idea es "probar en etapas tempranas y probar a menudo." Realmente es mucho más efectivo en costo desarrollar software robusto a desarrollarlos con una serie de errores, pero para esto se deben esforzar por mantener la calidad desde el primer día pues mientras más se pruebe el software, mayor cantidad de errores serán encontrados y eliminados.

Es importante que los productos de software sean usables y con la calidad necesaria pues sin esto se estaría defraudando la confianza de nuestros clientes los cuales siempre tienen que quedar complacidos.

La idea es realizar productos "fáciles de usar y con calidad ", si el producto no lo es se están colocando problemas a los usuarios finales en el manejo de las aplicaciones, en vez de proveerles soluciones.

La competencia mundial de productos de calidad demuestra la necesidad de contar con pruebas en los productos desde la primera etapa de concepción del mismo, ya que realizar una revisión de un proyecto de software ya terminado duplica o triplica los costos de inversión con respecto a las efectuadas desde un inicio.

Conclusiones.

Después de haberse realizado las investigaciones pertinentes sobre la calidad de un software, 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. Además en este capítulo se puntualizó cuales eran los niveles de prueba, los métodos, las técnicas y las herramientas escogidas para llevar a cabo las pruebas. Estos fueron:

- Nivel de Sistema.
- Método de Caja Negra.
- Técnicas Funcionales, Técnicas de Carga y Técnica de Stress.
- Herramienta JMeter.

Estas pruebas en el software son la clave para su éxito, pues se encargan de detectar los errores. Estas requieren de una planificación y de una estrategia para realizarlas. Es importante recalcar que deben realizarse en todas las fases de desarrollo del software. La etapa de prueba es tan o más importante que todas las realizadas hasta el momento puesto que en ella se refleja la calidad con que ha sido llevada a cabo la proyección del sistema. Teniendo en cuenta los resultados de la investigación y estando definido ya los niveles, métodos, técnicas y herramientas a utilizar se da paso al diseño y la aplicación de las pruebas del software.

CAPITULO 2. DISEÑO Y APLICACIÓN DE LAS PRUEBAS AL SOFTWARE.

Introducción.

El diseño y la aplicación de las pruebas de software son puntos vitales dentro del proceso de avance de las pruebas, pero antes de describir estos puntos es necesario tener conocimientos del sistema que se pretende revisar. Cada proceso de prueba se inicia con la elaboración del plan de prueba en el cual queda registrado el propósito de las pruebas así como los recursos y pasos a seguir para su desarrollo. Dentro del proceso de realización de las pruebas la planificación es un eslabón importante ya que de esta forma se establece una estrategia para llevar a cabo la prueba. Además de estos aspectos se argumentan las técnicas, métodos y herramientas utilizadas; así como el diseño de los casos de pruebas y la ejecución de las mismas.

2.1 Descripción del sistema a probar.

En la actualidad, existen grandes cantidades de centros que enfocan sus investigaciones al campo de la genética. Particularmente en nuestro país uno de ellos es el CIGB. Su impacto está destinado a la salud humana, las producciones agropecuarias, acuícola, y al medio ambiente. Debido a la gran importancia de las investigaciones que allí toman lugar ha surgido la idea de crear una aplicación Web capaz de llevar a cabo el estudio de los genes, así como efectuar un diseño de un siRNA más estable y seguro. Esta aplicación consta de un solo módulo que abarca todas las funcionalidades necesarias para el diseño de un siRNA. Este módulo cuenta con 4 casos de uso y 20 requisitos funcionales.

2.1.1 Descripción de los Casos de Uso del Sistema.

A continuación se hace alusión a la descripción de los 4 casos de uso del sistema los cuales son arquitectónicamente significativos.

Analizar Secuencia mRNA.

El caso de uso se inicia cuando luego de haber buscado y mostrado determinado gen o transcrito, el especialista decide analizar la secuencia encontrada o accede a la opción “Analize mRNA sequence”, procede a analizarla seleccionando las reglas y el tipo de análisis deseado, para de esta manera obtener el diseño de siRNA esperado.

Este caso de uso tiene como precondición haber realizado el CU Mostrar_Gen y el CU Mostrar_Transcrito.

Mostrar_Gen.

El caso de uso se inicia cuando el especialista escoge una de las opciones de búsqueda de secuencias ('By gene id' o 'By gene symbol') y desea conocer más información sobre los genes, sin necesidad de realizar un análisis previo. Entonces específica un criterio de búsqueda, a partir del mismo se busca en la base de datos la información necesaria para que sea consultada por los especialistas.

Mostrar_Transcrito.

El caso de uso se inicia cuando luego de haber generado a partir de la especificación de un criterio de búsqueda la secuencia del gen requerido se muestran los transcritos asociados al mismo, el especialista escoge un transcrito y del mismo se muestra su información.

Este caso de uso tiene como precondición haber realizado CU Mostrar_Gen.

Mostrar_miRNA.

El caso de uso se inicia cuando el especialista solicita al sistema buscar un gen determinado este muestra sus datos de los miRNA correspondientes.

Este caso de uso lleva como precondición haber realizado el CU Mostrar_Gen.

2.1.2 Requisitos Funcionales del sistema.

Los requisitos funcionales definidos para el buen funcionamiento de la aplicación son:

R 1: Buscar transcritos asociados a un gen.

R 1.1: Buscar gen.

R 1.2: Mostrar transcritos asociados al gen.

R 1.3 Mostrar cantidad de transcritos asociados.

R 2: Buscar datos de un transcrito indicado.

R 2.1: Seleccionar transcrito.

R 2.2: Buscar transcrito especificado.

R 2.3: Mostrar gráfica con el nombre, posición inicial y posición final del transcrito e id del gen al que pertenece.

R 2.4: Mostrar exones, diferenciando zona codificante de zona no traducida.

R 2.5 Mostrar SNPs.

R 2.6 Mostrar secuencia de nucleótidos.

R 3: Realizar análisis de las secuencias mRNA.

R 3.1: Buscar sitios blancos en la secuencia mRNA.

R 3.2: Mostrar gráfica de la secuencia de mRNA resaltando sitios blancos.

R 3.3: Mostrar subcadena de los sitios blancos, con posición inicial y final.

R 3.4: Mostrar, en caso de existencia de los polimorfismos de nucleótido simple (SNP), la subsecuencia mRNA con los SNP.

R 3.5: Mostrar resultados ordenados por reglas cumplidas.

R 4: Buscar miRNA.
R 4.1: Mostrar id, score y pvalue del miRNA.
R 4.2: Mostrar la posición del gen donde el miRNA actúa.

2.2 Plan de Pruebas.

El plan de prueba tiene como objetivo fundamental señalar el enfoque, la estrategia de prueba, los recursos y el esquema de actividades de prueba, así como la organización del equipo de prueba, la arquitectura técnica, además de las especificaciones de software y de hardware. Se realiza una leve descripción del plan de prueba donde se describen los requisitos y los casos de uso, se detallan los elementos a probar, las características, las actividades de prueba, el personal responsable y los riesgos asociados, así como se documentan los resultados obtenidos.

El propósito del plan de pruebas es dejar de forma explícita el alcance, el enfoque, los recursos requeridos, el calendario, los responsables y el manejo de riesgos de un proceso de pruebas.

Está constituido por un conjunto de pruebas. Cada prueba debe:

- Dejar claro qué tipo de propiedades se quieren probar.
- Dejar claro cómo se miden los resultados.
- Especificar en qué consiste la prueba.
- Definir cual es el resultado que se espera.

El plan de prueba constituye una guía para llevar a cabo las pruebas pues este es el primer paso a tener en cuenta antes de efectuar las pruebas.

Para más información sobre el mismo ver el documento Plan de Prueba.doc.

2.2.1 Estructura del Plan de Pruebas.

1. Introducción.

La introducción consta de tres elementos fundamentales los cuales son:

Alcance.

Definición, Acrónimos y Abreviatura.

Referencias.

Estos elementos reflejarán a que producto se le estará realizando la planificación de las pruebas, en caso de que exista alguna palabra de la cual no se tenga conocimientos se aclarará, y estarán las referencias a cada documento que se necesite para este proceso.

2. Organización del equipo de pruebas.

En este punto se da una leve descripción de los roles y las actividades que estos deben desarrollar.

3. Arquitectura técnica.

Se muestra el modelo de despliegue el cual se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos.

4. Especificaciones de Software y de Hardware.

Se muestran los elementos que se deben de tener para un buen funcionamiento de la aplicación. Estos elementos son requisitos que se tiene que cumplir tanto de hardware como de software.

5. Descripción del Plan de Prueba.

Se describen todos los requisitos y casos de uso que deben tenerse en cuenta para el perfecto funcionamiento del software, estos requisitos son: funcionales, de diseño y de integración.

6. Estrategia de Prueba.

La Estrategia de Prueba se puede definir como uno de los puntos más importantes dentro del proceso de desarrollo del plan de prueba. Este punto describe los objetivos, técnicas, entorno de trabajo, proceso con el cual se llevará a cabo la prueba, diseños de casos de pruebas y herramientas para llevar a cabo este proceso.

7. Recursos Requeridos.

Los recursos requeridos son aquel personal capacitado con sus determinadas tareas a realizar.

8. Plan de Proyecto.

Este plan no es más que el cronograma para llevar a cabo las tareas necesarias en un tiempo determinado. Estas tareas son llevadas a cabo por una persona X. Esto se realiza para el perfecto funcionamiento de las pruebas, pues este punto es una guía para el desarrollo de las mismas.

9. Calendario y Plazos.

El calendario es aquel tiempo que realmente se utiliza del día para llevar a cabo el proceso de las pruebas. Los plazos no son más que el tiempo utilizado en otras tareas que por X motivo no se pudo realizar las pruebas.

10. Definición de los Entregables.

Aquí es donde se definen todos aquellos documentos que se tienen que entregar después de haber realizado las pruebas.

11. Seguimiento y Reporte de Defectos.

Se especifica en que planilla se le dará seguimiento a los defectos encontrados además se les dará una categoría según su importancia.

12. Aprobación del Plan.

En este punto se dice si queda aprobado o no, el Plan de Prueba y en cualquiera de los dos casos se dice por quien quedó o no aprobado.

13. Documentación de los Resultados.

Se documentan los resultados después de haber realizado las pruebas.

14. Riesgos.

En este punto se identifican los riesgos conocidos y predecibles que afectan al funcionamiento de la aplicación por ejemplo: fallo del sistema eléctrico, pérdida de disco duro, fallo de la red, etc.

2.2.2 Cronograma.

Este cronograma fue desarrollado en el Plan de Prueba, para desglosar todas las actividades que se llevaron a cabo para ejecutar la estrategia de prueba, consta de tres aspectos fundamentales que se detallan a continuación:

Actividades: Es el evento que va a ser realizado.

Inicio-Fin: Es la fecha de inicio y fin de dicho evento.

Responsable: Es la persona encargada de desarrollar dicho evento.

Actividades.	**Inicio- Fin.**	**Responsable.**
Elaboración del plan de pruebas.	6/12/09 al 10/12/09	Yanet Pérez
Revisión del documento de Especificación de requisitos. (1era iteración)	10/12/08	Yanet Pérez
Elaboración de las No conformidades pertenecientes a este documento.	10/12/08	Yanet Pérez
Revisión del manual de BiosiRNA. (1era iteración)	11/12/08	Rafael Leyva
Elaboración de las No conformidades pertenecientes a este documento.	11/12/08	Rafael Leyva
Revisión del modelo de casos de usos del sistema. (1era iteración)	12/12/08	Yanet Pérez
Elaboración de la Planilla de No conformidades pertenecientes a este documento.	12/12/08	Yanet Pérez
Revisión del diccionario de datos. (1era iteración)	13/12/08	Rafael Leyva
Elaboración de la Planilla de No conformidades pertenecientes a este documento.	13/12/08	Rafael Leyva
Prueba de regresión. (1era iteración)	4/1/09	Rafael Leyva
Revisión del documento de Especificación de requisitos. (2era iteración)	5/1/09	Yanet Pérez
Elaboración de la Planilla de No conformidades pertenecientes a este documento.	5/1/09	Yanet Pérez
Revisión del manual de BiosiRNA. (2era iteración)	6/1/09	Rafael Leyva
Elaboración de la Planilla de No conformidades pertenecientes a este documento.	6/1/09	Rafael Leyva
Revisión del modelo de casos de usos del	7/1/09	Yanet Pérez

sistema. (2era iteración)		
Elaboración de la Planilla de No conformidades pertenecientes a este documento.	7/1/09	Yanet Pérez
Revisión del diccionario de datos. (2era iteración)	8/1/09	Rafael Leyva
Elaboración de la Planilla de No conformidades pertenecientes a este documento.	8/1/09	Rafael Leyva
Prueba de regresión. (2da iteración)	12/1/09	Rafael Leyva
Realización de las Pruebas Exploratorias.	13/1/09	Yanet Pérez Rafael Leyva
Elaboración de los Diseños de Casos de Pruebas. (1era iteración)	14/1/09 al 30/1/09	Rafael Leyva
Realización de prueba Funcionales. (1era iteración)	31/1/09 al 5/2/09	Yanet Pérez Rafael Leyva
Elaboración de la Planilla de No conformidades pertenecientes a esta iteración.	31/1/09 al 5/2/09	Yanet Pérez Rafael Leyva
Prueba de regresión. (1era iteración)	9/2/09	Yanet Pérez Rafael Leyva
Elaboración de los Diseños de Casos de Pruebas. (2era iteración)	9/2/09 al 10/2/09	Rafael Leyva
Realización de prueba Funcionales. (2era iteración)	10/2/09 al 12/2/09	Yanet Pérez Rafael Leyva
Elaboración de la Planilla de No conformidades pertenecientes a esta iteración.	10/2/09 al 15/2/09	Yanet Pérez Rafael Leyva
Realización de pruebas de carga y de stress.	15/2/2/09 al 20/2/09	Yanet Pérez Rafael Leyva
Análisis de los resultados arrojados después de la realización de estas	20/2/09 al 25/2/09	Yanet Pérez Rafael Leyva

pruebas.		
Elaboración de las Listas de Chequeo.	25/2/09 al 28/209	Yanet Pérez Rafael Leyva
Evaluación del producto utilizando las listas de chequeo.	1/3/09 al 5/3/09	Yanet Pérez Rafael Leyva

2.3 Descripción de la Estrategia de las Pruebas realizada.

La Estrategia de Prueba de Software integra un conjunto de actividades que describen los pasos que hay que llevar a cabo en un proceso de prueba como son: la planificación, el diseño de casos de prueba, la ejecución y los resultados, tomando en consideración cuánto esfuerzo y recursos se van a requerir, con el fin de obtener como resultado una correcta construcción del software. Antes de comenzar a explicar cómo fue que se realizaron las pruebas es preciso decir que estas fueron realizadas en el nivel de sistema, ya que se pretende probar el correcto funcionamiento de la aplicación.

En primer lugar, antes de comenzar a realizar las pruebas al sistema se revisa la documentación, para esto es necesario revisar las planillas: Modelo de Casos de uso del sistema, Manual BiosiRNA-Desing, Diccionario de Datos y Especificación de Requisitos. Los errores obtenidos después de haber realizado las pruebas a estos documentos se guardan en la planilla de no conformidades (Ver Anexo 2) la cual es enviada a los desarrolladores de la aplicación para su posterior corrección.

En segundo lugar se realizan las pruebas exploratorias las cuales son pruebas superficiales realizadas al sistema, estas pruebas se realizan sin tener ningún conocimiento sobre dicha aplicación y los errores obtenidos durante esta revisión se documentan en una planilla de no conformidades.

En la prueba de sistema se verifica las funcionalidades del sistema mediante el método de Caja Negra para el desarrollo del mismo se basan en el criterio “Particiones Equivalentes” el cual permite reducir el posible conjunto de diseños de casos de pruebas. En este método se examinan las especificaciones que señalan, lo que el programa debe hacer y como lo debe llevar a cabo. Para ejecutarlos se desarrollan diseños de casos de prueba reales para cada condición o combinación de condiciones y se analizan los resultados que arroja el sistema para cada uno de los casos, recogiendo en la planilla de no conformidades dichos resultados.

También se aplica las técnicas de pruebas carga y estrés, las cuales verificará el rendimiento de la aplicación. Estas técnicas serán aplicadas con el uso de la herramienta JMeter la cual es utilizada en aplicaciones Web. Esta herramienta permite detectar aquellos puntos que se quieren probar con respecto a la concurrencia de usuarios y velocidad de carga en el sitio además se puede utilizar para pruebas funcionales y de regresión en la aplicación.

2.3.1 Nivel de Prueba seleccionado.

El nivel de prueba seleccionado para desarrollar las pruebas a la aplicación informática siRNA, es el nivel de sistema el cual es el encargado de verificar que cada elemento encaja de forma adecuada y que se alcance la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Algunas de estas pruebas son:

Pruebas Exploratorias: En estas, el sistema es probado sin tener ningún previo conocimiento del mismo, es decir son pruebas a ciegas.

Pruebas basadas en el Método de Caja Negra: Estas pruebas son importante en este nivel pues estas revisan detalladamente cada caso de uso contra la aplicación.

Prueba de rendimiento: Prueba el rendimiento del software en tiempo de ejecución.

2.3.1.1 Método de Prueba Utilizado.

La prueba funcional tiene como objetivo asegurar el apropiado trabajo con los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de los resultados. Esta prueba utiliza el **método de caja negra** en el cual se ejecuta cada caso de uso, flujo de caso de uso o función usando datos válidos e inválidos, para verificar los siguientes puntos:

- Las funciones del software son operativa.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene.

Para confeccionar los diseños de casos de pruebas de Caja Negra se utilizó el criterio “Partición Equivalente”.

La partición equivalente se dirige a la definición de diseños de casos de prueba que descubran clases de errores, reduciendo así el número total de diseños de casos de prueba que hay que desarrollar. El objetivo de este criterio es reducir el posible conjunto de diseños de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software. Con esto se diseña un caso de prueba por cada uno de los casos de uso existentes del producto. Luego de haber realizado los diseños de casos de pruebas se procede a la realización de los mismos verificando siempre cada punto a probar del diseño de caso de prueba después de realizadas dichas pruebas se documentan en una planilla los resultados obtenidos para su posterior evaluación.

2.3.1.1.1 Diseños de Casos de Pruebas.

El diseño de casos de prueba está totalmente mediatizado por la imposibilidad de probar exhaustivamente el software. En consecuencia, las técnicas de diseño de casos de prueba tienen como objetivo conseguir una confianza aceptable en que se detectarán los defectos existentes sin necesidad de consumir una cantidad excesiva de recursos.

Como no se pueden probar todas las posibilidades de funcionamiento del software, 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.

La dificultad de esta idea es saber elegir los casos que se deben ejecutar. De hecho, una elección puramente aleatoria no proporciona demasiada confianza en que se puedan detectar todos los defectos existentes.

Para más información sobre el diseño de casos de pruebas, ver los documentos:

Diseño de Caso de Prueba mostrar_gen.doc
Diseño de Caso de Prueba Mostrar_Transicrito.doc

Diseño de Caso de Prueba Mostrar_miRNA.doc
Diseño de Caso de Prueba mostrar_Secuencia_mRNA.doc

2.3.1.1.1.1 Estructura del Diseño de Caso de Prueba.

1. Descripción General:

Se describe el caso de uso con el que voy a trabajar y cuales son las especificaciones necesarias para el desarrollo del mismo.

2. Condiciones de Ejecución:

Son las condiciones que hay que tener en cuenta para su funcionamiento (precondiciones).

3. Secciones a Probar:

En este punto se da una descripción de lo que realmente se debe hacer el caso de uso. Para esto se llena una tabla la cual posee: nombre de la sección, escenario de la sección, descripción de la funcionalidad y flujo central. Estos cuatro puntos son con los que se verificará la ejecución de las pruebas.

En este punto se describen las variables que serán probadas así como los valores válidos e inválidos que pueden tomar con su comportamiento en el sistema.

4. Registro de defectos y Dificultades detectadas.

Se registra en una tabla las no conformidades detectadas después de haber probado las variables.

Para mayor información ver Anexo 2.

2.3.1.2 Técnicas de Pruebas Utilizadas.

2.3.1.2.1 Técnica Funcional.

La técnica funcional está incluida dentro de las pruebas funcionales al software, esta técnica fue una de las escogidas para desarrollar en el proceso de liberación de la aplicación informática siRNA, esta se encarga de verificar si el comportamiento del software cumple o no con sus especificaciones. Además está relacionada con la validación de todas las funcionalidades del software. Dicha técnica fue empleada a la hora de realizar las Pruebas Exploratorias y las Pruebas basadas en el Método de Caja Negra.

2.3.1.2.2 Técnicas de Carga y Stress.

Las técnicas de carga y de stress están incluidas dentro de las pruebas de rendimientos, estas son pruebas realizadas desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. También puede servir para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos. Una vez el problema definido, se puede proporcionar soluciones para solucionar estos problemas siendo capaces de probar su programa informático bajo condiciones múltiples, a fin de definir cómo éste reacciona bajo condiciones predeterminadas. Además se puede comparar las capacidades de su programa informático y compararlos a sus objetivos de resultados. Se pueden definir los problemas y sus causas, y comprobar sus fiabilidades bajo condiciones de tensión.

La prueba de carga es el tipo más sencillo de pruebas de rendimiento. Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperada. Esta 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.

Mientras que las pruebas de stress se utilizan normalmente para provocar un fallo en la aplicación. Se va doblando el número de usuarios que se agregan a la aplicación y se ejecuta una prueba de carga hasta que se rompe. Este tipo de prueba se realiza para determinar la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada.

Estas técnicas traen como ventajas que a partir de estas se puede definir los problemas de resultados y sus causas, se puede aumentar las capacidades de las aplicaciones, además asegura que los cambios no afecten a los resultados. Estas técnicas serán llevadas a cabo haciendo uso de la herramienta JMeter.

2.3.1.2.2.1 Herramientas de Trabajo Utilizadas para las pruebas de Carga y de Stress.

Apache JMeter es una herramienta de carga diseñada para realizar Pruebas de Rendimiento sobre Aplicaciones Web.

Originalmente el Apache JMeter fue diseñado para realizar pruebas de estrés sobre aplicaciones web. Sin embargo hoy en día su arquitectura ha evolucionado, ahora no sólo puede llevar a cabo pruebas en componentes típicos de Internet (HTTP), sino que puede realizar también pruebas sobre Bases de Datos, scripts Perl, objetos java, servidores FTP y prácticamente cualquier medio de los que se pueden encontrar en la red.

Para un óptimo desarrollo de pruebas, es necesario tener ciertas nociones funcionales de la aplicación que se va a evaluar. Si esto no es así, las pruebas no serán completamente satisfactorias pues no se sabe por ejemplo si ha devuelto la respuesta apropiada a la petición hecha o si nos ha permitido acceder con una contraseña no apropiado.

Esta herramienta está diseñada para desarrollar diferentes tipos de pruebas; permitiendo diseñar tanto sencillas pruebas que soliciten simples páginas web, como complejas secuencias de requisiciones que permitan evaluar el comportamiento de una aplicación o como la capacidad de carga máxima que pueda tener una aplicación en un servidor.

JMeter también permite la ejecución de pruebas distribuidas entre distintos ordenadores, para realizar pruebas de rendimiento.

El Apache JMeter incluye una interfaz gráfica de usuario que facilita el diseño de las pruebas. Este interfaz gráfico además de aportar un entorno cómodo de trabajo, también permite guardar y alterar tanto las pruebas desarrolladas como los componentes que lo integran. Gracias a esto se pueden reutilizar las pruebas o módulos de las mismas en el desarrollo de nuevas pruebas.

Además de las funcionalidades de prueba antes mencionadas, el Apache JMeter ofrece la posibilidad de activar un Proxy web. Gracias a esto se puede grabar la navegación de un usuario para posteriormente usarla en la generación de una prueba.

2.3.1.2.2.1.1 Pasos a seguir para efectuar las pruebas con la Herramienta JMeter.

Los pasos a seguir para el proceso de automatización de las pruebas de rendimiento llevadas a cabo con la herramienta JMeter fueron:

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 y se almacenan los resultados 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.

2.4 Estrategia de Trabajo.

La estrategia de trabajo es una guía para llevar a cabo todo el proceso de prueba. Dado el criterio emitido por Pressman en su edición del 2002 sobre la estrategia de prueba se ha desarrollado la estrategia de trabajo, esta consta de los siguientes pasos:

- Realización de un estudio bibliográfico sobre:
  - Pruebas de Caja Negra.
  - Pruebas de Carga y Stress.

- Diseñar Plan de Pruebas.

  - Objetivos de las pruebas.
  - Recursos de hardware.
  - Pasos a seguir para la realización de las pruebas.
  - Estrategia de pruebas.

- Revisión de la documentación, en este punto se revisan las planillas:

  - Modelo de Casos de uso del sistema.
  - Manual BiosiRNA-Desing.
  - Diccionario de Datos.
  - Especificación de Requisitos.

  - Se documentan las No Conformidades existentes.

- Definimos los Niveles de Pruebas.
- Realización de las Pruebas Exploratorias.
  - Se documentan las No Conformidades existentes.
- Diseño de Casos de Pruebas.
- Realización de las Pruebas de Caja Negra.
  - Se documentan las No Conformidades existentes.
- Realización de las Pruebas de Carga y de Estrés.
  - Se hace uso de la Herramienta JMeter.
  - Se documentan las No Conformidades existentes.
- Evaluación del Producto, teniendo en cuenta las listas de chequeo.

Conclusiones.

En este capítulo se diseñó y aplicó las pruebas pertinentes a dicha aplicación. Con estas pruebas se hizo una revisión detallada del sistema y de la documentación, además quedó elaborado el plan de prueba y se argumentaron aquí los niveles, las técnicas, métodos y herramientas escogidos para aplicarle al software en el capítulo anterior. Para el diseño de estas pruebas se tuvo presente un conjunto de puntos los cuales facilitaron las pruebas y ayudaron a una mejor calidad de las mismas.

Algunos de estos puntos son:

- Las pruebas intentan descubrir la presencia de errores, no la ausencia de los mismos.
- Las pruebas de Caja Negra se basan en la especificación del sistema.
- Los errores en la interfaz están relacionados principalmente con errores en los parámetros, no comprensión de la especificación o asunciones temporales inválidas.

A partir de los puntos anteriores se tuvo en cuenta que los errores tienen que ser documentados para luego realizar la evaluación de los mismos, este tema será tratado en el próximo capítulo.

CAPÍTULO 3. EVALUACIÓN DE LOS RESULTADOS.

Introducción.

Después de aplicar el proceso de mejora un número determinado de veces, se llegó a obtener la calidad deseada en el producto de software. En este capítulo, se llevará a cabo un análisis de los resultados obtenidas del producto, así como la identificación y la cuantificación de las mejoras obtenidas durante el proceso. Este análisis deberá registrarse en documentos, ya que servirá para mejorar la calidad de productos de software futuros. También se evaluará el producto en cuanto a la confiabilidad, seguridad, portabilidad y usabilidad.

3.1 Estructura de las No Conformidades.

Se describen los aspectos a tener en cuenta a la hora de analizar los resultados de las pruebas. Este punto cuenta con tres aspectos:

- Aspectos Generales.
- Elementos probados.
- Elementos no probados y causas.

1. Tablas de No Conformidades Detectadas.

Dicha tabla presenta 7 aspectos a llenar, estos son:

Elementos: Elemento que se revisa, por ejemplo: Documentación NOMBREPROYECTO.

No: Número de la no conformidad.

No Conformidad: Aquí se describe el error como tal.

Aspecto Correspondiente: Es la parte especifica donde se encontró dicho error.

Etapa de Detección: Es la etapa de detección que se encontró el error, por ejemplo: Revisión de la documentación, Pruebas Exploratorias, Pruebas Funcionales.

Clasificación: Las no conformidades se clasifican en S (significativa), N (no significativas), R (recomendaciones).

Estado de la no Conformidad: Se coloca el estado de la NC y la fecha, cada vez que se revise se deja el estado anterior y se coloca el nuevo con la fecha en que se revisó. RA: Resuelta, PD: Pendiente, NP: No Procede.

Respuesta del equipo de desarrollo: Se llena a partir de la última iteración, y es responsabilidad del equipo de desarrollo, quien especifica la conformidad con lo encontrado y en caso de no proceder la no conformidad explica por qué.

2. Anexos:

   Si existen, se colocan fotos o algo que tenga que ver con la No Conformidad.

Para más información sobre dicha planilla ver Anexo 3.

3.2 Análisis de los Resultados de las Pruebas.

El valor que puede proporcionarse a un software en el mercado puede ser aumentado o disminuido por la respuesta de la dirección del proyecto a una no conformidad. Asegurando que el personal involucrado haya realizado satisfactoriamente la corrección, análisis de las causas y acción correctiva, se incrementará la probabilidad de que el software resultante logre la satisfacción del cliente.

Durante la realización de las pruebas internas llevadas a cabo al proyecto de siRNA se encontraron una serie de no conformidades.

	No conformidades significativas	No conformidades No significativas	Total	Iteraciones
Revisión de la documentación	66	20	86	2

Revisión de la Aplicación.	8	0	8	2

Tabla 3.1 No Conformidades por Iteraciones.

Además se encontraron errores de rendimiento en las pruebas de carga y de stress, estos errores son muy importantes para el buen funcionamiento de la aplicación.

3.2.1 Clasificación y Valoración de las No Conformidades por Tipos de Pruebas.

Las pruebas realizadas hasta el momento arrojaron una cierta cantidad de errores los cuales se pueden apreciar en la tabla anterior. En esta tabla se muestra la cantidad de no conformidades significativas y no significativas según las pruebas ejecutadas. Dichas pruebas fueron realizadas con éxito pues lo más importante a la hora de realizar estas pruebas es que arrojen la mayor cantidad de errores posible.

Las revisiones realizadas a la documentación resultaron satisfactorias pero se encontraron varios errores que fueron detectados y debidamente informados al equipo de desarrollo., luego de esto la documentación quedó con mejor presencia y legibilidad, por lo que se puede reafirmar que esta prueba fue un éxito.

A continuación detallamos algunos de los principales errores encontrados en las revisiones a la documentación:

- Faltas de ortografía en palabras como: críterios, asi, critico, posición, iniciañ, asciados, mustranlos, determiando, especislistas, asuario, informacion, axistencia, detallesde, area, imagenes, sequencia.
- Uso incorrecto de preposiciones, artículos y conjunciones, en algunos casos frases y palabras fuera de contexto.
- Algunas incoherencias, por ejemplo: En el documento Modelo de casos de uso del sistema no se corresponde el nombre del caso de uso del diagrama con el que aparece en la descripción del propio caso.

En cuanto a la aplicación, se probaron detalladamente todos los diseños de casos de prueba definidos para verificar su comportamiento, como resultado no arrojaron errores significativos para la ejecución de la aplicación, aunque se detectaron algunos con respecto a las descripciones de los casos de uso, pues no se encontraban bien redactados y existían algunas contradicciones entre la aplicación y dicha descripción. Ejemplo: En la descripción del caso de uso mostrar_gen aparece que el gen se busca mediante el criterio nombre del gen o id del gen y en la aplicación los criterios de búsqueda son By gene id o By gene simbol.

3.2.2 Análisis de los resultados de las Pruebas de Carga y de Stress.

Las pruebas de carga y de stress se desarrollaron adecuadamente, pues después de haber realizado las pruebas al software con la herramienta JMeter, estas arrojaron una serie de errores. La aplicación siRNA aparentemente carecía de errores pero luego de haber ejecutado las pruebas se mostraron algunas deficiencias en el sistema.

La herramienta JMeter primeramente se utilizó para simular la conexión de 20 usuarios navegando simultáneamente, esta cuota está por debajo de la cantidad de usuarios previstos. Posteriormente a esto se desarrollo otra simulación para un total de 70 usuarios todos estos navegando simultáneamente sobre el sistema.

Los datos obtenidos después de haber realizado las pruebas se recogieron en el Informe Agregado el cual es el encargado de exponer el estado en que se encuentra el software con respecto a los escenarios probados. Este artefacto presenta los siguientes aspectos:

- 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 total que demoraron las petición en cargarse.
- Mediana: Tiempo promedio que han tardado en cargarse las paginas.
- Línea 90 %: Tiempo máximo en que corrieron el 90 por ciento de las peticiones reales, o sea, el tiempo más probable que se puede demorar una petición.
- Min: Tiempo mínimo que ha demorado en cargarse una página.
- Max: Tiempo Máximo que ha tardado en cargarse una página.

- %Error: Por ciento de error de las paginas que no se llegaron a cargar de manera satisfactoria.
- Kb/Seg: Velocidad de carga de las paginas.

A continuación se hace un análisis de los errores encontrados en dichas simulaciones.

Pruebas efectuadas para 20 usuarios navegando simultáneamente.

Las pruebas efectuadas para 20 usuarios navegando simultáneamente arrojaron resultados satisfactorios, puesto que una prueba nunca es buena sino arroja errores. Los defectos encontrados fueron en imágenes pues estas se demoraron para cargar, las que presentaron un 100% de error lo que puede traer consigo una colisión del sistema. A continuación se mostrará un resumen de uno de los artefactos generados después de haber realizado dichas pruebas. Este artefacto es el Informe Agregado.

	Muestras	Media	Mediana	Línea 90%	Min	Max	%Error	Rendimiento	Kb/sec
TOTAL	524	3274	250	11812	15	30641	29.01	4.4/sec	193319.8

Tabla 3.2 Resumen de Resultados Arrojados por el Informe Agregado para la simulación de 20 usuarios.

Para mayor información estos resultados Ver Anexo 4.

Pruebas efectuadas para 70 usuarios navegando simultáneamente.

Las pruebas realizadas para simular la conexión de 70 usuarios navegando simultáneamente sobre el sistema también fueron un éxito, pues ayudaron a evaluar el sistema bajo condiciones de carga extrema.

Los defectos encontrados fueron al cargar las fotos en la aplicación pues estas se tardaron en ser mostradas, en este caso las imágenes fueron las mismas que en el ejemplo anterior. A continuación se mostrará un resumen de los resultados arrojados por el Informe Agregado.

	Muestras	Media	Mediana	Línea 90%	Min	Max	%Error	Rendimiento	Kb/sec
TOTAL	1322	8347	4078	23984	31	75453	27.69	5.0/sec	144991.9

Tabla 3.2 Resumen de Resultados Arrojados por el Informe Agregado para la simulación de 70 usuarios.

Para más información sobre estos resultados Ver Anexo 5.

3.3 Evaluación del Producto.

Para validar y verificar la calidad total del producto final existen varias técnicas de pruebas como la seguridad, la portabilidad, la usabilidad y la confiabilidad para verificar el cumplimiento de estas técnicas se emplearon las listas de chequeo estas constituyen un listado de preguntas, en forma de cuestionario que sirven para verificar el grado de cumplimiento de determinadas reglas establecidas con un fin dado.

El empleo de estas listas está generalizado en usos muy diversos que van desde verificar y determinar el potencial de mercados extranjeros hasta medir la confiabilidad y seguridad de sistemas informáticos, incluyendo aspectos tales como la evaluación de criterios de usabilidad de un sitio de Internet, como la verificación de un plan de vuelo en aeronáutica, siendo estos solo algunos usos para ejemplificar el amplio espectro. La lista de chequeo cubre las áreas de problemas más comunes.

Su objetivo es asistirle examinando cuidadosamente todas las áreas importantes y considerando que las mejoras pueden ser planificadas. Usar la lista de chequeo no resolverá sus problemas, pero puede ser un paso hacia la identificación y realización de mejoras en el software. Es necesario a la hora de evaluar el software tener presente los atributos de calidad estos abarcaran lo referente a su buen uso.

Estos atributos son:

Adaptabilidad: Es la capacidad del programa para adaptarse a medios ambientes diferentes a aquellos en los cuales fue diseñado o pensado para operar. Las computadoras se hacen obsoletas rápidamente, cada vez que cambiamos de computadora viene con un sistema operativo diferente y nuevas herramientas de escritorio, a lo cual cada sistema debería de adaptarse.

Confiabilidad: La confiabilidad de un sistema de software se define como la probabilidad de que este sistema satisfaga una función para un número especifico de los ensayos de entrada, bajo condiciones de entrada especificas, en un intervalo especifico de tiempo.

Disponibilidad: La disponibilidad es el grado en el que un sistema o componente es operacional y accesible cuando es requerido para su uso. Se describe también como el tiempo en el que un recurso particular es accesible y utilizable.

Extensibilidad: La extensibilidad permite que las modificaciones requeridas puedan ser hechas en localizaciones apropiadas sin efectos secundarios indeseables.

La extensibilidad de un sistema de software depende de su:

- Modularidad del sistema de software.
- Posibilidades que el lenguaje de implementación proporciona para este propósito.
- Legibilidad del código.
- Disponibilidad de la documentación del programa.

Mantenibilidad: Propiedad de un sistema que representa la cantidad de esfuerzo requerido para conservar su funcionamiento normal o para restituirlo una vez que se ha presentado un evento de falla.

Portabilidad: La portabilidad es un conjunto de atributos que tienen que ver con la capacidad del software de ser transferido de un ambiente a otro. Dichos atributos son:

- Instalabilidad.
- Conformidad.
- Reemplazabilidad.
- Adaptabilidad.

Reusabilidad: Algunas partes o todo el software se puede reutilizar en otros proyectos. Esto significa que el software es modular, que cada módulo del software individual tiene una interfaz bien definida, y que cada módulo individual tiene un resultado bien definido de su ejecución.

Usabilidad: Es la medida en la que un producto se puede usar por determinados usuarios para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado.

La calidad es sinónimo de todos estos conceptos pues cada uno de ellos aportan ideas al concepto de calidad .En el proceso de realización de las pruebas se evaluó el producto en cuanto a estos atributos de calidad observables durante la ejecución.

Para llevar a cabo esta evaluación se basan en una serie de preguntas las cuales tienen un nivel de importancia (!! Muy importante (5) o ! Menos importante (3)), una evaluación (-- Malo (0) o -Satisfactorio (2) o + Bien (4) o ++ Excelente (5)), se dice si esta procede o no, y un comentario.

3.3.1 Usabilidad.

Para evaluar la **usabilidad** del producto en específico, como esta se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso, se basan en las siguientes preguntas:

Nivel	**Evaluación**	**Eval**	**NP**	**Comentario**
!	¿Se consideran otros idiomas para las instrucciones en la pantalla de forma adecuada?		NP	Tiene un solo idioma, la aplicación esta en ingles.
!	¿Es simple el vocabulario utilizado?	4		
!	¿Se proporciona tiempo suficiente para realizar las entradas por teclado?	5		
!	¿Se posibilita la opción de incrementar el tamaño de los caracteres?	0		El sistema no brinda la opción de incrementar el tamaño de los caracteres ya que cuenta con una interfaz estable.

!	¿Hay algún tipo de asistencia para los usuarios que hacen uso del sistema por primera vez?	0		No tiene ningún asistente que le facilite al usuario entender la aplicación.
!	¿Resulta fácil instalar el software?	4		
!	¿Se entienden la interfaz y su contenido?	4		
!	¿Resulta fácil especificar un objeto o una acción?	5		
!	¿Resulta fácil entender el resultado de una acción?	5		
!!	¿Está diseñada la interfaz para facilitar la realización eficiente de las tareas de la mejor forma posible?	4		
!	¿Es fácil de utilizar el sistema en la realización de tareas?	4		
!	¿Facilita la interfaz de usuario un uso eficiente (para interfaces basadas en pantallas)?	2		
!!	¿Se muestran todas las imágenes y marcos correctamente?	0		Las imágenes no las carga correctamente.
!	¿Es apropiada la retro alimentación presentada por el sistema?		NP	El sistema no se retroalimenta.
!	¿Actúa el sistema en la prevención de errores?	5		
!!	¿Actúa el sistema en la información de los errores?	4		
!	¿Actúa el sistema en la corrección de errores?	0		El sistema no corrige los errores, solo los detecta.
!	¿Se usan adecuadamente los modos de trabajo de los usuarios en el software?	4		

!	¿Se permite la utilización del ratón o el teclado?	5		
!	¿Se utiliza mensajes y textos descriptivos?	4		
!!	¿Permite deshacer las acciones, e informar el estado?	4		
!!	¿Permite una cómoda navegación dentro del producto y una fácil salida de éste?	4		
!	¿Permite distintos niveles de uso del producto para usuarios con distintos niveles de experiencia?	0		Esta aplicación sirve para todo aquel usuario que necesite utilizarla.
!	¿Se permite al usuario personalizar la interfaz?		NP	La aplicación es estable.
!	¿Se permite al usuario manipular directamente los objetos de la interfaz?	4		
!	¿Se reconoce todas las tareas del software en cualquier momento?	4		
!!	¿Se proporciona información visual de dónde está el usuario, qué está haciendo y qué puede hacer a continuación?	2		
!	¿Existe atajos de teclado bien hechos?	0		No hay atajos de teclados
!	¿Se presenta al usuario la información que sólo necesita?	4		
!	¿Existen diferentes niveles de ayuda?	0		El sistema no presenta ningún tipo de ayuda

Tabla 3.3 Lista de Chequeo para verificar la usabilidad del Producto.

Dándoles respuestas a todas estas interrogantes con un total de 18 respuestas positivas y solo 9 negativas se demuestra que la usabilidad del producto esta regular.

3.3.2 Seguridad.

La calidad no puede existir sin **seguridad**, un producto sin seguridad seria un producto sin calidad. Para evaluar este atributo se formularon las siguientes preguntas:

Nivel	Evaluación	Eval.	NP	Comentario
!!	¿Las reglas de protección de Archivos de datos, las autoridades y códigos de identificación de usuario fueron establecidas por el dueño del sistema o asignado por una autoridad más alta?	2		
!!	¿El Acceso al control de protección está incorporado en el sistema?	0		No hay control de protección.
!	¿Todas las contraseñas del equipo de prueba y desarrolladores fueron borradas?		NP	No se autentifican los usuarios, el software no requiere de autentificación.
!	¿Toda la privacidad, la libertad de información, sensibilidad, y consideraciones de la clasificación fueron identificadas, resueltas, y establecidas?	4		
!	¿Están instrumentados autos chequeos que permitan la protección contra virus?	4		
!	¿Están instrumentados controles que permitan realizar la auditoría informática?	2		

Tabla 3.4 Lista de Chequeo para verificar la seguridad del producto.

Después de analizar y dar respuestas a estas interrogantes el equipo de probadores arribó a un total de 2 respuestas positivas (propiedad disponible) y 3 propiedad que no esta disponible.

3.3.3 Confiabilidad.

La **confiabilidad** es la capacidad del producto para mantener su nivel de desempeño cuando es utilizado bajo condiciones especificadas (durante un determinado período de tiempo). Para evaluar estas características en el producto se ha dado respuesta a las siguientes preguntas:

Nivel	Evaluación	Eval.	NP	Comentario
!	¿Se recupera el software ante fallas?	5		
!	¿La recuperación ante fallas se realiza en el tiempo requerido para la aplicación?	2		
!	¿Se carga correctamente la aplicación?	0		Cuando se conectan muchos usuarios las imágenes no salen.
	¿La aplicación realmente muestra lo que pretende?	5		
	¿Los especialistas pueden establecer análisis a partir de los resultados que ofrece la aplicación?	5		

Tabla 3.5 Lista de Chequeo para verificar la confiabilidad del producto.

Las respuestas a 1 preguntas fueron positivas y a 2 negativa asegurando la confiabilidad que presenta el producto ante cualquier tipo de falla no es la mejor y su capacidad de recuperarse antes esta falla tampoco es la más indicada.

3.3.4 Portabilidad.

La **portabilidad** es la capacidad del producto para ser transferido de un ambiente a otro. El ambiente puede ser organizacional, de hardware o de software. Para evaluar este atributo se basan en las siguientes preguntas:

Nivel	Evaluación	Eval.	NP	Comentario
!	¿Existe independencia del ambiente de hardware?	4		
!	¿Existe independencia del ambiente de software? (sistema operativo, lenguaje de programación)	5		

Tabla 3.6 Lista de Chequeo para la portabilidad del producto.

Las 2 respuestas a estas preguntas resultaron positivas (propiedad disponible).

3.4 Análisis de la Evaluación del producto.

Dado el análisis efectuado basándose en las listas de chequeo, se puede realizar una gráfica con los resultados cuantitativos sobre dichos atributos. Estos resultados serán mostrados en la gráfica que se expondrá a continuación la cual evaluará los atributos mediante una escala, dando a conocer como se encuentra la aplicación según estos atributos.

Este análisis se hizo realizando un promedio entre todos los puntos otorgados a cada pregunta de la lista de chequeo los cuales se encontraban entre los valores del 0 y 5. Se consideró como una respuesta negativa los valores entre 0 y 2, mientras que los valores del 4 al 5 se consideraron positivos.

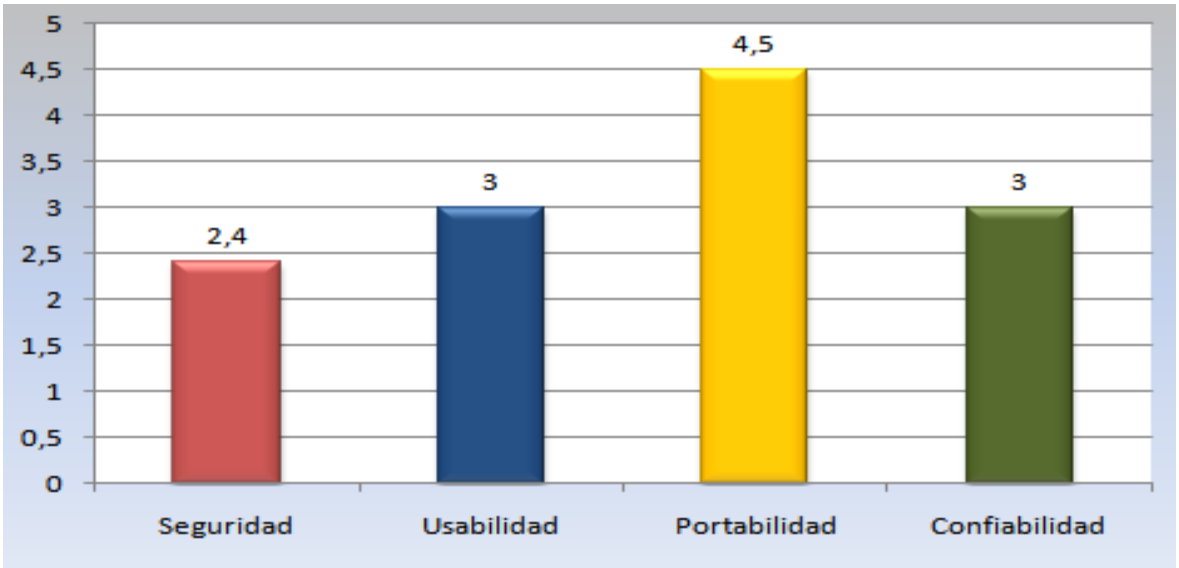


Fig.3.1 Resultados Cuantitativos sobre los Atributos de Calidad.

Como se pudo apreciar después del análisis del producto, quedó demostrado que este no es muy seguro. Otro atributo que no se encuentra muy bien es la confiabilidad pues la aplicación presenta algunos problemas para mostrar las imágenes. Además la aplicación no es lo suficientemente usable ya que para operarla se debe tener conocimientos básicos sobre biología e inglés. Mientras que la portabilidad es bastante buena pues esta aplicación es capaz de instalarse en cualquier ordenador.

Conclusiones.

En este capítulo se llevó a cabo el análisis de los resultados obtenidos luego de aplicar las pruebas en el producto de software, además se identificaron las mejoras obtenidas durante este proceso.

Se mencionan los atributos de calidad del software: la confiabilidad, la seguridad, la portabilidad y la usabilidad, se explica en qué consisten cada unos de ellos así como una evaluación del software en cuanto a estos atributos usando listas de chequeos, la cual arrojo resultados muy satisfactorios.

En general los resultados de las pruebas realizadas al software fueron muy productivos ya que estos arrojaron una gran cantidad de no conformidades lo que en conjunto con la respuesta del equipo de

desarrollo a estas no conformidades influye importantemente en la aceptación final del producto por el cliente, el cual es el principal objetivo de estas pruebas y de todo el proceso de calidad.

CONCLUSIONES

Dada la necesidad que existe de tener softwares confiables y seguros a nivel mundial, es necesario llevar a cabo una estrategia óptima para asegurar la calidad de los mismos. Cada cliente, institución o empresa se ve obligada a comprar softwares precisos y funcionales pues de muchos de estos software depende el desarrollo de sus industrias. No es menos cierto que la industria del software ha crecido inmensamente pero junto con esta también ha aumentado la calidad de los productos dado que mientras menos defectos y más seguro sea un producto más importante se vuelve en el mercado.

La búsqueda de un alto nivel de calidad en el producto es sinónimo de excelencia y esta solo se logra llevando a cabo una táctica perfecta. Para el desarrollo de esta táctica es necesario aplicar las pruebas necesarias al software porque los productos requieren comprobaciones específicas en cada caso y en cada fase de desarrollo.

Al planificarse y aplicarse las pruebas a la aplicación informática siRNA se llegó a las siguientes conclusiones:

- Se alcanzó un amplio conocimiento acerca de los niveles, técnicas, métodos y herramientas de pruebas.

- Se crearon durante este proceso los artefactos: Plan de Prueba, Diseño de los Casos de Pruebas y la Planilla de No Conformidades.

- Se realizó una revisión adecuada de la documentación eliminando de esta la mayor cantidad de errores posible.

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

- Se realizaron las pruebas funcionales empleando el método de Caja Negra.

CONCLUSIONES.

- Se realizaron las pruebas de carga y stress con la herramienta JMeter, las cuales verificaron el rendimiento del software.

- Se logró documentar todos los errores en la Planilla de No Conformidades.

- Se realizó una evaluación del software mediante listas de chequeos.

Se puede plantear que constituye una necesidad para la industria cubana del software, definir y aplicar las pruebas pertinentes al software pues estas son un punto clave en la determinación del nivel de calidad de cualquier aplicación.

Con el estudio realizado y la aplicación de las pruebas a la aplicación informática siRNA, se cumplió el objetivo general de dicho trabajo el cual consistió en desarrollar pruebas de calidad a la aplicación informática de siRNA con vistas a la obtención del mayor número de faltas existentes en la misma, logrando con su eliminación obtener una aplicación con un mínimo de errores y por consiguiente mayor calidad.

RECOMENDACIONES.

Se recomienda que:

- La aplicación siRNA se le realicen las pruebas pertinentes basadas en el método de Caja Blanca ya que esto aseguraría de forma general la calidad del producto.

- Además se deben de desarrollar pruebas durante todo el ciclo de vida del software para minimizar los errores.

- Es necesario que el grupo de desarrolladores elabore los diseños de casos de pruebas.

- Se recomienda a los grupos de calidad de las distintas facultades que incluyan las pruebas de Carga y de Stress en el proceso de liberación pues estas aseguran el buen rendimiento de la aplicación.

REFERENCIAS BIBLIOGRÁFICAS.

[1] Tirado, Naivys Ruiz. Sistema de Analisis de Secuencia mRNA para el diseño de siRNA. V1.1. Ciudad de La Habana : s.n., 2008.

[2]Comunicación, Instituto Nacional de Tecnologias de la. INTECO. [Consultado el: 5 de noviembre de 2008]. Disponible en: http://www.inteco.es/Calidad_del_S.W/Calidad_del_Software_1/Introduccion_a_la_Calidad/.

[3]Pressman, Roger. "Software Engineering: A Practitioner's Approach" (European daptation), McGrawHill. 2000.

[4]Brito, Irina Reyes.. Las pruebas de software, su aplicación a Config. CASE. Ciudad de La Habana. **2003**.

[5] Pressman, Roger. Can Internet-based Aplications Be Engineered. 1998.

[6] Departamento de Ingeniería y Gestión del Software. Conferencia: Flujo de Trabajo de Prueba. 2008.

[7] Pressman, Roger S. Ingeniería del software, un enfoque práctico. s.l. : McGrawHill, quinta edición. 2002

BIBLIOGRAFÍA

-**Cuba a la mano**. Informática. LA INDUSTRIA DEL SOFTWARE Y LA DUALIDAD MONETARIA.[consultado el 15 de diciembre del 2008]. Disponible en: http://cubaalamano.net/sitio/client/article.php?id=5780

-**Caenorhabdtis elegans.** COMO MODELO DE ESTUDIO DEL DESARROLLO. [Consultado el: 27 de noviembre de 2008]. Disponible en: http://bq.unam.mx/mensajebioquimico

- **Tirado, Naivys Ruiz.** Sistema de Analisis de Secuencia mRNA para el diseño de siRNA. V1.1. Ciudad de La Habana : s.n., 2008.

-**Comunicación, Instituto Nacional de Tecnologias de la.** INTECO. [Consultado el: 5 de diciembre del 2008.]. Disponible en: http://www.inteco.es/Calidad_del_S.W/Calidad_del_Software_1/Introduccion_a_la_Calidad/.

-**Departamento de Ingeniería y Gestión del Software.** Conferencia: Flujo de Trabajo de Prueba. **2008**.[Consultado el: 3 diciembre del 2008]

-**Natalia Juristo, Ana M. Moreno, Sira Vegas.** TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 2006 [Consultado el: 6 de diciembre del 2008]

-**Glenford J. Myers.**The art of sotware Testing. Wiley, second edition. 2004

-**Roger S. Pressman.** Ingeniería del software, un enfoque práctico. McGrawHill, quinta edition. 2002

-**Ron Patton.** Software Testing. Sams Publishing. 2005

-**Boris Beizer.** Black-Box Testing. Wiley. 1995

- **S. Pressman, Roger.** Software Engineering: A Practitioner's Approach" (European daptation) McGrawHill. ISBN: 0077096770. 2000

- **S. Pressman, Roger.** R. Can Internetbased Applications Be Engineered? in IEEE Software, September/October IEEE Press, 104110. 1998

BIBLIGRAFÍA

- **Brito, Irina Reyes.** Las pruebas de software, su aplicación a Config. CASE. Ciudad de La Habana: 2003.

- **IEEE.** IEEE Std1995. Metrics, IEEE. 1991

- **Kan, S.H.** Metrics and models in software quality engineering. Addison Wesley.
Reading, Ma., USA, 1995.

-Aseguramiento de la Calidad y Sistema de Calidad. [Consultado el: 27 de mayo de 2009]. Disponible en:

http://74.125.47.132/search?q=cache:9LZjBprU5AgJ:www.gestiopolis.com/canales/gerencial/articulos/27/asesis.htm+aseguramiento+calidad&cd=2&hl=es&ct=clnk&gl=cu

ANEXOS.

Anexo 1.

Listas de Chequeo de *<poner el nombre correspondiente>*

Interno

<Nombre del Proyecto>

<Nombre del producto>

<Versión>

[Documento para la confección de Listas de Chequeo]
*[**Forma de Uso:***

- *en el **Nivel** solo se define cuando existen aspecto de mayor importancia que otros, para ello se usarán signos de exclamación (!), entre más tenga más importante es*
- *en el **Criterio de evaluación** se ubicará el aspecto a evaluar, siempre con una redacción nítida*
- ***Evaluación** es para el caso de la aplicación de la Lista de Chequeo. Dicha evaluación se encontrará en el rango de 0-5*
- ***N.P.** significa No Procede y es para el caso de la aplicación que ese aspecto no sea factible su valoración*
- ***Observaciones** es para cualquier cosa que quiera incluir la persona que aplica dicha lista*
- *La parte de **Métricas, Submétricas y % cumplimiento** es para la aplicación de métricas a las iteraciones realizadas*

Estos comentarios en azul deben borrarse a la hora de entregar finalmente este documento de forma oficial.]

Control de versiones

Fecha	Versión	Descripción	Autor
<dd/mmm/yy>	<x.x>	<detalles>	<nombre>

Reglas de Confidencialidad

Clasificación: <<Clasificación>>

Este documento contiene información propietaria de **ALBET Ingeniería y Sistemas** y/o **"<<Empresa Cliente>>"** , y es emitido confidencialmente para un propósito específico.
El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.
Estas reglas son aplicables a las 3 páginas de este documento.

Fecha:
Responsable que la aplicó:

Nivel	Criterio de evaluación	Evaluación	N.P.	Observaciones	Métrica	Submétrica	%cumplimiento

Anexo 2.

<Nombre del Proyecto>

Módulo <Nombre del Módulo>

Versión del proyecto

Diseño de Casos de Prueba

Nombre del Caso: <Nombre del Caso de Uso>

Versión del CP

Control de versiones:

Fecha	Versión	Descripción	Autor
<dd/mmm/yy>	<x.x>	<detalles>	<nombre>

Tabla de Contenido

Descripción General

[Descripción general del CU]

Condiciones de Ejecución:

[Precondiciones del CU].

1 Secciones a probar en el Caso de Uso:

[Para cada sección los escenarios van a ser flujo básico + los alternativos].

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
[SC 1: Nombre de la sección]	*EC 1.1: Nombre del Escenario.*	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 1.2: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 1.n: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
[SC 2: Nombre de la sección]	*EC 2.1: Nombre del Escenario.*	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 2.2: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 2.n: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
[SC n: Nombre de la sección]	*EC 3.1: Nombre del Escenario.*	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 3.2: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para probar la Funcionalidad que se indicó.*
	EC 3.n: Nombre del Escenario.	*Descripción de la Funcionalidad.*	*Pasos a desarrollar para*

1.1 SC 1 : <Sección #1 a revisar>

Id del escenario	Escenario	Variable 1 *(Nombre de la variable)*	Variable 2 *(Nombre de la variable)*	Variable N *(Nombre de la variable)*	Respuesta del Sistema	Resultado de la Prueba
EC 1	*Nombre del escenario.*	*V*	*V*	*V*	*Se escribe el resultado que se espera al realizar la prueba.*	*Se escribe el resultado que se obtiene al realizar la prueba.*
		V	*V*	*V*		
		V	*V*	*V*		
EC 2	*Nombre del escenario.*	*NA*	*NA*	*NA*	*El sistema emite un mensaje para que llene los campos obligatorios*	*Se escribe el resultado que se obtiene al realizar la prueba.*

[Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.]

2 Registro de defectos y dificultades detectados

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Significativa	No Significativa	Recomendación	Estado NC	Respuesta del Equipo Desarrollo
<Nombre del Elemento>	*< 1>*	*<Descripción de la No Conformidad>*	*<Descripción del Aspecto correspondiente>*	*<Etapa de detección del error>*	*<X>*	*<X>*	*<X>*	*[Se coloca el estado de la NC y la fecha, cada vez que se revise se deja el estado anterior y se coloca el nuevo con la fecha en que*	*[Esta columna se comienza a llenar a partir de la 2da iteración, y es responsabilidad del equipo de desarrollo, quien especifica la conformidad con lo encontrado o no y en caso de no proceder la no*

3 Anexos

3.1 Anexo <1>

Anexo 3.

Nombre del proyecto

Versión 1.x

Plantilla de No Conformidades

Elemento a probar <Nombre del artefacto a probar>

Versión 1.x

Control de versiones:

Fecha	Versión	Descripción	Autor
<00/00/0000>	<1.0>	<Descripción>	<Autor>

Descripción General
[Descripción de Aspectos Generales a tener en cuenta a la hora de realizar el diseño de las pruebas, incidencias en el momento de su desarrollo y otros aspectos relevantes.]

Elementos probados
[Descripción general o lista de los Elementos Probados, y otros aspectos importantes a tener en cuenta a la hora analizar las No Conformidades Detectadas.]

Elementos no probados y causas
[Descripción de Aspectos Generales a tener en cuenta a la hora de realizar el diseño de las pruebas, incidencias en el momento de su desarrollo y otros aspectos relevantes.]

1. Registro de defectos y dificultades detectados

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado NC	Respuesta del Equipo Desarrollo
<Nombre del Elemento>	*< 1>*	*<Descripción de la No Conformidad>*	*<Descripción del Aspecto correspondiente>*	*<Etapa de detección del error>*	*S: Significativa* *NS: No Significativa* *R: Recomendación*	*[Se coloca el estado de la NC y la fecha, cada vez que se revise se deja el estado anterior y se coloca el nuevo con la fecha en que se revisó.] RA: Resuelta PD:Pendiente NP:No Procede*	*[Esta columna se comienza a llenar a partir de la 2da iteración, y es responsabilidad del equipo de desarrollo, quien especifica la conformidad con lo encontrado o no y en caso de no proceder la no conformidad explica por qué.]*

[La NC puede tener solo una de las tres clasificaciones: Significativa, No Significativa o Recomendación]

2. Anexos

Anexo <1>

Anexo 4.

Informe Agregado

Nombre: Informe Agregado

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | **Navegar...** | **Escribir en Log Sólo Errores** | **Configurar**

Label	# Muestras	Media	Mediana	Linea de ...	Mín	Máx	% Error	Rendimiento	Kb/sec
/biosiRNA-Design_v2.0/in...	20	67	62	110	15	141	0,00%	15,2/sec	72262,0
/biosiRNA-Design_v2.0/im...	20	68	63	125	31	157	0,00%	16,4/sec	167245,9
/biosiRNA-Design_v2.0/im...	20	70	78	110	15	172	0,00%	15,4/sec	74675,9
/biosiRNA-Design_v2.0/im...	20	74	63	203	15	219	0,00%	14,4/sec	12207,0
/biosiRNA-Design_v2.0/im...	20	79	78	125	15	156	0,00%	15,0/sec	203310,8
/biosiRNA-Design_v2.0/im...	20	71	62	141	31	203	0,00%	15,4/sec	2390,1
/biosiRNA-Design_v2.0/im...	20	64	63	94	15	187	0,00%	17,1/sec	2527,8
/biosiRNA-Design_v2.0/db...	111	11381	10031	26531	484	30641	0,00%	56,4/min	12471,9
/biosiRNA-Design_v2.0/im...	41	147	63	281	15	1094	100,00%	31,5/min	837,6
/biosiRNA-Design_v2.0/im...	61	217	63	516	15	2656	100,00%	46,8/min	1239,2
/biosiRNA-Design_v2.0/im...	30	461	297	1156	31	1641	100,00%	28,3/min	766,3
/biosiRNA-Design_v2.0/m...	41	8349	4297	18906	297	19688	0,00%	30,7/min	101069,2
/biosiRNA-Design_v2.0/wi...	20	1272	1047	3000	94	3062	0,00%	30,1/min	17625,8
/biosiRNA-Design_v2.0/wi...	20	634	563	1750	47	1891	0,00%	31,5/min	39928,6
/biosiRNA-Design_v2.0/wi...	20	668	562	1735	157	1796	0,00%	31,1/min	263179,3
/biosiRNA-Design_v2.0/wi...	20	328	250	813	15	1593	0,00%	31,1/min	953,3
/biosiRNA-Design_v2.0/im...	20	461	328	765	31	3172	100,00%	30,2/min	817,7
TOTAL	524	3274	250	11812	15	30641	29,01%	4,4/sec	193319,8

Anexo 5.

Informe Agregado

Nombre: Informe Agregado

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Escribir en Log Sólo Errores | Configurar

Label	# Muestras	Media	Mediana	Linea de 9...	Mín	Máx	% Error	Rendimiento	Kb/sec
/biosiRNA-Design_v2.0/index.html	70	707	484	1657	47	2312	0,00%	11,8/sec	56085,1
/biosiRNA-Design_v2.0/images/resources.js	70	1253	1141	2407	78	2546	0,00%	9,4/sec	95422,2
/biosiRNA-Design_v2.0/images/default.css	70	1145	1437	2281	125	2578	0,00%	8,5/sec	40973,8
/biosiRNA-Design_v2.0/images/img1.gif	70	1107	1375	1578	250	2406	0,00%	7,2/sec	6104,8
/biosiRNA-Design_v2.0/images/img4.jpg	70	1091	1312	1531	375	2500	0,00%	6,8/sec	91433,8
/biosiRNA-Design_v2.0/images/img2.gif	70	943	985	1390	438	1625	0,00%	6,4/sec	991,9
/biosiRNA-Design_v2.0/images/img3.gif	70	906	907	1406	391	2593	0,00%	4,8/sec	713,0
/biosiRNA-Design_v2.0/dbmain.html	257	19866	19921	34516	453	67671	0,00%	1,1/sec	12729,3
/biosiRNA-Design_v2.0/imageFindGene	92	5437	4204	12406	31	39734	100,00%	23,8/min	632,8
/biosiRNA-Design_v2.0/imageFindGeneMIRNA	162	5969	5656	10656	203	27281	100,00%	41,7/min	1104,1
/biosiRNA-Design_v2.0/imageFindTranscript	92	7284	6203	14547	203	18078	100,00%	27,3/min	736,7
/biosiRNA-Design_v2.0/main.html	73	24999	26266	50344	625	75453	0,00%	19,6/min	76251,7
/biosiRNA-Design_v2.0/window/adapter/ext/ext-...	49	9197	8594	14594	1109	16156	0,00%	20,2/min	11853,4
/biosiRNA-Design_v2.0/window/resources/css/...	47	9841	8187	18047	219	26859	0,00%	17,7/min	22389,9
/biosiRNA-Design_v2.0/window/ext-all.js	20	8837	8172	15328	4219	15609	0,00%	8,3/min	70101,9
/biosiRNA-Design_v2.0/window/hello.js	20	7739	7578	15047	1656	15641	0,00%	9,1/min	279,5
/biosiRNA-Design_v2.0/imageSequenceDesig...	20	11121	10531	22109	3063	29344	100,00%	8,8/min	236,8
TOTAL	1322	8347	4078	23984	31	75453	27,69%	5,0/sec	144991,9

GLOSARIO DE TÉRMINOS.

1 AVL: Análisis de Valor Limite.

2 CIGB: Centro de Ingeniería Genética y Biotecnología.

3 RNA: Ácido ribonucleico es un ácido nucleico.

4 siRNA: siglas en inglés (small interfering RNA); siglas en español (ARN pequeño de interferencia oARN de silenciamiento).

5 UCI: Universidad de las Ciencias Informáticas.