UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Evaluación estática del código para la capa de presentación de los proyectos desarrollados bajo tecnología Sauxe en el CEIGE.

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

AUTOR: Yaima Hernández Leonard

TUTOR: Ing. Iliannis Pupo Leyva

La Habana, junio 2012.

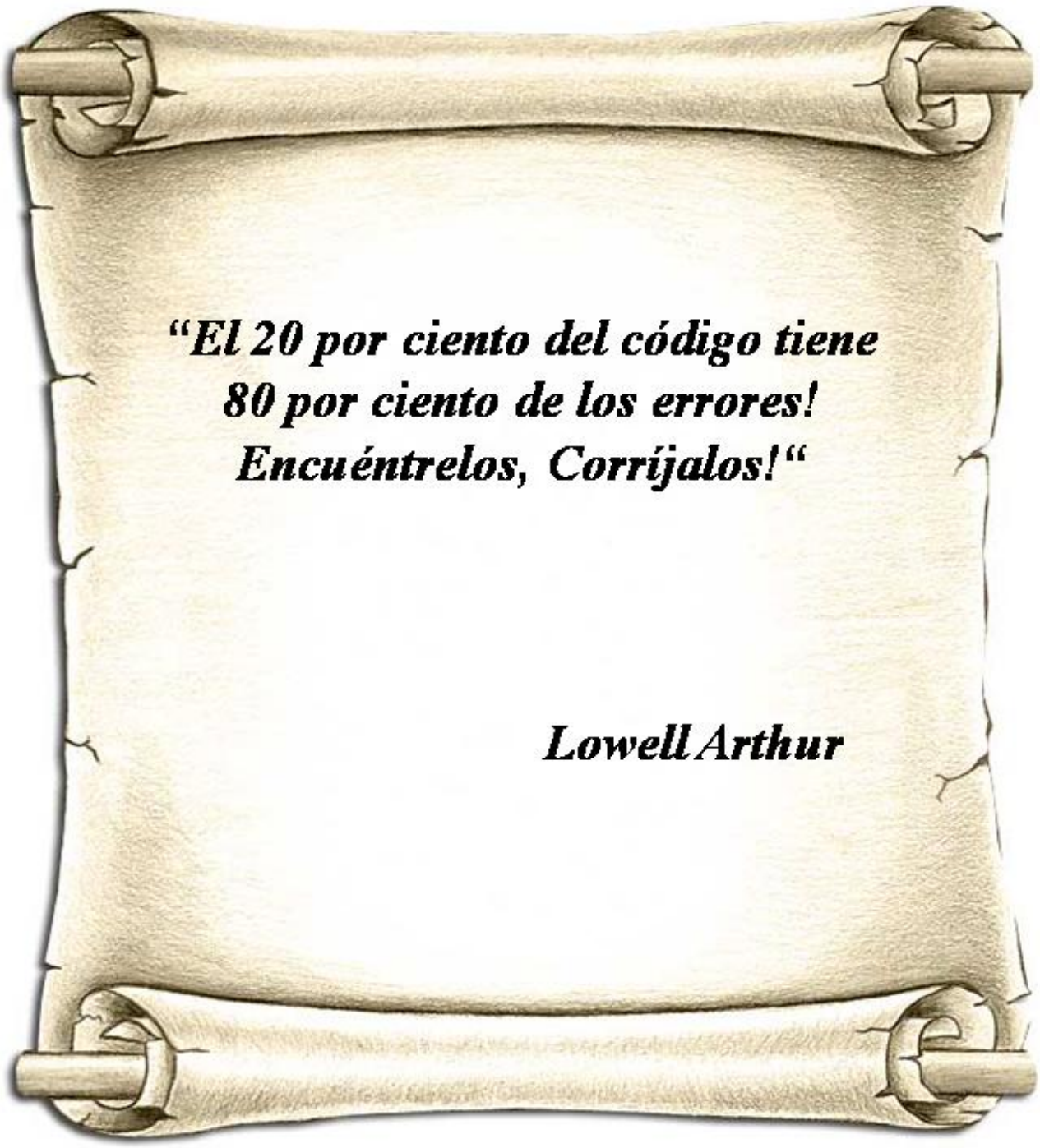
"Año 54 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Yaima Hernández Leonard	Ing. Iliannis Pupo Leyva
Autor	Tutor


"El 20 por ciento del código tiene
80 por ciento de los errores!
Encuéntrelos, Corrijalos!"
Lowell Arthur

DEDICATORIA

Dedico todo el esfuerzo, lágrimas y trabajo que empleé en la realización de este trabajo, que me convirtió en ingeniera en ciencias informática:

A mi madre, por ser la razón de mí existir.

A mi hermana por ser para mí lo más importante del mundo.

A mi padre por ser fuerte conmigo, cuando creí que no podría lograrlo.

A mi abuelo por quererme tanto, y estar siempre pendiente de mí.

A mis abuelas y abuelo que no están hoy en este mundo, pero sí en mi corazón por toda la eternidad.

AGRADECIMIENTOS

A mi Patri por ayudarme tanto, y estar siempre disponible para mí, porque la quiero mucho y por ser la que más tuvo que aguantar mis pesadeces a todos los niveles.

A Fide y a Lía por quererme tanto y preocuparse siempre por mí.

A Yangzet por considerarme una hermana en su vida y siempre acordarse de mí.

A Zaida porque a pesar de sus días en los que se portaba como una odiosa, logró que le tomara mucho cariño.

A Arlethy porque a pesar de ser bien resabiosa es muy buena persona, aunque ella diga lo contrario.

A Yake por estar ahí para entretenerme a diario y tener a quien criticar y molestar.

A Michael por darme amor en estos últimos meses de universidad y soportar mis niñerías.

A Marcia por darme siempre el regaño cuando no estaba haciendo algo productivo.

A Susana por enseñarme a llegar a la Virgen del Camino en más de una ocasión.

A Arnolis por estar disponible cada vez que lo molesté y necesité de su ayuda.

A JC porque aunque hoy no es lo mismo que ayer, hizo mucho por mí cuando lo necesité.

A Diosmani porque sin él no hubiera podido empezar lo que terminé.

A Yunet por ayudarme como lo hizo y comportarse como la mejor oponente del mundo.

A Iliannis por ayudarme en la realización de este trabajo.

Agradezco a todos los que se convirtieron en mis seres más cercanos y queridos en los 5 años que estuve en la Universidad de las Ciencias Informáticas, soportando y aguantando todas y cada una de mis pesadeces.

RESUMEN

El presente trabajo tiene como propósito fundamental evaluar mediante pruebas de caja blanca estática el código fuente de la capa de presentación de los sistemas de software que se encuentren en desarrollo bajo la tecnología Sauxe. Este tipo de pruebas tiene como tarea principal, encontrar errores en el código fuente que puedan afectar el rendimiento, la estandarización y el correcto funcionamiento de estos productos informáticos, mientras estos se encuentren en la fase de desarrollo de software.

Estas pruebas se realizaron de dos formas, de manera automática, mediante el uso de una herramienta que analiza el código fuente JavaScript, en busca de errores y de forma manual, mediante la intervención de una persona que revisa que el código fuente implementado en JavaScript está estandarizado.

Se aplicó este tipo de pruebas a 5 componentes que se encuentran en desarrollo, pertenecientes al subsistema Capital Humano del CEIGE. Una vez probados los componentes, se utilizaron métricas para medir la mantenibilidad de los mismos, asegurando de cierta manera que el código desarrollado es fácil de entender, que presenta una correcta estructura y no requiere de mucho esfuerzo, para adaptarse a nuevos cambios e incorporar nuevos requerimientos, una vez que ha sido desarrollado.

PALABRAS CLAVES: pruebas de caja blanca estática, código, errores, análisis estático, mantenibilidad.

TABLA DE CONTENIDO

INTRODUCCIÓN

Los nuevos avances tecnológicos en la rama de la informática han hecho que esta ciencia sea cada día más necesaria, útil y dinámica por lo que las grandes empresas desarrolladoras de software se empeñan en informatizar todo lo que les sea posible con su ingenio.

En Cuba se crea la Universidad de las Ciencias Informáticas (UCI) con el objetivo de informatizar el país y desarrollar la industria del Software para contribuir al desarrollo económico del mismo. La UCI es una universidad productiva, cuya misión es desarrollar software y servicios informáticos a partir de la vinculación estudio-trabajo como modelo de formación. La Producción de Software y Servicios Informáticos se basa en la integración de los procesos de formación, investigación y producción en torno a una temática para convertirla en una rama productiva. [1]

En Cuba se ha identificado la necesidad de introducir un sistema informático que permita centralizar la gestión de las entidades presupuestadas. Las limitaciones técnico-económicas para adaptar sistemas de este tipo internacionalmente conocidos a las condiciones cubanas, impulsaron la idea de desarrollar un Sistema Integral de Gestión con particularidades de la economía cubana., de esta manera y siguiendo los principios de independencia tecnológica surge Cedrux. Con el surgimiento de Cedrux se crea el marco de trabajo Sauxe el cual integra ExtJs y JavaScript para el desarrollo de la capa de presentación y Zend Framework y PHP para la capa de negocio. El presente trabajo se centra en la evaluación estática del código para la capa de presentación de los proyectos desarrollados bajo la tecnología Sauxe en el CEIGE.

Las pruebas de caja blanca se basan en un examen minucioso de los detalles procedimentales, logrando examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado. [2] Estas pruebas tienen entre sus clasificaciones las pruebas de caja blanca estática. El análisis estático del código es el proceso de evaluar el software sin ejecutarlo, con esto se quiere básicamente ganar en facilidad de mantenimiento y de desarrollo. [3]

El CEIGE consta de diferentes departamentos que desarrollan productos informáticos y estos productos son sometidos a disímiles pruebas. El centro desea mejorar el proceso de pruebas incluyendo las pruebas de caja blanca estática durante el desarrollo del software, principalmente se quiere probar la propuesta a

realizar con el subsistema Capital Humano, debido a que posee actualmente soluciones informáticas en desarrollo, definiendo para ello una herramienta que sea capaz de evaluar la estructura del código fuente.

Podría decirse que al no llevarse a cabo este método de prueba, las consecuencias son: errores en el software que se desarrolla, los mismos poseen una excesiva complejidad, el código fuente de estos sistemas no está estandarizado y un software con errores puede causar problema en la seguridad, pues se hace vulnerable a posibles inyecciones SQL, provocando además que se debilite el atributo de mantenibilidad al no tomarle medidas cuantitativas al código fuente, que verifiquen en cierta forma que se está desarrollando un software entendible, poco complejo y adaptable a nuevas modificaciones sin emplear demasiado esfuerzo en el cambio.

No llevar a cabo este método de prueba puede traer como resultado que el software que se desarrolla arrastre errores que pudieran aparecer cuando el sistema está en uso por el cliente, además que el proceso de mantenimiento del software se lleve el doble de esfuerzos y recursos que cuando se desarrolló el mismo, al no haber construido un software en cierta medida mantenible.

Estas pruebas ayudan a determinar la razón por la cual el código no alcanza su mayor grado de eficiencia, lo que pudiese traducirse en demora en el tiempo de respuesta del software al cliente, o la posibilidad de que pueda colapsar bajo algún tipo de situación.

Actualmente el centro no puede garantizar al cien por ciento que sus productos informáticos tengan la mayor mantenibilidad, ya que el grupo de calidad del mismo, encargado de la revisión y liberación interna de los productos de software que se realizan en el CEIGE bajo la tecnología Sauxe, solo realiza pruebas de caja negra a estos productos informáticos. Una vez que estos son revisados por el grupo de calidad interna del CEIGE, ingresan en Calisoft, donde se examinan nuevamente antes de ser liberados, para finalmente ser usados por el cliente, pero tampoco en Calisoft son sometidos a este tipo de pruebas, puesto a que las mismas se realizan cuando el software aún no está terminado. Cuando el producto de software llega a este centro, no es posible realizar este tipo de pruebas, pues sería demasiado trabajoso, por la cantidad de líneas de código que posee un sistema de software de esta índole.

Luego de haber fundamentado los aspectos anteriores se propone el siguiente **problema a resolver:** en los proyectos del CEIGE desarrollados bajo la tecnología Sauxe las dificultades existentes con el rendimiento, la complejidad y la estandarización del código afectan la mantenibilidad de los componentes.

Se define como **objeto de estudio**: pruebas de software de caja blanca en el CEIGE y el **campo de acción** se enmarca en pruebas de software de caja blanca estática en la capa de presentación de los sistemas desarrollados bajo la tecnología Sauxe en el CEIGE.

El **objetivo general** es evaluar el código mediante pruebas de caja blanca estática de la capa de presentación de los proyectos desarrollados bajo tecnología Sauxe, que contribuya a elevar la mantenibilidad de los componentes.

Objetivos Específicos:

Identificar tipos de pruebas, técnicas y herramientas para la realización de pruebas de caja blanca estática a nivel internacional y nacional.

Describir tipos de pruebas, actividades, artefactos y herramientas a utilizar para aplicar pruebas de caja blanca estática a la capa de presentación de los componentes desarrollados bajo la tecnología Sauxe.

Validar la solución propuesta a través de su aplicación en uno de los productos del CEIGE desarrollado bajo la tecnología Sauxe.

Para dar cumplimiento a los objetivos trazados se emplearon los siguientes métodos:

Analítico – Sintético: Para llegar a conclusiones en la investigación a partir de la información que se procese y precisar características necesarias para la realización del trabajo propuesto.

Método – Sistémico: Para definir los elementos de cómo realizar la evaluación de la mantenibilidad al código fuente de la capa de presentación de los componentes en desarrollo bajo la tecnología Sauxe mediante pruebas de caja blanca estática.

Encuesta: Para obtener información que fundamente actividades a definir en la solución.

El siguiente trabajo de diploma tiene una estructura de tres capítulos:

Capítulo 1: Fundamentación Teórica. Estado del arte. Presenta alguno de los conceptos más importantes sobre las pruebas de software y la calidad del software, se hace énfasis en las pruebas de caja blanca y se profundiza en las pruebas de caja blanca estática, se estudian también las formas en que se pueden realizar estas pruebas y se hace un estudio de herramientas que pueden ser utilizadas para llevar a cabo este tipo de pruebas.

Capítulo 2: Procedimiento propuesto. Se propone un procedimiento para llevar a cabo las pruebas de caja blanca estática, haciendo uso de una herramienta que automatiza parte de este proceso de pruebas. Además va a intervenir de forma manual una persona que revise el cumplimiento del estándar de codificación, garantizando que se estén desarrollando sistemas informáticos estandarizados.

Capítulo 3: Evaluación del Procedimiento Propuesto. Se valida la solución propuesta mediante su aplicación en uno de los subsistemas que se encuentran en desarrollo bajo la tecnología Sauxe, el mismo perteneciente al CEIGE.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se realiza la fundamentación teórica de la investigación, una breve descripción de las pruebas de software y definiciones de los términos calidad de software, aseguramiento de la calidad, control de la calidad, así como un análisis de conceptos y elementos como pruebas de caja blanca, pruebas de caja blanca estática además de abordarse sobre los estándares de codificación y las ventajas de los mismos.

1.2 Calidad de software

La definición de la calidad del software según la IEEE, Std. 610-1990, es “el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”. [5]

Según Pressman es la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente. [6]

La anterior definición sirve para hacer hincapié en tres puntos importantes:

1. Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.

2. Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería del software. Si no se siguen esos criterios, casi siempre habrá falta de calidad.

3. Existe un conjunto de requisitos implícitos que a menudo no se mencionan (por ejemplo: el deseo por facilitar el uso y un buen mantenimiento). Si el software se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del software queda en entredicho.

Según lo antes planteado podrá decirse que cuando se habla de calidad de software se entenderá por condiciones funcionales que cumple el software para satisfacer al cliente, pero también depende del correcto uso de estándares de codificación y buenas prácticas de programación que den lugar a un software entendible, bien estructurado y sin errores de codificación.

1.2.1 Gestión de la calidad de Software

La Gestión de la Calidad de Software es según ISO 9000 el conjunto de actividades de la función general de la dirección que determina la calidad, los objetivos y las responsabilidades y se implanta por medios tales como la planificación de la calidad, el control de la calidad, el aseguramiento (garantía) de la calidad y la mejora de la calidad, en el marco del sistema de calidad. [7]

Las actividades de gestión de la calidad se verán por tanto encaminadas a garantizar que se esté construyendo un software de calidad, a través del seguimiento, control y mejora del mismo.

1.2.2 Aseguramiento de la calidad del software

El aseguramiento de la calidad es un modelo planificado y sistemático para proporcionar una adecuada confianza en que un producto es conforme con los requerimientos técnicos establecidos.

"Conjunto de actividades sistemáticas necesarias para aportar la confianza en que el software satisfaga los requisitos dados de calidad". La IEEE1 lo define como "conjunto de actividades para evaluar el proceso mediante el cual se desarrolla el software" [8]. Entre las principales técnicas para el aseguramiento de la calidad, según el estándar IEEE 1074, están las siguientes: - Métricas del software para controlar el proyecto. - Verificación y validación del software (incluyendo pruebas y revisiones) en todas las fases del ciclo de vida. - Gestión de la configuración del software. [9]

1.2.3 Control de la calidad de software

El control de calidad es una serie de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso del software, para asegurar que cada producto cumpla con los requisitos que le han sido asignados.

“Técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en dos objetivos fundamentales: mantener bajo control un proceso y eliminar las causas de errores en las diferentes fases del ciclo de vida”. [8]

1.3 Pruebas de software

Una de las actividades críticas en el aseguramiento de la calidad es la prueba, la cual para Myers [22], es el "proceso de ejecutar un programa con el fin de encontrar errores. El nombre "prueba", además de la actividad de probar, se puede utilizar para designar "un conjunto de casos y procedimientos de prueba" [25].

"La prueba es el flujo de trabajo fundamental cuyo propósito general es comprobar el resultado de la implementación mediante las pruebas de cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema que van a ser entregadas a terceras partes". [23]

El responsable de la prueba debe entender el software e intentar desarrollar una imagen mental de cómo podrá fallar el mismo.

Se trata de diseñar pruebas que tengan una alta probabilidad de encontrar el mayor número de errores, con la mínima cantidad de esfuerzo y de tiempo.

1.3.1 Objetivos de las pruebas de software

El objetivo de las pruebas del software es detectar el mayor número posible de errores.

Glen Myers establece varias normas que pueden servir como objetivos de la prueba del software [22]:

Un buen caso de prueba es aquel que tiene una probabilidad alta de mostrar un error no descubierto hasta entonces.

Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Las pruebas están diseñadas para encontrar errores y minimizar los mismos, no garantizan un software libre de errores. Mientras que Bill Hetzel [24], en *"The Complete Guide To Software Testing"*, planteó como objetivos de las pruebas de software:

- ✓ Planificar las pruebas necesarias en cada iteración.
- ✓ Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba que especifican cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.

- ✓ Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan errores son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los errores importantes puedan ser arreglados.

Un producto de software se puede probar de dos formas, mediante pruebas de caja negra y pruebas de caja blanca.

1.3.2 Pruebas de caja negra

Las pruebas de caja negra se centran en los requisitos funcionales, permitiendo al ingeniero del software derivar conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. [2]

1.3.3 Pruebas de Caja Blanca

Durante el proceso de desarrollo de software se hace necesario no solo la comprobación de la funcionalidad de este, sino también la eficiencia de su código fuente. Con el objetivo de garantizar esto se realizan las pruebas de caja blanca, las cuales se basan en un examen minucioso de los detalles procedimentales, logrando examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado. [2]

1.3.3.1 Antecedentes

En la UCI se han elaborado trabajos de diplomas vinculados con las pruebas de caja blanca. Para aplicarlas a módulos específicos del proyecto productivo al que pertenece, quien elabora la solución. Estos trabajos se centran en las pruebas de caja blanca aplicando la técnica del camino básico, sin entrar a evaluar de manera estática el código fuente que se desarrolla, en busca de errores de codificación mediante pruebas de caja blanca estática. A continuación se citan las tesis que se desarrollaron con este fin.

Estrategia para la aplicación de Pruebas de Caja Blanca y Pruebas de Caja Negra al proyecto Registros y Notarías. Trabajo de diploma desarrollado por estudiantes de la Facultad #3. [2]

Propone una estrategia para aplicar Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías. La estrategia para las Pruebas de Caja Blanca propone a RUP como metodología pues tiene un

proceso de pruebas bien definido, así como un conjunto de artefactos. Expone una serie de roles que son responsables de diseñar, aplicar y dar seguimiento a todas las pruebas a realizar. La herramienta propuesta es la NUnit que es un *framework* que se puede ejecutar desde la consola o a través de una interfaz gráfica y se puede integrar con el Visual Studio en cualquiera de sus versiones.

Procedimiento general de Pruebas de Caja Blanca aplicando la técnica del Camino Básico. [16]

Trabajo de diploma desarrollado por estudiantes de la Facultad #7.

Propone un procedimiento para aplicar Pruebas de Caja Blanca aplicando la técnica del Camino Básico. El procedimiento presenta un conjunto de actividades a llevar a cabo durante el proceso de realización de las pruebas y las diferentes plantillas para registrar los resultados de estas. Pero no propone una herramienta para la automatización de pruebas.

Implementación de herramientas para viabilizar el proceso de Pruebas de Caja Blanca. [17]

Trabajo de diploma desarrollado por estudiantes de la Facultad #7.

Se encarga de la implementación de una herramienta para realizar Pruebas de Caja Blanca usando la técnica de Camino Básico. La herramienta es una aplicación de escritorio que utilizando un analizador léxico, lee código fuente y de ahí, aplicando la técnica del Camino Básico obtiene casos de prueba. Esta herramienta solo lee el código C#.

Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS. [18]

Propone un procedimiento para aplicar Pruebas de Caja Blanca aplicando la técnica del Camino Básico. El procedimiento presenta un conjunto de actividades a llevar a cabo durante el proceso de realización de las pruebas y las diferentes plantillas para registrar los resultados de estas. Se especifica la actividad que realizará cada trabajador y los artefactos de entrada y salida. La herramienta de automatización estudiada para los desarrollos en PHP no se ajusta al desarrollo del software en el Área Temática. Debido a esto se propone una herramienta para el lenguaje Java, pues los nuevos desarrollos serán implementados en este lenguaje.

Estos trabajos de diplomas no resuelven lo que se quiere realizar con la actual investigación, pues la nueva solución está enfocada a resolver los errores de código que pudieran darse por el uso incorrecto

del estándar de codificación, por errores de implementación que pudieran cometer los desarrolladores o por atentar contra las buenas prácticas de programación, errores que serán encontrados mediante el uso de una herramienta de análisis estático, que verifique la estructura del código a medida que se vaya realizando la fase de implementación y también serán encontrados errores de forma manual, realizando revisiones visuales que verifican que el código desarrollado cumple con el estándar de codificación definido. Por lo que se evidencia la necesidad de realizar una nueva propuesta de solución.

En la actualidad las pruebas de caja blanca han tomado un gran auge, tanto que sin esta prueba un sistema desarrollado carecería de garantías y credibilidad en sus resultados. Estas pruebas constituyen un método mediante el cual el ingeniero del software puede obtener casos de prueba que garanticen: [10]

- ✓ Ejercitar por lo menos una vez todos los caminos independientes de cada módulo.
- ✓ Ejercitar todas las decisiones lógicas en sus vertientes verdadera y falsa.
- ✓ Comprobar los ciclos en sus límites y con sus límites operacionales.
- ✓ Ejercitar las estructuras de datos internas para asegurar su validez.

A este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Este método se centra en cómo diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así, la lógica interna del programa. [10]

Tanto para la realización de verificaciones como de validaciones se pueden utilizar distintos tipos de técnicas. En general, estas técnicas se agrupan en dos categorías:

Técnicas de Evaluación Estática: Buscan errores sobre el sistema en reposo. Esto es, estudian los distintos modelos que componen el sistema software buscando posibles errores en los mismos. Así pues, estas técnicas se pueden aplicar, tanto a requisitos como a modelos de análisis, diseño y código. [10]

Técnicas de Evaluación Dinámicas: Generan entradas al sistema con el objetivo de detectar fallos, cuando el sistema ejecuta dichas entradas. Los fallos se observan cuando se detectan incongruencias entre la salida esperada y la salida real. La aplicación de técnicas dinámicas es también conocida como pruebas de software o *testing* y se aplican generalmente sobre código puesto que es, hoy por hoy, el único producto ejecutable del desarrollo. [10]

1.3.3.2 Pruebas de caja blanca estática

Pruebas de caja blanca estática: es el proceso que cuidadosamente y metódicamente revisa el diseño del software, la arquitectura o el código para encontrar errores sin necesidad de ejecutar el código. Esto algunas veces se refiere a un análisis estructural. [10]

Es necesario evaluar el sistema software a medida que se va avanzando en el proceso de desarrollo de dicho sistema. De esta forma se intenta que la detección de errores se haga lo antes posible y tenga menor impacto en el tiempo y esfuerzo de desarrollo.

Las técnicas de evaluación estática se aplican en el mismo orden en que se van generando los distintos productos del desarrollo. Esto no es más que: la evaluación estática acompaña a las actividades de desarrollo, a diferencia de la evaluación dinámica que únicamente puede dar comienzo cuando finaliza la actividad de implementación. [10]

La evaluación estática (conocida con el nombre genérico de Revisiones) se realiza en paralelo al proceso de desarrollo de software, constando de una actividad de evaluación emparejada con cada actividad de implementación. Es decir, la actividad de Definición de Requisitos de Usuario va acompañada de una actividad de Revisión de Requisitos de Usuario, la actividad de Definición de Requisitos Software va emparejada con su correspondiente actividad de revisión y así, sucesivamente.[10]

Las actividades de revisión marcan el punto de decisión para el paso a la siguiente actividad de desarrollo. Es decir, la actividad de requisitos interactúa con la actividad de revisión de requisitos en un bucle de mejora iterativa hasta el momento en que la calidad de los requisitos permite abordar la subsiguiente fase de desarrollo. Lo mismo ocurre con el diseño arquitectónico: sufrirá una mejora iterativa hasta que su nivel de calidad permita pasar al diseño detallado y así, sucesivamente. Nótese que esto también ocurre en la fase de implementación. La actividad siguiente a la de implementación es la fase de pruebas unitarias. No obstante, antes de pasar a ella, los programas deberán evaluarse estáticamente. [10]

En otras palabras, las actividades de revisión acompañan las actividades del modelo de desarrollo de software que guía el proyecto. En los modelos de desarrollo de software tradicionales, las actividades de evaluación tanto estáticas como dinámicas tienen una inmersión clara dentro de cada una de las fases del proceso. Es decir, se puede diferenciar claramente donde se introducen las actividades de revisión pues cada fase de desarrollo está claramente diferenciada. [10]

La necesidad de hacer liberaciones de código a intervalos cortos de tiempo (totalmente probadas) permite involucrar la evaluación en cada una de las actividades diarias que acompañan el proceso de desarrollo. [10]

Nótese cómo la evaluación de los productos software mediante revisiones permite contar con una estimación temprana de la calidad con que se está llevando a cabo el desarrollo. Esto es así porque las revisiones encuentran errores, pero la cantidad de errores encontrados en un producto dan una idea de los que aún pueden quedar, así como de la calidad del trabajo de desarrollo de dicho producto. La experiencia parece indicar que donde hay un error hay otros. Es decir, la probabilidad de descubrir nuevos errores en una parte del software es proporcional al número de errores ya descubiertos. [10]

En general, las actividades de evaluación estática constituyen los puntos de control o revisión utilizados por los gestores de proyectos y las organizaciones para evaluar tanto la calidad de los productos como el progreso del proyecto. Es decir, las actividades de revisión son una herramienta de control para el producto software.

La experiencia demuestra que entre el 30% y el 70% de los errores, de diseño y código son detectados por las técnicas estáticas. Esto supone un gran ahorro, pues la corrección es más fácil y menos costosa durante la evaluación estática que durante la dinámica. Nótese que cuando en la evaluación dinámica del sistema se detecta un fallo en un programa, lo que se detecta es el fallo, no el error que lo provoca. Es decir, tras la detección del fallo, se requiere una labor de localización en el programa del error que provocó el fallo. Sin embargo, con las técnicas estáticas, lo que se detecta son directamente errores. Por tanto, una vez detectado, se puede pasar a la fase de corrección. Es decir, desaparece la tarea de localización del error. Esto significa, que las técnicas estáticas son menos trabajosas por encontrar el error y poder pasar a su corrección en el momento que se encuentra, que las técnicas dinámicas. [10]

En el caso concreto de las revisiones de código, estas permiten localizar secciones críticas, lo que permitirá dedicar un mayor esfuerzo a ellas en la fase de pruebas.

Los errores que se buscan al evaluar estáticamente los productos de software son:

Para el Código Fuente:

Corrección: El código no debe contener errores. Los errores de corrección se pueden referir a aspectos como: Errores de “escritura”, es decir, lo que habitualmente se conoce por “programa que no funciona”.

Por ejemplo, bucles infinitos, variable definida de un tipo pero utilizada de otro, contador que se sale de las dimensiones de un arreglo, etc. [10]

1.4 Análisis Estático del código

Es una técnica que se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo. La idea es que, en base a ese código fuente, se pueda obtener información que permita mejorar la base de código manteniendo la semántica original. [3]

Existen herramientas que hacen varios tipos de análisis, algunos de los cuales pueden ser divididos en diferentes categorías, dependiendo del valor que devuelva el análisis hecho por la herramienta. [4]

Las categorías más comúnmente utilizadas son: [4]

Revisión de Código

Este tipo de herramientas generalmente realizan análisis automatizado mediante la carga y "parseo" del código, en búsqueda de patrones de programación particulares que violen un conjunto de reglas establecidas.

Dependencia de código

Examina la relación entre el código fuente y sus dependencias para crear un mapa de toda la arquitectura de la aplicación.

Complejidad de código

Analizan y comparan la complejidad del código de los programas (métricas) para determinar que parte del sistema es innecesariamente complejo.

Tendencias

Las herramientas de análisis de tendencias, no usan los artefactos de código de forma directa, de lo que se encargan es del estudio de las mejoras o degradaciones en la calidad del código, basándose en los resultados de los análisis de las otras categorías anteriormente mencionados.

1.4.1 Formas de realizar análisis estático del código

Se puede decir que existen dos formas de realizar la evaluación estática del código. Por un lado está el análisis **automático** que realiza un programa de ordenador sobre el código y por otro está el análisis **manual** que realiza una persona.

El análisis realizado por un programa de ordenador, o análisis automático, reduce la complejidad que supone detectar problemas en la base del código ya que busca los errores utilizando unas reglas que tiene predefinidas. [3]

El análisis realizado por una persona, o análisis manual, se centra en apartados propios de la aplicación en concreto como, por ejemplo, determinar si las librerías que está utilizando el programa se están utilizando debidamente o si la arquitectura del software es la correcta. [3]

1.4.2 Beneficios del análisis estático de código automatizado

Algunos beneficios ya fueron mencionados "entre líneas", sin embargo, es mucho más básico como dos razones simples: **ahorrar tiempo para ahorrar dinero.** [3]

Uno de los aspectos del ahorro de tiempo es muy obvio, realizar análisis estático de código mediante una herramienta se toma mucho menos tiempo en obtener código de mejor calidad, que cuando se realiza frente a la realización de la inspección por método manual.

El análisis automático se centra únicamente en facetas de más bajo nivel como la sintaxis y la semántica del código, funcionando este análisis en cualquier tipo de aplicación.

El análisis automático, puede ser realizado con una mayor periodicidad ya que no requiere de intervención humana y, lo que es mejor, puede ser programado y repetido tantas veces como sea necesario. Además obra con objetividad, siempre va a devolver la misma respuesta ante el mismo código fuente de entrada. [3]

Como se puede ver, su misión es la de servir de ayuda en los desarrollos, ayudando a detectar errores y ofreciendo posibles soluciones a los mismos.

1.5 Herramientas para realizar análisis estático al código JavaScript

Existen herramientas que permiten analizar de manera estática el código desarrollado en diferentes lenguajes de programación, por lo que es bien recomendable usar dichas herramientas mientras se está desarrollando el producto de software. Es por ello que para el presente trabajo de diploma se pretende seleccionar una herramienta que verifique la correcta implementación del lenguaje JavaScript de los componentes que están siendo desarrollados bajo la tecnología Sauxe del proyecto ERP en el CEIGE.

Existen herramientas que dan la posibilidad de analizar el código de forma estática en el lenguaje JavaScript en busca de errores, con el objetivo de aumentar la calidad del código, entre ellas se encuentran JSLint, JSHint y JavaScript Lint.

Los analizadores de código son herramientas que devuelven observaciones o puntos en los que el código fuente puede mejorarse desde la percepción de buenas prácticas de programación y código limpio.

JSLint

JSLint es un analizador online de código JavaScript creado por Douglas Crockford que permite mostrar puntos en lo que el código no cumpla algunas determinadas reglas establecidas de “código limpio“. [15]

JSLint no es una herramienta óptima ya que es bastante exhaustiva y da muchos falsos positivos. Además tiene muchos detractores que alegan que los criterios evaluados son bastante subjetivos según el punto de vista de su creador. [15]

JSHint

JSHint es una versión mejorada de JSLint desarrollada para corregir todas las incongruencias que presentaba JSLint. El objetivo de JSHint es mejorar las mediciones que eran bastante arbitrarias en JSLint. Es frustrante ejecutar un código realizado y ver como la herramienta menoscaba tu aplicación de una manera innecesaria que era lo que pasaba con JSLint. [14]

JSHint soporta diferentes entornos, navegador y consola, y trabajando con diferentes librerías: jQuery, Prototype, Rhino, Navegador, ES5 (ECMAScript5), Node.js, etc. Como punto positivo la herramienta es también online así que es posible probarla sin necesidad de instalarla. Ambas herramientas, JSLint y JSHint, tienen licencias MIT. [13]

Además, permite configurar una serie de parámetros dependiendo de cada estilo de programación, para que los pase por alto cuando realice la validación:

Permitir estamentos de debug y logging (los típicos 'console.log ()')

Exigir siempre igualdad estricta '==='

Permitir asignaciones dentro de los bucles "if"/"for"/"while"/"do"

JavaScript Lint

Con JavaScript Lint, se puede comprobar todo el código fuente para buscar errores comunes sin tener que ejecutar el script o la apertura de la página web. [12]

También examina las técnicas de codificación utilizadas.

Algunos errores comunes que la herramienta JavaScript Lint busca:

- ✓ Falta un punto y coma al final de una línea.
- ✓ Los palabras reservadas: ("{ }", "if", "for", "while") escritas de forma incorrecta.
- ✓ Las declaraciones de casos en un interruptor que no tiene una sentencia break.
- ✓ Principio y final de puntos decimales en un número.
- ✓ Un cero inicial que convierte un número en octal (base 8).
- ✓ Comentarios en los comentarios.
- ✓ La ambigüedad si dos líneas adyacentes son parte de la misma declaración.
- ✓ Las declaraciones que no hacen nada.

Solo que esta herramienta sustituye el código donde encuentra errores por lo que en realidad cree que es lo correcto y esto no es una cuestión aceptable, puede darse el caso de que la herramienta detecte un error significativo y el desarrollador no lo considere como tal y de ser así se debe pasar por alto, no ir directamente a modificar lo implementado.

De todas estas herramientas JSHint es la que cumple con las características para realizar las pruebas de caja blanca estática, la misma es una libre, multiplataforma, con facilidad de uso y de comprensión para el usuario. Una vez realizadas las pruebas no emite falsos positivos como las demás anteriormente

analizadas, y de encontrar errores no los modifica, solo informa de la existencia del mismo y recomienda posibles soluciones a esos errores, dejando que sea el desarrollador que decida si realizar la modificación.

En el Anexo 5 se establece una comparación de las herramientas.

1.6 Estándares de codificación

Un estándar de codificación es un conjunto de reglas a seguir durante el proceso de escribir el código fuente de una aplicación y abarca todos los aspectos de la generación de código como son el formato de nomenclatura de los diferentes elementos que puede contener y la forma en que deben estar escritos y distribuidos los comentarios en el código. El estándar de programación a utilizar debe ser seleccionado muy cuidadosamente, pero este siempre debe ser pensado de forma que facilite la programación eficaz. Un estándar de codificación bien pensado y aplicado logra hacer parecer que el código entero fue escrito por un solo programador. [20]

Un código fuente debe ser legible, este aspecto tiene influencia directa en la capacidad de un programador para entender el sistema de software. La mantenibilidad es una característica que determina la medida en la que un software puede ser modificado, corregido o mejorado. Si bien la legibilidad y la mantenibilidad son afectadas por diferentes factores, el programador influye mucho al aplicar determinada técnica de codificación y la mejor manera de asegurar que todo el equipo de desarrolladores se apegue a los requerimientos de calidad es establecer un estándar de codificación al que se le someterá a revisiones posteriores.[20]

Cumpliendo condiciones anteriormente planteadas y haciendo uso de técnicas de codificación sólidas y buenas prácticas de programación es el modo de asegurar que un proyecto de software se convierta luego en un sistema fácil de comprender y mantener. [20]

Las revisiones de código deben reforzar los estándares de codificación de manera uniforme, aunque su principal objetivo sea localizar errores en el mismo. El estándar de codificación a utilizar en un proyecto de software debe especificarse al inicio de este, no tiene ningún sentido, ni es prudente imponer un estándar luego de iniciada la codificación. [20]

1.6.1 Ventajas del uso de estándares de codificación

El uso de estándares de codificación conlleva disímiles ventajas, algunas de ellas son: [20]

- ✓ Asegura la legibilidad del código, facilitando el trabajo en un equipo de programadores.
- ✓ Brinda una guía para el encargado de mantenimiento y/o actualización del sistema al contener este un código claro y bien documentado.
- ✓ Facilita la portabilidad entre plataformas y aplicaciones.

1.7 Mantenibilidad

El IEEE1 (19990) define mantenibilidad como: “La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno”. [19]

Esta definición está directamente conectada con la definición del IEEE para mantenimiento del software: “es el proceso de modificar un componente o sistema software después de su entrega para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarlo a cambios en el entorno”. [19]

La mantenibilidad se puede considerar como la combinación de dos propiedades diferentes: La reparabilidad y la flexibilidad. [19]

Reparabilidad

Un sistema software es reparable, si permite la corrección de sus errores con una cantidad de trabajo limitada y razonable. [11]

La reparabilidad se ve afectada por la cantidad y tamaño de los componentes o piezas. Un producto software que consiste en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, pero el incremento del número de módulos no implica un producto más reparable, ya que también aumenta la complejidad de las interconexiones entre módulos. [11]

Así pues, se debe buscar un punto de equilibrio con la estructura de módulos más adecuada para garantizar la reparabilidad facilitando la localización y eliminación de los errores en unos pocos módulos. [11]

Flexibilidad

Un sistema software es flexible si permite cambios para que se satisfagan nuevos requerimientos, es decir, si puede evolucionar. Por su naturaleza inmaterial, el software es mucho más fácil de cambiar o

incrementar por lo que respecta a sus funciones que otros productos de naturaleza física, por ejemplo, equipos hardware, pero esta flexibilidad se ve disminuida con cada nueva versión de un producto software, ya que cada versión complica la estructura del software y, por tanto, las futuras modificaciones serán más difíciles. [11]

Aunque esto puede considerarse una generalidad, la aplicación de técnicas y metodologías apropiadas pueden minimizar el impacto en la flexibilidad de cada nueva modificación en el software. Es por esto que la flexibilidad es una característica tanto del producto software como de los procesos relacionados con su construcción. [11]

1.7.1 Aspectos que influyen en la Mantenibilidad

Dependiendo de cómo se haya construido el software se puede aumentar la mantenibilidad. Los generadores de código, por lo general, no producen un código claro ni fácil de comprender, por lo que el mantenimiento del software así generado es peor. Por otro lado, las técnicas de programación estructurada, la aplicación de metodologías de ingeniería del software y el seguimiento de estándares, permiten la obtención de sistemas o componentes software con menos necesidades de mantenimiento, y en el caso de que se produzca la necesidad de darle mantenimiento al software, será mucho más fácil de llevar a cabo esta tarea. [11]

Decir acerca de la mantenibilidad que:

- ✓ Es el atributo de calidad del software que más directamente influye en los costes y necesidades del mantenimiento:
- ✓ A mayor mantenibilidad menores costes de mantenimiento, y viceversa.
- ✓ La mantenibilidad debe establecerse como objetivo en las fases iniciales del ciclo de vida para reducir las posteriores necesidades de mantenimiento. [11]
- ✓ También se debe tener como objetivo durante la fase de mantenimiento para reducir los efectos laterales y otros inconvenientes ocultos. (Se trata de conocer la situación en la que se encuentra el producto software para poder decidir mejor cómo gestionar y realizar cada petición de mantenimiento y, en general, planificar mejor el proceso de mantenimiento).[11]

1.7.2 Los factores concretos que influyen en la mantenibilidad son los siguientes [11]

- ✓ Falta de cuidado en las fases de diseño, codificación o prueba.
- ✓ Pobre configuración del producto software.
- ✓ Adecuada calificación del equipo de desarrolladores del software.
- ✓ Estructura del software fácil de comprender.
- ✓ Facilidad de uso del sistema.
- ✓ Empleo de lenguajes de programación y sistemas operativos estandarizados.
- ✓ Estructura estandarizada de la documentación.
- ✓ Documentación disponible de los casos de prueba.
- ✓ Disponibilidad del equipo (computador y periféricos) adecuado para realizar el mantenimiento.
- ✓ Disponibilidad de la persona o grupo que desarrolló originalmente el software.

Efectos sobre la **mantenibilidad**: Algunos cambios en el software pueden reducir la mantenibilidad. Los que producen este efecto con más frecuencia son [11]:

- ✓ Violar los estándares de codificación.
- ✓ Reducir la cohesión.
- ✓ Incrementar el acoplamiento.
- ✓ Incrementar la complejidad esencial.

1.7.3 Estándar ISO 9126 del IEEE y la Mantenibilidad

ISO 9126 es un estándar internacional para la evaluación del Software. Está supervisado por el proyecto SQuaRE, ISO 25000:2005, el cual sigue los mismos conceptos. [11]

El estándar está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso.

El estándar identifica 6 atributos clave de calidad entre ellos se encuentra la mantenibilidad. [11]

Mantenibilidad: Facilidad con que una modificación puede ser realizada. Está indicada por los siguientes sub-atributos:

- ✓ Facilidad de análisis
- ✓ Facilidad de cambio
- ✓ Estabilidad
- ✓ Facilidad de prueba
- ✓ Conformidad

1.7.4 Métricas de Mantenibilidad

Métricas internas son aquellas que no dependen de la ejecución del software (medidas estáticas).

Métricas externas son aquellas aplicables al software en ejecución.

La calidad en las métricas de uso están sólo disponibles cuando el producto final es usado en condiciones reales.

Idealmente, la calidad interna determina la calidad externa y esta a su vez la calidad en el uso.

Las métricas de mantenibilidad miden atributos relacionados con la conducta del mantenedor, el usuario o el sistema software, cuando dicho software se mantiene o se modifica durante la realización de pruebas o el mantenimiento. [11]

Analizabilidad: miden atributos relacionados con el esfuerzo del mantenedor o el usuario o los recursos gastados para diagnosticar deficiencias o causas de fallos, o para identificar las partes que deben ser modificadas. [11]

Métrica de Analizabilidad:

Tiempo medio en analizar un fallo

X = sum (Tout-Tin) / N

Siendo:

Tout = momento en el que se encuentran las causas del fallo (o son reportadas por el usuario)

Tin = momento en el que se recibe el informe del fallo y N = número total de fallos registrados

Cambiabilidad: miden atributos relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema software cuando se intenta llevar a cabo una modificación determinada. [11]

Métrica de Cambiabilidad:

-Registrabilidad de cambios

Fórmula:

X = A/B

Siendo:

A = número de cambios a funciones o módulos que tienen comentarios confirmados.

B = total de funciones o módulos modificados.

Interpretación:

0 <= X <= 1

Mientras más cercano a 1, más registrable.

0 indica un control de cambios deficiente o pocos cambios y alta estabilidad.

Estabilidad: miden atributos relacionados con la conducta inesperada del sistema software cuando dicho software es probado u operado después de una modificación. [11]

Métrica de Estabilidad:

Fórmula**:**

X = 1 - A / B

– Frecuencia de fallos debidos a efectos laterales producidos después de una modificación

Siendo:

A= número de fallos debidos a efectos laterales detectados y corregidos

B= número total de fallos corregidos

Cuanto mayor sea X es predecible que más difícil será de mantener en el futuro el sistema.

Facilidad de prueba: miden atributos relacionados con el esfuerzo del mantenedor o el usuario para medir la conducta del mantenedor, el usuario o el sistema software cuando se intenta probar el software. [11]

Métrica de Facilidad de prueba o Examinabilidad

Ejemplos:

Pruebas sin esfuerzo

Fórmula:

Sum (T) / N

Siendo:

T = tiempo empleado en probar con el fin de asegurar si el informe de fallo ha sido o no resuelto.

N = número de fallos resueltos.

Interpretación:

Si 0 <= X <= 180 segundos, la conducta del mantenedor, el usuario o el sistema de software es Correcta.

Si X > 180 segundos, la conducta del mantenedor, el usuario o el sistema de software es Incorrecta.

Disposición de funciones de prueba predefinidas

Fórmula:

X = A/B

Siendo:

A = número de veces que el personal de mantenimiento puede utilizar funciones de prueba predefinidas adecuadas.

B = número de oportunidades de prueba.

Reiniciabilidad de pruebas

Fórmula:

X = A/B

Siendo:

A = número de veces que el personal de mantenimiento puede hacer una pausa y reiniciar al ejecutar un programa de prueba en los puntos deseados para comprobar paso a paso.

B = número de veces de pausa al ejecutar un programa de prueba

Conformidad: miden atributos relacionados con el número de casos u ocurrencias en que el producto software no cumple las normas, convenciones o regulaciones requeridas relacionadas con la mantenibilidad. [11]

Métrica de Conformidad de la mantenibilidad:

Ejemplo:

Cobertura de satisfacción de elementos de conformidad relativos a la mantenibilidad.

Fórmula:

X = 1 – (A / B)

Siendo:

A = número de elementos de conformidad fallados durante las pruebas.

B = número de elementos de conformidad totales.

Luego de haber interpretado las métricas anteriores expuestas en este documento se decide utilizar las siguientes debido a que fueron las más fáciles de comprender ya que los datos necesarios para poder llevarlas a cabo los proporciona la herramienta a utilizar y los que no, pueden ser obtenidos sin dificultad alguna:

- ✓ Métrica de Cambiabilidad
- ✓ Métrica de Facilidad de prueba o Examinabilidad (Prueba sin Esfuerzo)

1.8 Conclusiones parciales

En este primer capítulo se realizó una revisión de varias documentaciones, referente a las pruebas de caja blanca estática, se definió una herramienta capaz de realizarle revisiones al código JavaScript de la capa de presentación de los componentes desarrollados bajo la tecnología Sauxe.

Hacer uso de esta técnica no traerá costo ni esfuerzo ninguno, ya que la herramienta a utilizar para aplicar la técnica de caja blanca estática es libre, fácil de utilizar y permite mejorar el trabajo de los desarrolladores.

Se puede concluir que incorporar la mantenibilidad en los productos de software es una actividad importante, ya que puede traer como consecuencias que el producto a la hora de recibir mantenimiento pueda necesitar el empleo de más recursos y esfuerzo, que lo que se utilizó en su desarrollo.

Además se pudo determinar mediante la investigación realizada, que la tecnica de caja blanca no solo detecta los errores que presenta el software sino que también ayuda a incorporar la mantenibilidad en el mismo. El uso de estándares de codificación en la implementación, contar con un equipo de desarrollo capaz de implementar el código como si fuera una sola persona la que se haya encargado de la fase de desarrollo, incorporar todos los posibles comentarios en las funcionalidades realizadas, para facilitar su entendimiento, así como reducir al máximo la complejidad esencial de las soluciones, dan lugar a que se desarrolle un software mantenible y fácil de realizarle mantenimiento.

También se lograron encontrar métricas internas que pueden medir el atributo mantenibilidad, a partir de sub-características de la misma y que se pueden aplicar en el código que será revisado, para así poder determinar, el grado de mantenibilidad del código fuente JavaScript de los productos informáticos que se desarrollan bajo la tecnología Sauxe en el CEIGE.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En este capítulo se describe la solución a proponer, para aplicar pruebas de caja blanca estática al código fuente de la capa de presentación de los sistemas que se encuentren en desarrollo bajo la tecnología Sauxe y pertenezcan además al CEIGE. Se establecen también los objetivos y alcance de la propuesta de solución, se describen las actividades a realizar, los roles a participar y artefactos a ser generados para que sean llevadas a cabo las pruebas de caja blanca estática.

2.2 Nombre de la Solución

Procedimiento para realizar pruebas de caja blanca estática al código de la capa de presentación de los sistemas que se encuentren en desarrollo bajo la tecnología Sauxe.

2.2.1 Objetivo

Establecer un procedimiento para realizar pruebas de caja blanca estática al código de la capa de presentación de los sistemas en desarrollo bajo la tecnología Sauxe en el CEIGE.

Definir roles, actividades y artefactos a intervenir en las pruebas de caja blanca estática.

2.2.2 Alcance

La propuesta de solución podrá ser aplicada en soluciones informáticas que se encuentren en desarrollo bajo la tecnología Sauxe.

2.2.3 Términos y Definiciones

Actividades: es el conjunto de acciones que se llevan a cabo para cumplir las metas de un programa o subprograma de operación, que consiste en la ejecución de ciertos procesos o tareas.

Artefactos: productos tangibles del proyecto, que son producidos, modificados y usados por las actividades.

Procedimiento: es la acción de proceder o el método de ejecutar algunas cosas. [26] Se trata de una serie común de pasos definidos, que permiten realizar un trabajo de forma correcta.

Rol: papel que asume un individuo en una organización.

2.2.4 Referencia

- ✓ Procedimiento para la realización de pruebas de caja blanca en la UCID.
- ✓ Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS.
- ✓ Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías.

2.2.5 Roles y Responsabilidades

Los roles que a continuación se describen, están dados luego de haber aplicado una encuesta, en algunos de los distintos departamentos del CEIGE que desarrollan bajo la tecnología Sauxe, para definir los que pudieran interactuar con los del grupo calidad del CEIGE. Los resultados fueron los siguientes. Para ver las preguntas realizadas ver Anexo 1.

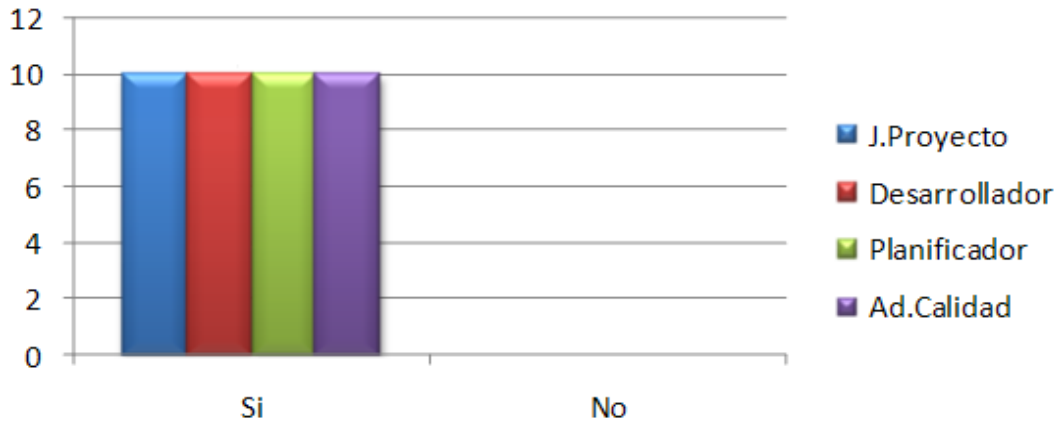


Figura 1 Resultado de la encuesta para la detección de roles en los proyectos.

De 10 proyectos encuestados los roles a intervenir en la solución, están presentes en todos los proyectos.

Administrador de calidad: Es el máximo responsable del éxito de las pruebas, pues es el encargado de verificar y asegurar que se realice una correcta planificación y administración de los recursos. Comprueba el progreso y efectividad de las pruebas, evalúa los resultados de cada ciclo, y debe dar solución a los problemas que impiden el buen desarrollo del proceso.

Probador: Encargado de realizar las pruebas según hayan sido planificadas en el cronograma de pruebas, además de registrar y reportar las no conformidades encontradas al código fuente.

Desarrollador: En la etapa de corrección de los errores es el encargado de asumir esta tarea y de actualizar la última versión del código fuente en el repositorio siempre que se inicie una iteración.

2.2.6 Artefactos a generarse en el proceso de pruebas de caja blanca estática

1- Registro de Violaciones. El mismo es creado a partir de modificaciones previamente realizadas al artefacto Registro de No Conformidades utilizado como artefacto en las pruebas de caja negra por el grupo de calidad del CEIGE.

2- Registro de No conformidades. El mismo es utilizado como artefacto en las pruebas de caja negra (funcionales) por el grupo calidad de CEIGE. Para ajustarlo a estas pruebas fue necesario realizar una pequeña modificación en la leyenda donde los tipos de errores a detectar no son los establecidos en el artefacto.

3- Plan de pruebas. El mismo está establecido por Calisoft para llevar a cabo las pruebas de caja negra (funcionales) al software, pero para ser utilizado en este tipo de pruebas sufrió previas modificaciones.

4- Cronograma de pruebas. Está incluido en el artefacto plan de pruebas.

5- Informe final de las pruebas. Es un artefacto que se crea para introducir los resultados que se obtienen a lo largo de todo el proceso de pruebas.

Artefacto a consultarse

1- Estándar de codificación. Artefacto establecido para la estandarización del código fuente desarrollado bajo la tecnología Sauxe.

2.2.7 Criterios de Criticidad

Se tuvieron en cuenta para establecer estos criterios los mismos que se definen para abortar las pruebas de caja negra, solo que se le hicieron algunas modificaciones para su aplicación en las pruebas de caja blanca estática.

Los Criterios de Criticidad establecen los parámetros para que se continúen realizando las pruebas.

Son aplicables en 2 momentos fundamentales:

- ✓ Cuando se hace la reunión de inicio con el equipo de proyecto (RI).
- ✓ Cuando se verifica el cumplimiento del estándar de codificación por parte del subsistema. (VCEC)
- ✓ Durante las iteraciones de prueba (IP).

Prueba Detenida (PD): Se detiene la prueba y se reinicia en la misma actividad que se detuvo. Como máximo una prueba puede estar detenida 1 semana, este plazo se decide según los motivos que hicieron que se parara y debe cumplirse por parte del equipo de desarrollo. En caso de que no se cumpla el tiempo, se declara Prueba Abortada.

Criterios de Criticidad para Pruebas Detenidas:

- ✓ No se presenta el desarrollador de apoyo requerido para realizar la prueba (RI).
- ✓ No está lista la versión del artefacto que debe probarse (RI) (IP).
- ✓ Se mantienen NC detectadas en iteraciones anteriores (IP).
- ✓ Se ha violado el cumplimiento del estándar de codificación indiscriminadamente (VCEC).

Prueba Abortada (PA): Se detiene la prueba y para reiniciarla debe hacerse una nueva solicitud, comenzando el proceso desde el inicio. Como mínimo se necesitan 3 días para recomenzar la prueba.

Criterios de Criticidad para Pruebas Abortadas:

- ✓ No están presentes todos los elementos componentes del sistema, producto o entregable (hardware, software, documentación y artefactos de apoyo) (RI) (IP).
- ✓ No se corresponden totalmente los requisitos funcionales documentados con los implementados. (IP).

- ✓ Supera el artefacto la cantidad de NC Significativa por unidad de revisión (IP) (Las unidades de revisión pueden ser funcionalidades.)
- ✓ Excede el producto las 3 iteraciones establecidas en la planeación inicial (IP).
- ✓ Se incumplen las reglas para la documentación establecidas por la entidad y/o el proyecto (IP).
- ✓ Existe incoherencia entre lo documentado y lo implementado (IP).
- ✓ Se incumple con lo pactado en el Plan de Pruebas (IP).
- ✓ Se excede el tiempo total de la prueba según lo planificado en el cronograma pactado en el Plan de Pruebas.

2.2.8 Técnica de prueba a emplear

1- Para realizar pruebas de caja blanca estática al código JavaScript de la capa de presentación de los componentes en desarrollo bajo la tecnología Sauxe, se utilizará la herramienta JSHint que automatizará una parte del proceso de pruebas.

2- Para realizar pruebas de caja blanca estática de forma manual al código JavaScript de la capa de presentación de los componentes en desarrollo bajo la tecnología Sauxe, se hará mediante el artefacto estándar de codificación, revisando que el código fuente este implementado, cumpliendo las pautas que establece el mismo.

2.2.9 Normas Generales (Precondiciones)

Para la aplicación de este procedimiento debe considerarse su puesta en funcionamiento sobre un módulo que aún no esté terminado. El probador debe tener acceso al código fuente del módulo a probar, para ello deberá tener instalado en su PC lo necesario para poder acceder al código fuente de la capa de presentación, de los componentes en desarrollo bajo la tecnología Sauxe, así como tener instalado en su PC la herramienta de apoyo para realizar las pruebas de caja blanca estática (JSHint). Además tiene que tener a su disposición el estándar de codificación definido para este lenguaje de programación.

2.2.9.1 Para poder acceder al código fuente de los componentes desarrollados bajo la tecnología Sauxe es necesario seguir estas indicaciones

Para Linux:

- ✓ Se tiene que tener instalado el RapidSVN, para poder descargar el repositorio de Sauxe.
- ✓ Se abre la herramienta RapidSVN una vez instalada, en la dirección: Aplicaciones/Programación/ RapidSVN.
- ✓ Luego se selecciona la opción Comprobación de nueva copia de trabajo.
- ✓ Se abre la ventana Comprobación y en el campo url se ingresa la dirección del repositorio y en el campo directorio de destino donde se desea guardar el repositorio.
- ✓ Siempre tener en cuenta que se necesita de un usuario y contraseña con permiso de acceso al repositorio.

Para Windows:

- ✓ Se tiene que tener instalado la herramienta TortoiseSVN para poder descargar el repositorio de Sauxe.
- ✓ Se crea una carpeta donde desee guardar el repositorio de Sauxe.
- ✓ Luego se da clic derecho sobre esa carpeta y se selecciona la opción SVNCheckKout
- ✓ En la ventana que se abre se ingresa la url donde está el repositorio y luego se da Aceptar.
- ✓ Siempre tener en cuenta que se necesita de un usuario y contraseña con permiso de acceso al repositorio.

2.3 Diagrama de procesos

A continuación se establece una visión general de todos los subprocesos por los que transitará este procedimiento de pruebas de caja blanca estática. Donde los mismos seguirán una secuencia lineal para ser llevadas a cabo, sin que haya que detenerse en la ejecución de algún subproceso en particular.

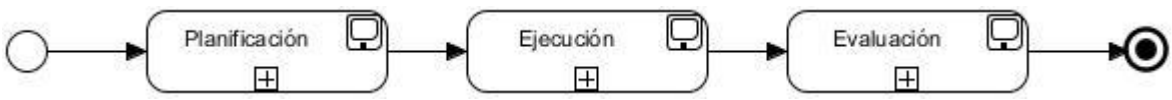


Figura 2 Fases del procedimiento.

Tabla 1 Descripción de las fases del procedimiento.

Subprocesos	Descripción
Planificación	En este subproceso se definirá como se va a realizar el proceso de pruebas de caja blanca estática, todo lo necesario para llevar a cabo este proceso de pruebas se establecerá en esta etapa.
Ejecución	Este subproceso será el encargado de verificar que los desarrolladores hayan programado los componentes en lenguaje de programación JavaScript rigiéndose por el estándar de codificación definido. Además se encargará de detectar los errores que cometieron los desarrolladores a la hora de implementar los componentes de software en el lenguaje JavaScript.
Evaluación	En este subproceso se le aplicarán a esos errores encontrados métricas que miden sub-características de la mantenibilidad para determinar cuan mantenible son los componentes que pasaron por este proceso de pruebas. También se calculará la complejidad ciclomática y la complejidad esencial para determinar si el software que se desarrolla es fiable y mantenible.

2.3.1 Desarrollo de la propuesta de solución

Para el desarrollo de la propuesta de solución se definieron un conjunto de actividades que serán llevadas a cabo para lograr un correcto funcionamiento de este proceso de pruebas de caja blanca estática.

2.4 Actividades a desarrollar

Las actividades definidas son descritas a continuación con la mayor claridad posible para que sean fáciles de entender y de cumplir en el proceso de pruebas de caja blanca estática.

Paso 1: Planificar pruebas de caja blanca estática.

Para ver el diagrama de actividades del subproceso planificación ver Anexo 6.

Tabla 2 Descripción textual del subproceso Planificar pruebas de caja blanca estática.

Precondiciones: El jefe de proyecto ha llenado una solicitud de prueba de caja blanca estática.				
Roles	**Entradas**	**Actividades**	**Descripción**	**Salidas**
Jefe de Proyecto	Solicitud de prueba	1. Enviar solicitud.	Se solicita al administrador de calidad realizar pruebas de caja blanca estática al software.	
Administrador de calidad	Solicitud de prueba	2. Revisar solicitud	Se revisa que los datos de la solicitud sean correctos. De no estar correcta se pasa al flujo alterno 1.a.	
		3. Crear Plan de Pruebas	Se crea el Plan de pruebas para definir cómo se va a llevar a cabo el proceso de pruebas de caja blanca estática.	Plan de pruebas. Cronograma de pruebas.
		4. Informar día de Reunión de inicio	Se acuerda la fecha de la reunión de inicio para aprobar el Plan de Pruebas	
Jefe de proyecto		5.Confirmar participación en la reunión de inicio	El jefe de proyecto confirma su presencia en la reunión de inicio	
Administrador de calidad		6. Realizar reunión de inicio.	Se lleva a cabo la reunión de inicio para aprobar el Plan de Pruebas y el Cronograma de Pruebas.	
Flujo Alterno				
Administrador de		1. a Reenviar Solicitud de	Al no estar correcta la solicitud el jefe de	

calidad	prueba.	proyecto debe corregir los errores de la solicitud
Jefe de Proyecto	1.b Corregir solicitud de prueba	El jefe de proyecto corrige la solicitud de prueba. Pasar a la actividad 1.

Para llegar a conclusiones de que esta actividad era necesaria fue aplicada una encuesta, a algunos de los desarrolladores del CEIGE, que utilizan la tecnología Sauxe para el desarrollo de las soluciones informáticas. Los resultados se muestran en la gráfica siguiente. Para ver las preguntas realizadas Ver Anexo 2.

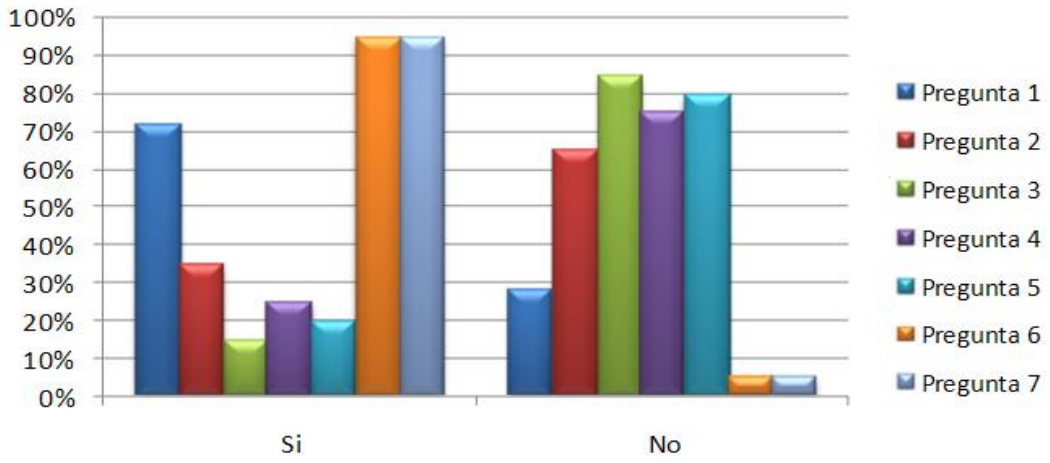


Figura 3 Resultados de la encuesta sobre el uso del estándar de codificación.

De 20 desarrolladores encuestados se obtuvo como resultado que un 72% de los encuestados conoce qué es un estándar de codificación, pero un 65% no conoce el que está establecido para el desarrollo del lenguaje JavaScript, un 85% no se rige por el estándar de codificación para implementar las soluciones informáticas, un 75% no posee alguien encargado de revisarles, que lo que implementan cumpla con el estándar de codificación, un 80% afirma que después de implementar, no revisan que lo desarrollado, se rija por el estándar de codificación y el 95% cree que es necesario regirse por un estándar de codificación para desarrollar y que alguien les revise que lo que han implementado cumpla con el estándar de codificación definido.

Paso 2: Ejecutar pruebas de caja blanca estática.

Para ver diagrama de actividades del subproceso ejecución ver Anexo 7.

Tabla 3 Descripción textual del subproceso Ejecutar pruebas de caja blanca estática.

Precondiciones: Se ha creado el plan de pruebas Se ha creado el cronograma de pruebas				
Roles	**Entradas**	**Actividades**	**Descripción**	**Salidas**
Probador		7. Acceder al código que será revisado por el grupo de calidad.	Se descarga el repositorio para acceder al código fuente que se va revisar en las pruebas de caja blanca estática.	Código Fuente.
	Código Fuente Estándar de codificación	7.1. Comenzar a revisar código fuente de forma manual	Se comienza el proceso de ejecución de las pruebas una vez que está todo listo para comenzar a probar el código fuente definido para esa iteración. Si fueron encontrados errores pasar al flujo alterno10.1.a	Registro de violaciones
		7.2. Informar al administrador de calidad	De no encontrarse errores en el código fuente se le informa al administrador de calidad que no hay errores detectados	
	Registro de NC Código fuente	7.3. Realizar pruebas al código fuente con la herramienta	Se comienzan las pruebas al código fuente con la herramienta. Si no se encontraron errores pasar al flujo alterno10.3.a	
		7.4. Registrar errores.	Se registran los errores detectados en el registro de NC	Registro de NC
		7.5. Entregar errores	Se le entrega el registro de NC al administrador de calidad con los errores detectados	Registro de NC
Administrador de calidad	Registro de NC	7.6. Enviar errores	Se le envía al desarrollador el Registro de NC para que sean resueltas	
Desarrollador	Registro de NC	7.7. Corregir errores	El desarrollador corrige todas las NC encontradas	
		7.8. Informar de errores resueltos	Una vez que el desarrollador ha resuelto todas las NC encontradas, so lo informa al administrador de calidad	
Probador	Código fuente	7.9. Comenzar pruebas de regresión con	El probador verifica con la herramienta de pruebas que las NC encontradas han sido resueltas. Si no se resolvieron	Registro de NC

		la herramienta	las NC pasar al flujo alterno 7.9.a	
		7.10. Informar de errores resueltos	El probador le informa al administrador de calidad que las NC han sido resueltas.	
Administrador de calidad		7.11. Pasar a otra iteración	El administrador de calidad decide si es necesario realizar otra iteración de pruebas. De pasar a otra iteración ver flujo alterno 7.3.a	
Flujo Alterno				
Probador		7.1.a. Registrar errores	El probador registra los errores encontrados	Registro de violaciones
	Registro de violaciones	7.1.b. Entregar errores	El probador entrega los errores registrados en el registro de violaciones al administrador de prueba	
Administrador de calidad		7.1.c. Enviar errores	El administrador de calidad envía al desarrollador los errores encontrados para que sean resueltos.	Registro de violaciones
Desarrollador	Registro de violaciones	7.1. d. Corregir errores.	El desarrollador corrige los errores detectados	
		7.1.e. Informar de errores resueltos	El desarrollador informa al administrador de calidad que los errores han sido resueltos	
Probador		7.1.f. Confirmar que los errores fueron corregidos	El probador pasará a confirmar que los errores han sido resueltos. Pasar a la actividad 7.1	
Probador		7.3.a Informar al administrador de calidad	Se le informa al administrador de calidad que no fueron encontradas NC	
		7.3.b Comenzar otra iteración de pruebas	Se pasa a otra iteración de pruebas. Ver actividad 7.3	
		7.9.a Informar de errores no resueltos	Se le informa al administrador de pruebas que los errores no han sido resueltos. Ver actividad 7.6	

Manual de usuario de la herramienta

1- Se selecciona el editor de texto de Linux.

1.1- Aplicaciones/Accesorios/Editor de textos.

2- Luego en la opción herramientas del gedit se busca JSHint.

2.1- Herramienta/ClientSide/JSHint.

3- Una vez que la aplicación esté lista para usarse se introducirá el código a revisar copiando y pegando el mismo en la parte superior del cuadro de texto que aparece en la herramienta.

4- Una vez copiado y pegado el código en la parte superior de la herramienta se selecciona la secuencia de pasos:

4.1- Herramienta/ClientSide/JSHint.

Luego en la parte inferior aparecerán todos los errores encontrados una vez que la aplicación haya analizado el código.

Como se puede ver es una herramienta fácil de utilizar, pues el trabajo con la misma no es nada complejo.

Para ver la interfaz de la herramienta ver Anexo 9.

Paso 3: Evaluar las pruebas de caja blanca estática.

Para el cumplimiento de esta fase se partirá de las NC detectadas, una vez que se tengas todas estas NC se procederá a evaluar el código fuente probado mediante dos métricas una de ellas es la Facilidad de prueba del sistema y otra es la Registrabilidad de cambios del mismo, ambas métricas son sub-características de la mantenibilidad dependiendo de que el valor de las mismas sea favorable, se podrá determinar si el sistema que se está construyendo es fácil de probar, si es posible que se registren cambios en el mismo o si es estable.

Para ver el diagrama de actividades del subproceso evaluación ver Anexo 8.

Tabla 4 Descripción textual del subproceso Evaluar pruebas de caja blanca estática.

Precondiciones: Se han registrado todas las NC encontradas				
Roles	**Entradas**	**Actividades**	**Descripción**	**Salidas**
Administrador de Calidad	Listado de NC	8. Aplicar métricas según la cantidad de NC registradas.	Según la cantidad de NC detectadas se le aplicarán las métricas definidas, y se establecerá el grado de mantenibilidad que alcanzó el código fuente en cuanto a facilidad de prueba y registrabilidad de cambios que posea el sistema desarrollado. De no haberse registrado NC pasar al flujo alterno 8.a.	
		9. Realizar informe final del proceso de pruebas de caja blanca estática.	Resumen de todo el proceso de pruebas según los resultados obtenidos en las mismas. Ver flujo alterno 8.b.	Informe final
Flujo Alterno				
		8. a. Se concluye la etapa de pruebas.	El módulo probado está 100% libre de errores.	Informe final
	Informe final	8.b. Realizar reunión de cierre	Se realiza la reunión de cierre y se dan los resultados del proceso de pruebas de caja blanca estática.	

2.5 Conclusiones parciales

En este capítulo se definió la propuesta de solución a llevar a cabo para realizar las pruebas de caja blanca estática a la capa de presentación de los sistemas en desarrollo bajo la tecnología Sauxe. Se definió una estrategia fácil de aplicar y de entender, pero que se espera que sea una solución provechosa y exitosa una vez aplicada.

Las fases definidas constan de actividades que garantizan una mejora del desarrollo del software, ya que a medida que se vaya implementando el código fuente, se le harán revisiones al código desarrollado y se velará porque el mismo esté estandarizado, todo esto se realizará, antes que el producto esté finalmente terminado.

CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

3.1. Introducción

En este capítulo se lleva a cabo la validación de la propuesta de solución. Para ello se aplicará el procedimiento propuesto sobre la vista de presentación de algunos componentes que se encuentra en desarrollo en el subsistema Capital Humano del proyecto ERP perteneciente al CEIGE desarrollado bajo la tecnología Sauxe. Por lo que cumple con la característica de poder ser aplicado sobre el mismo, las pruebas de caja blanca estática.

Para la aplicación del procedimiento se llevaron a cabo todas las actividades definidas en el Capítulo 2, siendo aplicadas sobre la capa de presentación de algunos componentes que están en desarrollo en el subsistema Capital Humano desarrollado bajo la tecnología Sauxe.

3.2 Planificar pruebas de caja blanca estática

Para el cumplimiento de esta actividad, una vez aprobada la Solicitud de pruebas de caja blanca estática por el Administrador de Calidad, se efectuó la reunión de inicio en la que estuvieron presentes el Jefe de Proyecto, Planificador , Administrador de calidad y el Jefe del grupo de calidad donde se aprobó el Plan de pruebas de caja blanca estática para el subsistema Capital Humano, donde quedó acordado todo lo referente al proceso de pruebas de caja blanca estática, además los involucrados estuvieron todos de acuerdo con lo planteado en el plan de pruebas y se comprometieron a cumplir con lo que se estableció en el mismo.

El Plan de Pruebas una vez aprobado incluyó los siguientes puntos:

3.2.1 Plan de prueba de caja blanca estática para el subsistema Capital Humano

Definiciones y acrónimos

CEIGE: Centro de Informatización de la Gestión de Entidades.

No Conformidades (NC): Errores existentes en el producto que hacen que el mismo no cumpla con los requisitos especificados por el cliente para su uso o exista un incumplimiento de los estándares definidos.

Plan de Prueba: Es un artefacto que define estrategias y recursos necesarios para ejecutar una metodología de pruebas que incluye los objetivos de las pruebas, el enfoque que se adopta. Constituye el

documento Rector de los procesos de pruebas y sirve como guía de referencia para las partes involucradas.

Pruebas de Regresión: Pruebas que verifican que los errores detectados fueron corregidos y que este proceso no afecte funcionalidades que operaban correctamente.

3.2.2 Introducción

Este documento se confecciona con el objetivo de definir el Plan de pruebas de caja blanca estática del artefacto (Código Fuente) de Capital Humano, solicitado por el Jefe del proyecto Capital Humano.

En el plan de pruebas definido para el sistema en cuestión, se identifican los elementos que serán probados, los recursos necesarios para hacer las pruebas, así como la estrategia de pruebas que se llevará a cabo para lograr que se encuentren con este proceso, la mayor cantidad de errores en la implementación (código fuente) del software. Todo este trabajo debe ser documentado y aprobado por los participantes en las pruebas que se realicen.

3.2.3 Descripción del Proyecto

El sistema debe ser capaz de:

- ✓ Gestionar toda la información relacionada con las personas de una entidad.
- ✓ Gestionar los puestos de trabajo y el régimen de trabajo descanso.
- ✓ Gestionar toda la información referente a los trabajadores de la entidad.
- ✓ Permitir que se puedan contratar trabajadores, así como darle seguimiento a la vida laboral de los mismos.
- ✓ Generar de forma correcta la nómina, así como brindar un conjunto de reportes necesarios para el trabajo con el capital humano.
- ✓ De forma general gestionar toda la información referente al capital humano de una entidad y finalmente generar la nómina.

3.2.4 Objetivo

Los objetivos del Plan de Pruebas de caja blanca estática:

- ✓ Identificar los elementos, recursos y configuraciones necesarios para realizar las pruebas de caja blanca estática.
- ✓ Describir las estrategias de las pruebas a ser empleadas.
- ✓ Definir el cronograma de las pruebas.

3.2.5 Alcance

Este documento involucra al subsistema Capital Humano del Proyecto ERP perteneciente al CEIGE. El alcance de las pruebas está dado por la intención de evaluar el código fuente en busca de errores cometidos por los desarrolladores y verificar que el mismo cumpla con el estándar de codificación que fue establecido para cumplir con las pautas de programación establecidas.

3.2.6 Estrategia de pruebas de caja blanca estática

Teniendo en cuenta las características del producto Capital Humano se define la siguiente estrategia para la aplicación de las pruebas:

Inicialmente las pruebas que se le aplicarán al producto son las de caja blanca estática. Estas pruebas se basan en encontrar errores en el código fuente a evaluar, sin la necesidad de ejecutarlo. Se comprueba, que el código implementado no contenga errores, esto facilitará en fases posteriores la etapa del mantenimiento del software. Esta técnica no requiere de grandes esfuerzos, ni recursos para su ejecución.

La actividad ejecutar pruebas comienza inicialmente de forma manual verificando que el código fuente esté estandarizado, o sea que cumpla con el estándar de codificación definido para el lenguaje JavaScript. Una vez que ha sido llevada a cabo la evaluación manual, se procede a evaluar el código de manera automatizada, mediante el uso de la herramienta de pruebas establecida, la cual analiza el código JavaScript en busca de errores de codificación y detecta secciones críticas en el código, que pueden ser mejoradas, la propia herramienta, recomienda cómo hacer estas secciones más eficiente.

Cuando el código fuente del módulo esté listo para una 1ra iteración de pruebas, se seleccionan los probadores a participar en esta tarea y se procede a la realización de las pruebas, para detectar las NC que presenta el código fuente.

Cuando son registradas las NC el administrador de calidad es el encargado de mandarlas al desarrollador del subsistema Capital Humano, realizándose luego de la respuesta a las NC por parte del desarrollador, las pruebas de regresión por el probador del grupo calidad, con el objetivo de verificar la corrección de las NC antes señaladas y analizar si con la corrección de estos errores no se introdujeron otros nuevos.

De este mismo modo se realizarán la 2da y 3ra iteración, mientras no se cumpla algún criterio de criticidad, hasta alcanzar la liberación del código fuente del producto.

En la medida que se detecten errores, molestias o incomodidades en el trabajo con el código fuente del producto, etc., estas serán anotadas en el Anexo 4 para el caso de los errores detectados de forma manual y en el Anexo 3 para el caso de los errores detectados por la herramienta.

Al finalizar la sesión de pruebas, estas no conformidades son entregadas al administrador de calidad, quien las envía al desarrollador para su posterior corrección.

Al concluir el período de pruebas se realizará el Informe Final, atendiendo a los resultados por cada iteración de pruebas.

Una vez terminada de aplicar al subsistema todas las fases definidas, se libera el código fuente del subsistema Capital Humano para proceder a nuevas implementaciones y el jefe de proyecto y el administrador de calidad deberán firmar un acta de liberación para el código fuente probado.

3.2.7 Roles y responsabilidades

Tabla 5 Roles y responsabilidades.

Rol	Cantidad	Responsabilidad
Administrador de Calidad	1	Diseñar la estrategia de prueba a seguir durante las Pruebas de Liberación. Confeccionar el Plan de pruebas de caja blanca estática y velar por su estricto cumplimiento. Revisar y asignar al Proyecto Capital Humano las NC detectadas. Conciliar con el Equipo de Proyecto las NC detectadas. Realizar una reunión con los involucrados en el proceso, para realizar el cierre de las pruebas caja blanca estática.
Desarrollador	1	Apoyar a los probadores en la ejecución de las pruebas.

		Responder las NC detectadas durante la ejecución de las pruebas en el tiempo establecido. Actualizar la última versión del código fuente en el repositorio siempre que se le realicen nuevas revisiones al código fuente.
Probador	1	Ejecutar las pruebas de caja blanca estática. Documentar las NC detectadas durante la ejecución de las pruebas.

3.2.8 Escenario de pruebas

✓ **Recursos**

PC Cliente

Tabla 6 Descripción de los recursos.

PC Cliente	
Cantidad	1
Descripción	HDD 80 GB, 0.99 GB RAM Mínimo, Pentium IV o superior.
Software base	Multiplataforma (Windows- Linux).JSHint, RapidSVN o TortoiseSVN

3.2.9 Cronograma de pruebas

Tabla 7 Cronograma de pruebas.

No.	Tarea	Fecha	Responsable	Participantes	Observaciones
1.	Elaboración del Plan de Pruebas.	28/04/2012	Administrador de calidad	Administrador de calidad	
2.	Reunión de Inicio.	30/04/2012	Administrador de calidad	Jefe de Proyecto ,Planificador, Administrador de calidad y jefe del grupo de calidad	
3.	Aprobar Plan	30/04/2012	Administrador	Jefe de	

	de Pruebas.		de calidad	Proyecto,Planificador, Administrador de calidad y Jefe del grupo de calidad	
4.	Revisar el cumplimiento del estándar de codificación definido.	2/05/2011 al 4/05/2012	Probador	Probador,Desarrollador	

3.3 Ejecutar pruebas de caja blanca estática

Para la ejecución de las pruebas de caja blanca estática se descargó del repositorio del marco de trabajo Sauxe el código fuente a realizarle las pruebas de caja blanca estática.

La primera parte de esta fase es la revisión del código fuente de forma manual, se verificó que el código fuente cumpliera con el estándar de codificación para JavaScript.

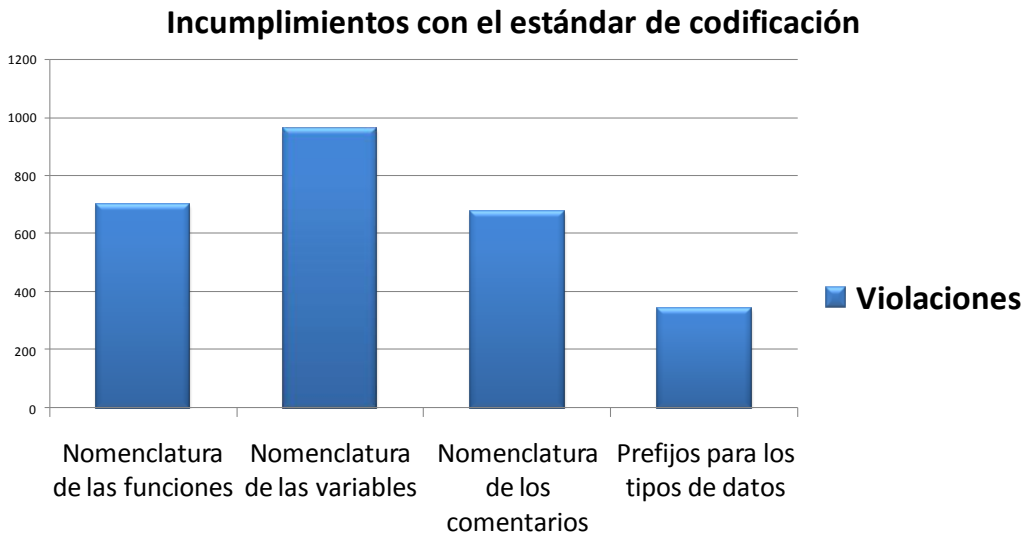


Figura 4 Resultados de la revisión del cumplimiento del estándar de codificación.

Se encontraron entre las 10 clases probadas 700 violaciones en las nomenclaturas de las funciones, 960 en las nomenclaturas de las variables, 675 en la nomenclatura de los comentarios y 340 en los prefijos para los tipos de datos.

Estos datos son la constancia de que sí se cometen violaciones por parte de los desarrolladores en la implementación del código fuente, pues no tienen en cuenta las pautas que establece el estándar de codificación para JavaScript, pautas que son inviolables y obligatorias a cumplir en la fase de implementación.

Una vez terminada esta actividad se pasó a evaluar el código fuente con la herramienta de pruebas, los resultados fueron:

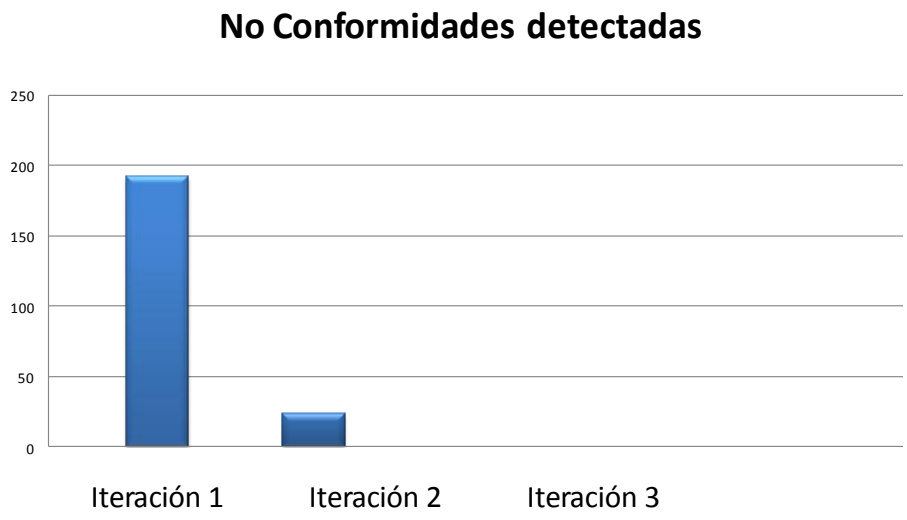


Figura 5 Resultado de las iteraciones de prueba.

Se realizaron un total de 2 iteraciones de pruebas, donde en la primera iteración se encontraron unas 192 NC y en la segunda iteración un total de 23 NC, luego de estas dos iteraciones realizadas, no se realizó una tercera iteración, pues no se detectaron nuevos errores en el código fuente de las clases JavaScript probadas, de manera general se encontraron 215 NC en total, como resultado del proceso de pruebas llevado a cabo, y todas las NC encontradas fueron resueltas.

El subsistema Capital Humano actualmente tiene 8 componentes en desarrollo, para la solución se probaron 5 componentes, la capa de presentación de estos contiene en total de 10 clases las cuales fueron evaluadas, las mismas son:

1-Gestionar Impuestos

2-Asociar Impuestos

3-Gestionar Pagos Adicionales

4-Gestionar Concepto de Pago

5-Gestionar Periodo de Pago

6- Contratos

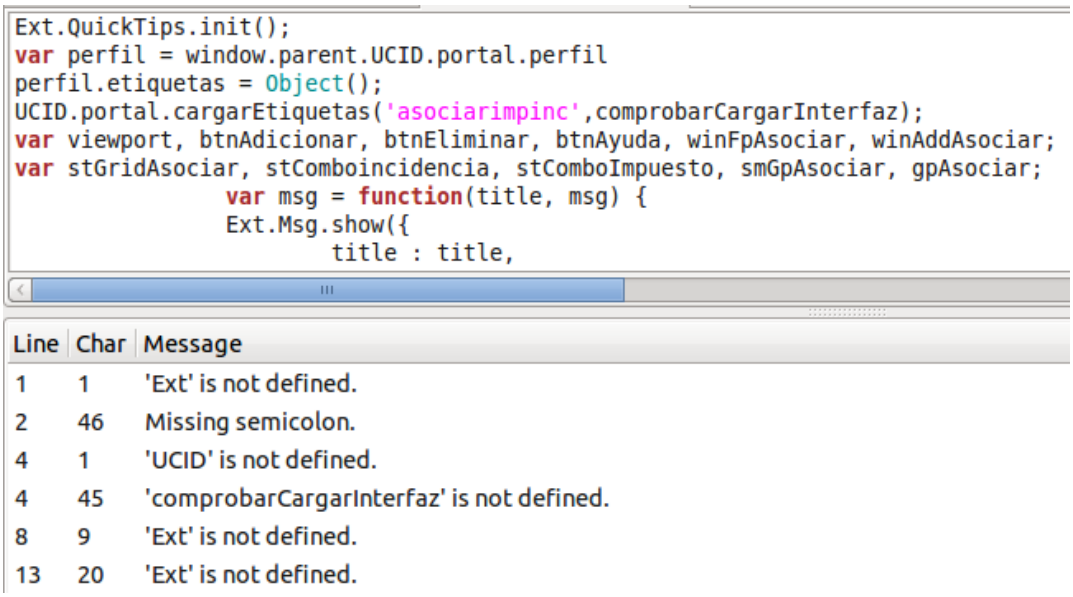
7- Contratos Vencidos

8- FileUploadField

9- Movimiento Fuerza de Trabajo

10- Movimiento de Nomina

Algunos ejemplos de errores encontrados por la herramienta JSHint en las clases probadas:



```
Ext.QuickTips.init();
var perfil = window.parent.UCID.portal.perfil
perfil.etiquetas = Object();
UCID.portal.cargarEtiquetas('asociarimpinc',comprobarCargarInterfaz);
var viewport, btnAdicionar, btnEliminar, btnAyuda, winFpAsociar, winAddAsociar;
var stGridAsociar, stComboincidencia, stComboImpuesto, smGpAsociar, gpAsociar;
                var msg = function(title, msg) {
                Ext.Msg.show({
                        title : title,
```

Line	Char	Message
1	1	'Ext' is not defined.
2	46	Missing semicolon.
4	1	'UCID' is not defined.
4	45	'comprobarCargarInterfaz' is not defined.
8	9	'Ext' is not defined.
13	20	'Ext' is not defined.

Figura 6 Ejemplo de errores detectados.

```
Ext.QuickTips.init();
var perfil = window.parent.UCID.portal.perfil
perfil.etiquetas = Object();
UCID.portal.cargarEtiquetas('gestimpuesto',cargarInterfaz);
var viewport, btnAdicionar, btnModificar, btnEliminar, btnImprimir, btnAyuda,
winModImpuesto, winImportar, btnImportar, btnExportar, fpImportar, stFormatoR
winReporte;
var stGridImpuesto, stComboIdioma, stComboTema, stComboEspecialidad, smGpImpu
var tfdenom,tfporciento;
```

Line	Char	Message
186	45	'rbTrabajador' is not defined.
192	58	'rbNomina' is not defined.
194	5	'rbNomina' is not defined.
194	41	'rbNomina' is not defined.
199	58	'rbTrabajador' is not defined.
228	26	'Ext' is not defined.

Figura 7 Ejemplo de errores detectados.

```
Ext.QuickTips.init();
var perfil = window.parent.UCID.portal.perfil;
perfil.etiquetas = Object();
UCID.portal.cargarEtiquetas('gestionarperiodopago',cargarInterfaz);
var btnAdicionar, btnModificar, btnEliminar, btnAyuda, winImportar, btnImportar,
fpImportar,windowAddier,windowAdd,windowAdd2,txPeriodoContable1er;
var stGdPeriodos, smGdPeriodos, gpGestpago, vpGestpago, stAnnos, stParte, myMask;
var fpAdicionarPeriodo, winAdd, winMod,winGen, fpGenerarPPago,stPeriodos1er;
var  stFormatoReporte, cmbFormatoReporte, reportForm, winReporte, stPeriodos;
var mensual, quincenal, cbannos,txffin,txfinicio,txffin2,txDenominacionMod,txDenc
```

Line	Char	Message
230	22	'Ext' is not defined.
242	5	'txffinMod' is not defined.
242	19	'Ext' is not defined.
249	25	Duplicate member 'name'.
253	44	'txfinicioMod' is not defined.
253	76	Missing semicolon.
257	21	'txffinMod' is not defined.
265	5	'stPeriodosMod' is not defined.

Figura 8 Ejemplo de errores detectados.

No hubo motivos para que las pruebas de caja blanca estática fueran abortadas ni detenidas, ya que las condiciones para que esto ocurra, no se presentaron en todo el proceso de pruebas realizado.

3.4 Evaluar pruebas de caja blanca estática

Para medir el grado de mantenibilidad del código fuente JavaScript, de los componentes revisados, se aplicaron métricas.

Teniendo en cuenta las NC detectadas por la herramienta JSHint las métricas aplicadas fueron:

- ✓ Registrabilidad de cambios.
- ✓ Facilidad de prueba.

Si el valor de X da entre 0 y 0.09 la estabilidad será alta y los cambios deficientes o pocos cambios.

Si el valor de X da entre 0.10 y 0.5 los cambios serán registrables y la estabilidad será media.

Si el valor de X da entre 0.6 y 1 los cambios serán registrables y la estabilidad será baja.

3.4.1 Métrica Registrabilidad de cambios

Tabla 8 Aplicación de la métrica registrabilidad de cambios para la clase 1.

Valores de A y B	X = A/B	Por tanto:
A = 2 **B = 25**	2/25 =0.08	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0 y 0.09, la estabilidad de la clase es alta ya que fueron efectuados pocos cambios.

Tabla 9 Aplicación de la métrica registrabilidad de cambios para la clase 2.

Valores de A y B	X = A/B	Por tanto:
A = 5 **B = 8**	5/8 =0.625	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0.5 y 1, la estabilidad de la clase es baja ya que fueron efectuados varios cambios.

Tabla 10 Aplicación de la métrica registrabilidad de cambios para la clase 3.

Valores de A y B	X = A/B	Por tanto:
A = 0 **B = 0**	0/0 =0	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0 y 0.09, la estabilidad de la clase es alta, ya que no se efectuaron cambios.

Tabla 11 Aplicación de la métrica registrabilidad de cambios para la clase 4.

Valores de A y B	X = A/B	Por tanto:
A = 0 **B = 0**	0/0 =0	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0 y 0.09, la estabilidad de la clase es alta, ya que no se efectuaron cambios.

Tabla 12 Aplicación de la métrica registrabilidad de cambios para la clase 5.

Valores de A y B	X = A/B	Por tanto:
A = 23 **B = 47**	23/47 =0.489	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0.10 y 0.5, la estabilidad de la clase es media ya que fueron efectuados varios cambios.

Tabla 13 Aplicación de la métrica registrabilidad de cambios para la clase 6.

Valores de A y B	X = A/B	Por tanto:
A = 2 **B = 17**	5/17 =0.294	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0.10 y 0.5, la estabilidad de la clase es media ya que fueron efectuados varios cambios.

Tabla 14 Aplicación de la métrica registrabilidad de cambios para la clase 7.

Valores de A y B	X = A/B	Por tanto:
A = 4 **B = 10**	4/10 =0.4	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0.10 y 0.5, la estabilidad de la clase es media ya que fueron efectuados varios cambios.

Tabla 15 Aplicación de la métrica registrabilidad de cambios para la clase 8.

Valores de A y B	X = A/B	Por tanto:
A = 0 **B = 1**	0/1 =0	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0 y 0.09, la estabilidad de la clase es alta, ya que no se efectuaron cambios.

Tabla 16 Aplicación de la métrica registrabilidad de cambios para la clase 9.

Valores de A y B	X = A/B	Por tanto:
A = 21 **B = 55**	21/55 =0.381	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0.10 y 0.5, la estabilidad de la clase es media ya que fueron efectuados varios cambios.

Tabla 17 Aplicación de la métrica registrabilidad de cambios para la clase 10.

Valores de A y B	X = A/B	Por tanto:
A = 12 **B = 38**	12/38 =0.315	Dado que el coeficiente de registrabilidad se encuentra en el rango entre 0.10 y 0.5, la estabilidad de la clase es media ya que fueron efectuados varios cambios.

Se obtuvo un total de 5 clases con estabilidad media, otras 4 con alta estabilidad y solo 1 con estabilidad baja.

De forma general la mayoría de las clases tienden a tener una estabilidad media, pero una vez que todos los cambios fueron realizados para corregir los errores encontrados, las clases han alcanzado una alta estabilidad.

3.4.2 Métrica facilidad de prueba

Aplicación de la métrica Facilidad de prueba o Examinabilidad

Se probaron un total de 10 clases.

Clase 1: Gestionar Pagos Adicionales.

T_1=0 N_1=0

Clase 2: Gestionar Impuestos.

T_2=13 min = 780 segundos N_2=27

Clase 3: Asociar Impuestos.

T_3=4 min = 240 segundos N_3=8

Clase 4: Gestionar Concepto de Pago.

T_4=0 N_4=0

Clase 5: Gestionar Periodo de Pagos.

T_5= 22 min = 1320 segundos N_5=47

Clase 6: Contrato

T_6= 7 min = 420 segundos N_6=17

Clase 7: Contratos Vencidos

T_7= 5 min = 300 segundos N_7=10

Clase 8: FileUploadField

T_8= 1 segundo N_8=1

Clase 9: Movimiento Fuerza de Trabajo

T_9= 25 min = 1500 segundos N_9=55

Clase 10: Movimiento Nomina

T_{10}= 17 min = 1020 segundos N_{10}=38

Para: Sum (T) / N

Siendo:

$T=T_1+T_{2+}T_3+T_4+T_5+T_6+T_7+T_8+T_9+T_{10}$ T=0+780+240+0+1320+420+300+1+1500+1020 T=5581

$N=N_1+N_2+N_3+N_4+N_5+N_6+N_7+N_8+N_9+N_{10}$ N=0+27+8+0+47+17+10+1+55+38 N=203

T/N = 5581/203 = 28 seg/fallos

Siendo 28 un número mayor que 0 y menor que 180 se determina que la conducta del sistema de software es correcta.

3.5 Conclusiones parciales

Mediante la puesta en práctica de la solución, se demostró que sí existen errores en el código fuente que a la vista del programador no son detectados, y los mismos pueden causar que llegue al cliente un software de calidad deficiente. Además se evidenció que no se desarrolla un software estandarizado regido por pautas de codificación, lo que puede traer como consecuencia que en posteriores fases de mantenimiento el costo y esfuerzo de realizarle una modificación, mejora o ingreso de nuevos requisitos al producto de software cueste mucho más que la propia implementación del mismo.

CONCLUSIONES

Con la investigación realizada se pudo llegar a la comprensión de las técnicas, herramientas y pruebas de caja blanca estática.

Se definió un proceso bien estructurado, el cual fue validado mediante su aplicación en un componente desarrollado bajo la tecnología Sauxe.

La herramienta utilizada demostró ser capaz de detectar errores en el código fuente JavaScript del componente revisado.

Se revisó que el código probado estuviera estandarizado para asegurar la legibilidad y entendibilidad del mismo, asegurando en cierta medida, que sea mantenible y de esta forma, fácil de modificar y de reutilizar.

Se aplicaron métricas para evaluar si el código probado era mantenible en cuanto a 2 sub-características de mantenibilidad.

De esta forma se cumplen los objetivos trazados y a medida que se lleve a cabo la evaluación estática del código fuente en los componentes que se desarrollen bajo la tecnología Sauxe, se estará incorporando en los mismos el atributo de mantenibilidad, el cual debe estar presente en todo sistema de software.

RECOMENDACIONES

Luego de haber cumplido con el objetivo general propuesto, se recomienda:

Continuar el estudio de herramientas que puedan ir surgiendo y sean mucho más potentes y mejores que la seleccionada, para analizar la estructura del código fuente JavaScript de la capa de presentación de los componentes desarrollados bajo la tecnología Sauxe.

Realizar un seguimiento de las actualizaciones que puedan ir surgiendo de la herramienta JSHint utilizada en la solución.

BIBLIOGRAFÍA REFERENCIADA

1. Portal de la Universidad de las Ciencias Informáticas. [Online] [Cited: Septiembre 25, 2012.] http://www.uci.cu/entorno-productivo.

2. Espinosa, Susana Gonzalez and Durán Cutiño, Dania. *Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías.* Universidad de las Ciencias Informáticas,La Habana,San Antonio : s.n., 2008.

3. Expósito, Raúl. *¿Qué es el Análisis Estático del Código ?* Madrid : s.n., 2009.

4. M.F.O.N.D Análisis estático del código. [Online] octubre 27, 2009. [Cited: enero 28, 2012.] http://3dgiordano.blogspot.com/2009/10/genexus-analisis-estatico-de-codigo.html.

5. Manakin. [Online] Septiembre 2008. [Cited: febrero 2, 2012.] http://repositorio.utp.edu.co/xml/bitstream/handle/123456789/542/111233326-331.pdf?sequence=1. 0122-1701.

6. ROGER S. PRESSMAN, 2002. Ingeniería del software. S.l.: McGrawHill. ISBN 8448132149.

7. Estrada, A F. *Calidad del software.* Cuba, Universidad de Matanzas "Camilo Cienfuegos" : s.n., 2006.

8. Ballesteros, J.M.M.M.J.F.H. La calidad del software y su medida. s.l. : Centro de estudio Ramón.

9. Lovelle, Juan Manuel Cueva. *Calidad del software.* Universidad de Oviedo:España : s.n., 1999.

10. Juristo, Natalia, Moreno, Ana M. and Vegas, Sira. TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 2006.

11. Ruiz, Francisco and Polo, Macario. Mantenimiento del Software. Castilla-La Mancha : s.n., 2006.

12. Javascript Lint. [Online] Septiembre 2007. [Cited: mayo 27, 2012.] http://www.javascriptlint.com/.

13. EtnasSoft. [Online] febrero 2011. [Cited: abril 15, 2012.] http://www.etnassoft.com/2011/02/19/jshint-herramienta-para-medir-la-calidad-del-codigo-javascript/.

14. The JavaScript Code Quality Tool. [Online] [Cited: marzo 13, 2012.] http://www.jslint.com/lint.html**.**

15. node-jslint Guithub. [Online] [Cited: febrero 25, 2012.] https://github.com/reid/node-jslint.

16. Alpízar, Yaimí Márquez and Valdés Hechavarría, Yenni. Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico. Universidad de las Ciencias Informáticas,La Habana, San Antonio : s.n., 2007.

17. Guerra, Yurién Ricardo Fuentes and Jordán Borjas, Ernesto. Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca. La Habana, San Antonio : s.n., 2008.

18. Morales, Dayanis Isaac. Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS. La Habana, San Antonio : Universidad de las Ciencias Informáticas, 2009.

19. Sicilia, Miguel-Angel. Conexxions. [Online] [Cited: abril 20, 2012.] http://cnx.org/content/m17457/latest/.

20. Pérez, Darling Darias. ANÁLISIS Y DISEÑO DE COMPONENTES PARA PRUEBAS DE CAJA BLANCA. San Antonio, La Habana : s.n., 2008.

21. Estrada, A F. Calidad del software. Cuba, Universidad de Matanzas "Camilo Cienfuegos" : s.n., 2006.

22. Myers, Glenford J. *The Art of Software Testing.* s.l. : Second edition, 1979.

23. Jacobson, Ivar and Rumbaugh, James. *Proceso Unificado de Desarrollo del Software.* Madrid : Addison Wesley, 2000. ISBN:84-7829-036-2.

24. Hetzel, Bill. *The Complete Guide to Software Testing .* 1998.

25. Lopéz, J . Oracle. *Fundamentos para el desarrollo de aplicaciones Web.* Buenos Aires -Argentina : MP Edisiones S.A .

26. Real Academia Española. [Online] [Cited: junio 18, 2012.] http://www.rae.es/rae.html.

BIBLIOGRAFÍA CONSULTADA

Portal de la Universidad de las Ciencias Informáticas. [Online] [Cited: Septiembre 25, 2012.] http://www.uci.cu/entorno-productivo.

Espinosa, Susana Gonzalez and Durán Cutiño, Dania. *Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías.* Universidad de las Ciencias Informáticas,La Habana,San Antonio : s.n., 2008.

Expósito, Raúl. *¿Qué es el Análisis Estático del Código ?* Madrid : s.n., 2009.

M.F.O.N.D Análisis estático del código. [Online] octubre 27, 2009. [Cited: enero 28, 2012.] http://3dgiordano.blogspot.com/2009/10/genexus-analisis-estatico-de-codigo.html.

Manakin. [Online] Septiembre 2008. [Cited: febrero 2, 2012.] http://repositorio.utp.edu.co/xml/bitstream/handle/123456789/542/111233326-331.pdf?sequence=1. 0122-1701.

ROGER S. PRESSMAN, 2002. Ingeniería del software. S.l.: McGrawHill. ISBN 8448132149.

Estrada, A F. *Calidad del software.* Cuba, Universidad de Matanzas "Camilo Cienfuegos" : s.n., 2006.

Ballesteros, J.M.M.M.J.F.H. La calidad del software y su medida. s.l. : Centro de estudio Ramón.

Lovelle, Juan Manuel Cueva. *Calidad del software.* Universidad de Oviedo:España : s.n., 1999.

Juristo, Natalia, Moreno, Ana M. and Vegas, Sira. TÉCNICAS DE EVALUACIÓN DE SOFTWARE. 2006.

Ruiz, Francisco and Polo, Macario. Mantenimiento del Software. Castilla-La Mancha : s.n., 2006.

Javascript Lint. [Online] Septiembre 2007. [Cited: mayo 27, 2012.] http://www.javascriptlint.com/.

EtnasSoft. [Online] febrero 2011. [Cited: abril 15, 2012.] http://www.etnassoft.com/2011/02/19/jshint-herramienta-para-medir-la-calidad-del-codigo-javascript/.

The JavaScript Code Quality Tool. [Online] [Cited: marzo 13, 2012.] http://www.jslint.com/lint.html**.**

node-jslint Guithub. [Online] [Cited: febrero 25, 2012.] https://github.com/reid/node-jslint.

Alpízar, Yaimí Márquez and Valdés Hechavarría, Yenni. Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico. Universidad de las Ciencias Informáticas,La Habana, San Antonio : s.n., 2007.

Guerra, Yurién Ricardo Fuentes and Jordán Borjas, Ernesto. Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca. La Habana, San Antonio : s.n., 2008.

Morales, Dayanis Isaac. Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS. La Habana, San Antonio : Universidad de las Ciencias Informáticas, 2009.

Sicilia, Miguel-Angel. Conexxions. [Online] [Cited: abril 20, 2012.] http://cnx.org/content/m17457/latest/.

Pérez, Darling Darias. ANÁLISIS Y DISEÑO DE COMPONENTES PARA PRUEBAS DE CAJA BLANCA. San Antonio, La Habana : s.n., 2008.

Estrada, A F. Calidad del software. Cuba, Universidad de Matanzas "Camilo Cienfuegos" : s.n., 2006.

Myers, Glenford J. *The Art of Software Testing.* s.l. : Second edition, 1979.

Jacobson, Ivar and Rumbaugh, James. *Proceso Unificado de Desarrollo del Software.* Madrid : Addison Wesley, 2000. ISBN:84-7829-036-2.

Hetzel, Bill. *The Complete Guide to Software Testing .* 1998.

Lopéz, J . Oracle. *Fundamentos para el desarrollo de aplicaciones Web.* Buenos Aires -Argentina : MP Edisiones S.A .

Real Academia Española. [Online] [Cited: junio 18, 2012.] http://www.rae.es/rae.html.

ANEXOS

Anexo 1

Encuesta para los desarrolladores

La siguiente encuesta es de carácter confidencial lo que usted ponga no será revelado a nadie, sea lo más sincero posible.

1. ¿Sabe lo que es un estándar de codificación?
2. ¿Sabe si el CEIGE posee algún estándar de codificación para implementar en JavaScript?
3. ¿Se rige por algún estándar de codificación para implementar en JavaScript?
4. ¿Alguien le revisa que lo que desarrolla cumpla con el estándar de codificación?
5. ¿Después de implementar revisa si ha violado alguna pauta del estándar de codificación?
6. ¿Cree que es necesario regirse por un estándar de codificación para implementar?

Si ____ ¿Por qué? No ____ ¿Por qué?

7. ¿Cree conveniente que le revisen si ha violado alguna pauta del estándar de codificación? ¿Por qué?

Anexo 2

Encuesta para detectar los roles de su proyecto.

¿Departamento al cual pertenece?

¿Cargo que ocupa en su proyecto?

¿Marque con una X cuál de estos roles existen en su proyecto?

Planificador ___

Desarrollador___

Especialista de Calidad___

Jefe de Línea de Desarrollo___

Administrador de Calidad___

Probador___

¿Existe algún otro rol a parte de los antes expuesto? ____Si ____No ¿Cuáles son? De existir otros roles ingresar su descripción.

¿Cree que su proyecto cuenta con todos los roles definidos en el modelo de desarrollo del ERP? ¿Por qué?

Anexo 3

Registro de No Conformidades

												Estado		
Fecha	**Turno**	**Probador**	**Elemento**	**Etapa de detección**	**No**	**No conformidad**	**Aspecto correspondiente**	**Tipo**	**Significativa**	**No Significativa**	**Recomendación**	**RA**	**PD**	**NP**

Anexo 4

Registro de Violaciones del estándar

Fecha	Probador	Cantidad de violaciones encontradas	Tipo	Nombre de la clase	Ubicación

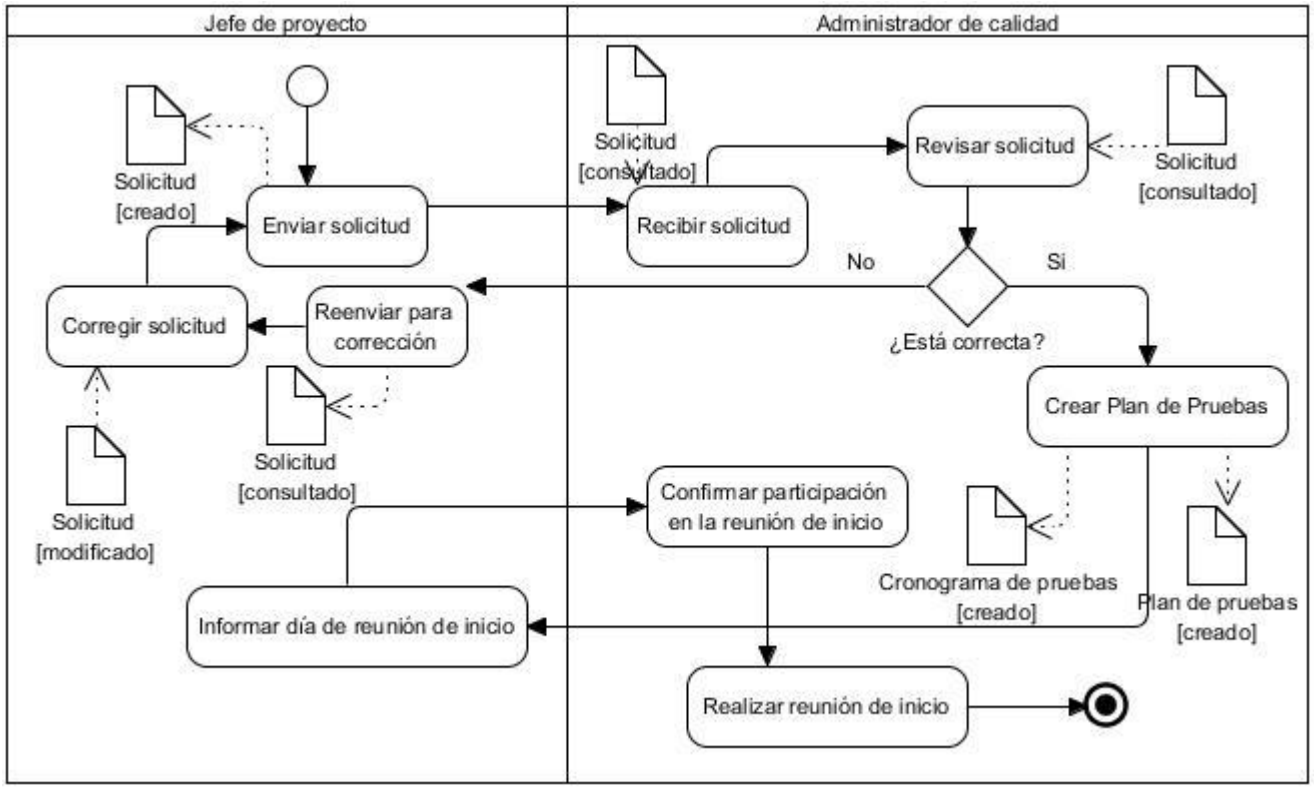
Anexo 5

Tabla comparativa de las herramientas

Herramienta	Licencia	Plataforma	Fiable	Lenguaje de programación	Facilidad de uso	Configurable
JavaScript Lint	GPL	Windows/Linux	poco	JavaScript	media	no
JSLint	MIT	Windows/Linux	no	JavaScript	alta	no
JSHint	MIT	Windows/Linux	si	JavaScript	alta	si

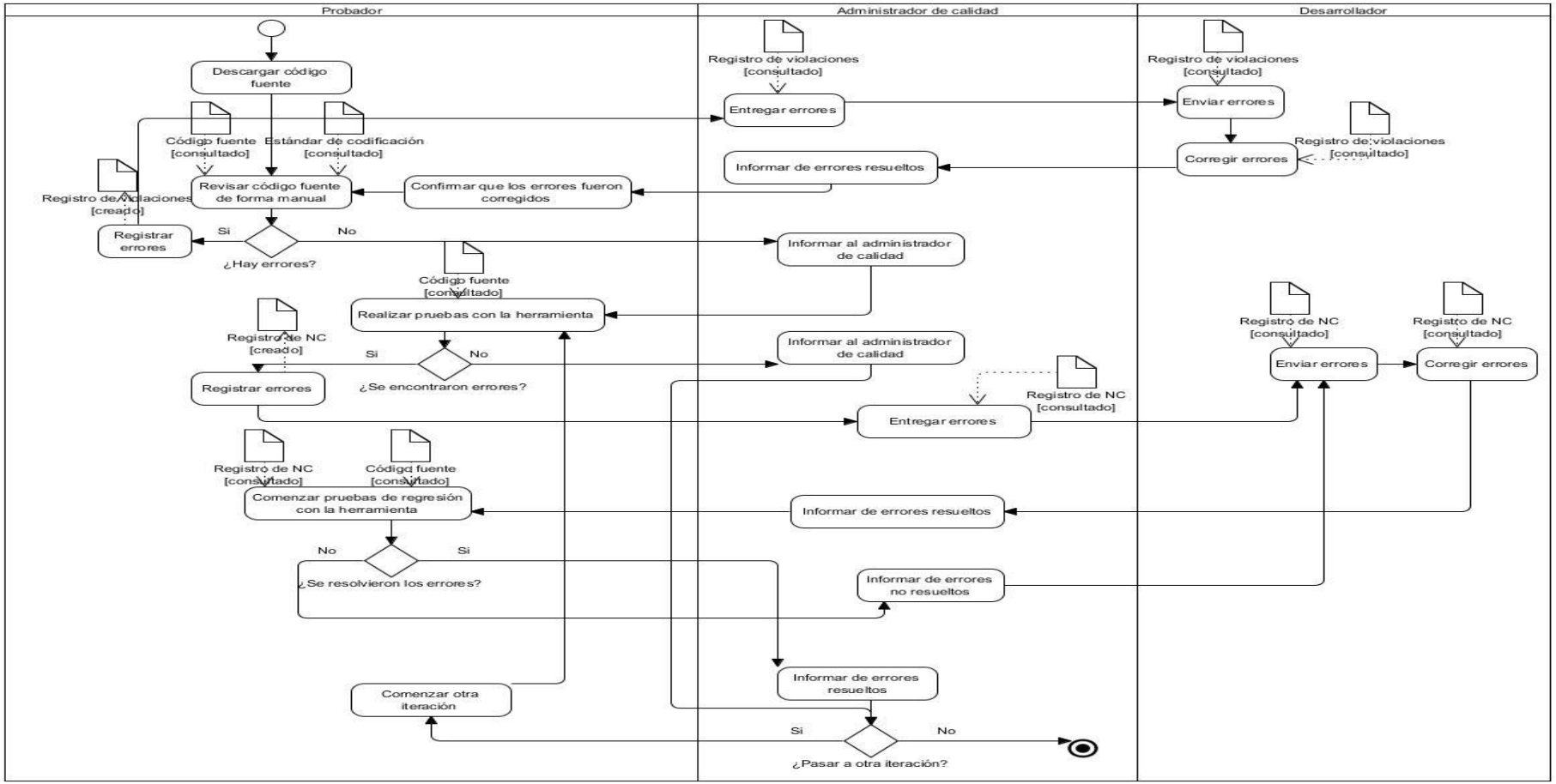
Anexo 6

Diagrama de actividades del subproceso planificación



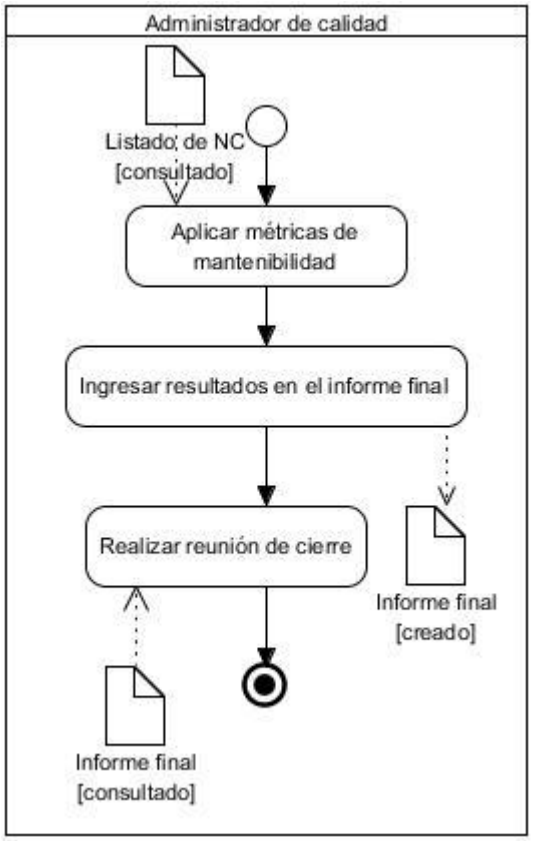
Anexo 7

Diagrama de actividades del subproceso ejecución



Anexo 8

Diagrama de actividades del subproceso evaluación



Anexo 9

Interfaz de la herramienta