Universidad de las Ciencias Informáticas

Facultad 3

"Evaluación estática del código para los proyectos desarrollados bajo tecnología Spring y lenguaje de programación Java en el CEIGE."

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Arlethy Betancourt Matos

Tutor (es): Ing. Giselle Almeida González

Ing. Iliannis Pupo Leyva

Co-tutor: Ing. Amirka Palacio Macía

La Habana, Junio 2012

"Año 54 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser el autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma el presente a los ____ días del mes de _____ del año _____.

_____

Firma del Autor

Arlethy Betancourt Matos

_____

Firma del Tutor

Ing. Iliannis Pupo Leyva

_____

Firma del Tutor

Ing. Giselle Almeida González

_____

Firma del Co-tutor

Ing. Amirka Palacio Macía


"El conocimiento humano
se basa en dos palabras:
confiar y esperar."
Alejandro Dumas

DEDICATORIA

A mi abuelita Hilda por cuidarme siempre... Me hubiese encantado que estuvieras aquí.

A mi mami Dora por ser mi ángel guardián y alumbrar mi camino en la oscuridad.

A mi hermano Héctor, para que vea en mí un espejo y no cometa los mismos errores. Yo siempre voy a ser tu apoyo y tu guía. Todo mi esfuerzo va dedicado a ti.

AGRADECIMIENTOS GENERALES

A Fidel y la Revolución por crear la universidad y darme la oportunidad de estudiar en ella.

A mis tutoras por ayudarme en todo lo que estaba en sus manos.

A las locas que vivieron conmigo por aguantarme en todos estos años.

A todos los que de una forma u otra me alentaron a seguir por sobre todas las dificultades.

AGRADECIMIENTOS PERSONALES

A mi mamá por confiar en mí y darme las fuerzas para seguir.

A mi hermano por quererme tanto y enseñarme a querer.

A mi papá por ser mi aliento, mi amigo, mi confidente y mi niño grande.

A mis tías y tío Efrén por confiar en mí y brindarme todo su apoyo.

A mi abuelita Nena. Te adoro.

A Yake y Libet por cuidar de mí y demostrarme que puedo ser una mejor persona.

A Patri y Puig por compartir sus conocimientos sin protestar.

A Hanny, mi peluquera y compañera de viajes.

A Sonia y Rosalina por brindarme sus consejos y alentarme a seguir cuando flaquee.

A mis amigos por estar siempre ahí y demostrarme cuán fuerte soy.

RESUMEN

En todo centro de desarrollo de software existe un equipo de calidad, el cual se encarga de garantizar que el producto final satisfaga las necesidades del cliente. Dentro del equipo de calidad los probadores son los encargados de realizar una serie de comprobaciones al software, que pueden ser tanto a las funcionalidades como al código en sí. Las pruebas que se realizan directamente sobre el código se denominan Pruebas de Caja Blanca y su función es verificar que la estructura interna de un componente funcione correctamente.

El presente trabajo de diploma tiene como objetivo evaluar el código mediante Pruebas de Caja Blanca Estática a los proyectos desarrollados bajo tecnología Spring y el lenguaje de programación Java que contribuya a elevar la mantenibilidad de los componentes desplegados en el Centro de Informatización de la Gestión de Entidades (CEIGE). Para ello se propone un procedimiento para realizar este tipo de pruebas a los componentes desarrollados bajo las tecnologías antes mencionadas.

La puesta en práctica de la propuesta de solución, implica que se examine la estructura interna del software, lo que disminuye el número de errores existentes en los sistemas y garantiza mayor calidad, confiabilidad y mantenibilidad. La utilización de herramientas de automatización agilizará el proceso de pruebas, por lo que se escogió el analizador Checkstyle en su integración con Eclipse.

Palabras claves: análisis estático, calidad, componente, mantenibilidad, prueba y verificación.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Actualmente el desarrollo de software juega un papel importante en la vida económica de cada país por lo que las grandes empresas productoras de software se empeñan en informatizar todo lo que le sea posible con su ingenio. En un principio solo preocupa la entrega en tiempo del producto al cliente, pero a medida que se avanza en el desarrollo de la solución la calidad va formando parte indisoluble de esta tarea, siendo imprescindible que el producto satisfaga las necesidades del cliente. Para ello se han creado instituciones encargadas de documentar todo sobre el proceso de controlar la calidad, publicando normas y estándares que rigen el proceso de desarrollo de software para que el producto final adquiera la eficacia requerida sin que su producción traiga consigo costos y tiempo innecesarios.

La Universidad de las Ciencias Informáticas (UCI) espera convertirse en una Industria de Software de alto prestigio a nivel nacional e internacional. Para ello cuenta con el apoyo del Centro de Calidad para Soluciones Informáticas (Calisoft), siendo este el encargado de liberar los productos software a nivel de país.

Con motivo de organizar la actividad productiva en la universidad se crearon centros de desarrollo de software, ejemplo de ello es el Centro de Informatización de la Gestión de Entidades (CEIGE), que cuenta con un Grupo de Calidad encargado de verificar el perfecto funcionamiento de las aplicaciones que se desarrollan en los distintos departamentos del centro. Actualmente en dicho laboratorio no se realizan pruebas de caja blanca (prueba que tienen en cuenta el mecanismo interno de un sistema o componente de software), cuestión que está provista en las siguientes particularidades:

- No se asegura la legibilidad del código entre los distintos programadores lo cual no facilita el debugueo del mismo.
- No se provee de un código claro y bien documentado lo cual compromete la reusabilidad y el mantenimiento del mismo.
- No se facilita la portabilidad entre plataformas y aplicaciones lo que dificulta la migración hacia tecnologías libres.
- No se utilizan las decisiones en su parte verdadera y en su parte falsa provocando que el software colapse bajo situaciones extremas.

- No se ejecutan todos los bucles en sus límites lo que puede ocasionar demora en el tiempo de respuesta del sistema bajo situaciones extremas.
- No se utilizan todas las estructuras de datos internas ocasionando incoherencias en la estandarización del código.

Se considera a la prueba de caja blanca como uno de los tipos de pruebas más importantes que se le aplican al software, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (Pressman, 2000)

Calisoft tampoco realiza este tipo de pruebas debido a que “los artefactos llegan al Departamento de Pruebas cuando están finalizados y listos para ser entregados al cliente. Realizar este tipo de pruebas en esta etapa, con el rigor que lleva, podría complejizar mucho la liberación de los productos y realmente sería muy trabajoso por la cantidad de líneas de código que poseen”. (Tayche Capote, 2011)

No realizar este tipo de pruebas trae consigo, que el código no alcance 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 extrema.

Obtener un software con calidad implica utilizar metodologías o procedimientos en todo el ciclo de desarrollo del mismo, que permitan uniformar la filosofía de trabajo, para lograr mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad. La mantenibilidad es la capacidad del producto de software de ser modificado. (ISO, 9126-1)

Actualmente con la nueva tendencia hacia las metodologías de desarrollo ágiles (pruebas automatizadas) sobre las vías tradicionales, es de vital importancia la utilización de pruebas de caja blanca para de esta forma disminuir en un gran porciento el número de errores existentes en los sistemas y a su vez garantizar una mayor calidad y confiabilidad en el producto. Para garantizar que el producto final posea la menor cantidad de errores posibles es necesario desarrollar un correcto procedimiento de Pruebas de Caja Blanca Estática.

Por lo antes planteado surge el siguiente **Problema a resolver**: En los proyectos del CEIGE desarrollados bajo tecnología Spring y el lenguaje de programación Java las dificultades existentes con el rendimiento, la complejidad y la estandarización del código afectan la mantenibilidad de los componentes.

El **Objeto de estudio** está centrado en las Pruebas de Software de Caja Blanca en el CEIGE.

El **Campo de acción** se enmarca en las pruebas de software de Caja Blanca Estática en los sistemas desarrollados bajo la tecnología Spring y el lenguaje de programación Java en el CEIGE.

El **Objetivo general**: Evaluar el código mediante pruebas de Caja Blanca Estática a los proyectos desarrollados bajo tecnología Spring y el lenguaje de programación Java en el CEIGE para elevar la mantenibilidad de sus componentes.

La **Idea a defender** que se plantea es la siguiente: Al evaluar la mantenibilidad de los componentes desarrollados en el CEIGE bajo la tecnología Spring y el lenguaje de programación Java se podrá detectar el grado de mantenibilidad de los mismos en la fase de desarrollo.

Objetivos específicos:

- Elaborar el marco teórico referencial de la investigación.
- Describir actividades, artefactos y herramientas para aplicar Pruebas de Caja Blanca Estática a los componentes desarrollados en el CEIGE bajo la tecnología Spring y el lenguaje de programación Java.
- Validar la propuesta de solución a través de su aplicación en uno de los productos del CEIGE.

Posibles resultados:

- Evaluación de la mantenibilidad de los componentes desarrollados bajo la tecnología Spring y el lenguaje de programación Java mediante pruebas de Caja Blanca Estática en el CEIGE.
- Grado de mantenibilidad de los componentes desarrollados bajo la tecnología Spring y el lenguaje de programación Java mediante las Pruebas de Caja Blanca Estática en el CEIGE.

El presente trabajo de diploma tiene una estructura en tres capítulos.

Capítulo 1: Fundamentación Teórica

En este capítulo se realiza un análisis de los conceptos y elementos de gran utilidad para la aplicación de las Pruebas de Caja Blanca Estática en los componentes desarrollados bajo la tecnología Spring y el lenguaje de programación Java. Además se hace un estudio de algunas de las herramientas existentes para realizar análisis estático del código fuente, de las cuales se escogerá una para validar la solución.

Capítulo 2: Propuesta de Solución

En este capítulo se explican los métodos, las técnicas y las herramientas que se van a emplear y se lanza la propuesta de solución para evaluar la mantenibilidad de los componentes, desarrollados bajo la tecnología Spring y el lenguaje de programación Java, mediante pruebas de Caja Blanca Estática.

Capítulo 3: Validación de la Propuesta de Solución

En este capítulo se valida la propuesta de solución a partir de la aplicación de pruebas de Caja Blanca Estática a uno de los componentes, desarrollados en el proyecto Sistema Automatizado de la Gestión Bancaria (SAGEB) bajo la tecnología Spring y el lenguaje de programación Java, evaluando la mantenibilidad de los mismos mediante métricas.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se hace un análisis de conceptos y elementos de gran utilidad para elaborar un procedimiento para la aplicación de las Pruebas de Caja Blanca Estática en los componentes desarrollados dentro del Centro de Informatización de la Gestión de Entidades en el proyecto Sistema Automatizado de la Gestión Bancaria, con el objetivo de evaluar la mantenibilidad del código fuente escrito en el lenguaje de programación Java.

Calidad de software

La calidad es un "conjunto de características de un producto o servicio que le confieren su aptitud para satisfacer las necesidades expresadas e implícitas" (ISO 8402-UNE 66-001-92)

La calidad del software es el "grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario." (IEEE, Std. 610-1990)

Es "la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente". (Pressman, 2005)

Factores de calidad del software (Hewlett-Packard)

- Funcionalidad: Se valora evaluando el conjunto de características y capacidades del programa, la generalidad de las funciones entradas y la seguridad del sistema global.
- Usabilidad: Se valora evaluando el conjunto de características y capacidades del programa, la generalidad de las funciones entregadas y la seguridad del sistema global.
- Fiabilidad: Se evalúa midiendo la frecuencia y gravedad de los fallos, la exactitud de las salidas, el tiempo medio de fallos, la capacidad de recuperación de un fallo y la capacidad de predicción del programa.
- Rendimiento: Se mide por la velocidad del procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia.

- Capacidad de Soporte: Combina la capacidad de ampliar el programa, adaptabilidad y servicios, así como la capacidad para hacer pruebas, compatibilidad, capacidad de configuración del software, la facilidad de instalación de un sistema y la facilidad con que se pueden localizar los programas.

Factores que afectan a la calidad del software

- Corrección: Hasta dónde satisface un programa su especificación y logra los objetivos propuestos por el cliente.
- Fiabilidad: Hasta dónde se puede esperar que un programa lleve a cabo su función con la exactitud requerida.
- Eficiencia: La cantidad de recursos informáticos y de códigos necesarios para que un programa realice su función.
- Integridad: Hasta dónde se puede controlar el acceso al software o a los datos por personas no autorizadas.
- Usabilidad: El esfuerzo necesario para aprender a operar con el sistema, preparar los datos de entrada e interpretar las salidas (resultados) de un programa.
- Facilidad de mantenimiento (mantenibilidad): El esfuerzo necesario para localizar y arreglar un error en un programa.
- Flexibilidad: El esfuerzo necesario para modificar un programa que ya está en funcionamiento.
- Facilidad de prueba: El esfuerzo necesario para probar un programa y asegurarse de que realiza correctamente su función.
- Portabilidad: El esfuerzo necesario para transferir el programa de un entorno hardware/software a otro entorno diferente.
- Reusabilidad: Hasta dónde se puede volver a emplear un programa en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones que realiza el programa.
- Interoperatividad: El esfuerzo necesario para acoplar un sistema con otro. (McCall)

Por lo antes analizado se concluye que la calidad del software consiste en desarrollar productos que, cumpliendo las normas, satisfagan las necesidades del usuario, los requisitos que rigieron su diseño y que tiendan a cero defectos. Los factores que afectan en la calidad del software pueden usarse para establecer métricas que permitan medir la calidad durante todo el proceso de desarrollo de software.

Estándar de codificación

Un estándar de codificación son reglas que se siguen para la escritura del código fuente; de tal manera que a otros programadores se les facilite entender el código ya realizado. Un estándar de codificación completo comprende todos los aspectos de la generación de código.

En un proyecto se definen estándares de codificación porque conservar un mismo estilo de programación permite que todos los involucrados lo puedan entender en menos tiempo y que a su vez el código sea mantenible. La idea es que parezca como si un mismo programador hubiera escrito todo el código de una sola vez.

¿Para qué se necesita un estándar de codificación? (Miguel Matas, 2008)

- Eleva la mantenibilidad del código.
- Sirve como punto de referencia para los desarrolladores.
- Mantiene un estilo de programación.
- Ayuda a mejorar el proceso de codificación, haciéndolo más eficiente.

¿Cómo realizar un buen código?

Un buen código es aquel que funciona sin fallos, además debe ser legible y mantenible, se debe ajustar a los estándares de la organización para que todos los desarrolladores del sistema manejen y entiendan las mismas herramientas y mecanismos en la codificación.

A continuación se definen reglas que las organizaciones dedicadas al desarrollo de software consideran importantes, para evitar problemas, en la mayoría de los lenguajes de programación más utilizados. (Raúl Expósito)

- Minimizar el uso de variables globales.
- Nombres descriptivos de funciones, variables, módulos, objetos y métodos.

- Minimizar el tamaño de funciones, métodos y procedimientos. No exceder las 50 líneas.
- Las descripciones deben ser cortas y claras para no confundir la lectura del código.
- Organización de los métodos; una buena disposición del código hará que futuros cambios sean posibles.
- Uso generoso de espacios en blanco.
- Es importante que cada línea de código no supere los 70 caracteres.
- Escribir una sentencia por línea.
- Es fundamental que el grupo de desarrolladores respete el mismo estilo de codificación.
- Toda aplicación debe ser documentada sin importar lo pequeña que sea.

Ventajas de un estándar de codificación

- Asegurar la legibilidad del código entre distintos programadores, facilitando el debugueo del mismo.
- Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones. (Raúl Expósito)

La utilización de estándares de codificación en un proyecto es un factor fundamental para garantizar que el producto final sea mantenible, además de asegurar total calidad en la escritura del mismo. Tener en cuenta las reglas para escribir código fuente, independientemente del lenguaje de programación, conserva las buenas prácticas en el proceso de desarrollo de software.

Revisiones de código y estándares de codificación

Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurar que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previamente implementado, o cuando se realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debe establecer cómo maniobrar con el código ya existente, porque el mejor método para asegurar que un equipo de programadores mantenga un

código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica continuamente un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas y se revisa el código periódicamente, existen muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener. Aunque el propósito principal para llevar a cabo revisiones del código a lo largo de todo el desarrollo es localizar defectos en el mismo. Las revisiones también pueden afianzar los estándares de codificación de manera uniforme. La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez inicializado el trabajo.

Mantenibilidad

La mantenibilidad es "la rapidez con la cual las fallas, o el funcionamiento defectuoso en los equipos son diagnosticados y corregidos, o el mantenimiento programado es ejecutado con éxito." (Dounce, 2000)

Otra definición de la mamtenibilidad es: “la facilidad con que un sistema o componente de software puede ser modificado para corregir defectos, mejorar el rendimiento u otros atributos, o adaptarse a un cambio de entorno”. (IEEE, 1990)

Esta definición está directamente conectada con la definición de mantenimiento de software que no es más que “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”. (IEEE, 1990)

En consecuencia, la mantenibilidad es una característica de calidad del software relacionada con la facilidad de mantenimiento, por lo que se considera como una actividad de mantenimiento. A mayor mantenibilidad, menores costes de mantenimiento y viceversa.

Categorías de la mantenibilidad (Ghezzi)

- Correctiva: concerniente a remover pequeñas fallas remanentes después del testeo.
- Adaptativa: concerniente al cambio del producto necesario por el cambio de sus requerimientos.

- Perfectiva: busca sólo mejorar los algoritmos usados para hacerlos más eficientes.

Métricas de mantenibilidad

- Analizabilidad: atributos que miden esfuerzo necesario para el diagnóstico de deficiencias o causas de fallas, o para identificación de las partes que deben ser modificadas.
- Cambiabilidad. Facilidad para el cambio: atributos que miden el esfuerzo necesario para realizar modificaciones, eliminación de fallas o cambios en el contexto.
- Estabilidad: atributos que se relacionan con el riesgo de efectos no esperados en las modificaciones.
- Examinabilidad. Facilidad de prueba: atributos que miden el esfuerzo necesario para validar el software modificado.
- Cumplimiento. Conformidad de la mantenibilidad: atributos que hacen que el software adhiera a estándares relacionados con la aplicación, y convenciones o regulaciones legales. (ISO 9126-3)

Las métricas escogidas para evaluar el procedimiento son: Examinabilidad y Cambiabilidad por ser las que más se ajustan a la etapa de desarrollo en que se encuentra el subsistema a probar y las más fáciles de interpretar.

Factores que influyen en la mantenibilidad

- 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.

- Incorporación en el sistema de facilidades de depuración.
- Disponibilidad del equipo (computador y periféricos) adecuado para realizar el mantenimiento.
- Disponibilidad de la persona o grupo que desarrolló originalmente el software.
- Planificación del mantenimiento. (Kopetz, 1979)

Cuando se realiza el análisis estático inciden directamente en la mantenibilidad de los componentes los factores resaltados anteriormente.

Propiedades de la mantenibilidad (Ruiz, F. y Polo, M., 2012)

La mantenibilidad se puede considerar como la combinación de dos propiedades diferentes: Reparabilidad y Flexibilidad.

- Reparabilidad

Un sistema software es reparable si permite la corrección de sus defectos con una cantidad de trabajo limitada y razonable. La reparabilidad se ve afectada por la cantidad y el 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 de módulos no implica un producto más reparable, ya que también aumenta la complejidad de las interconexiones entre módulos. 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. La reparabilidad de un producto software está influida por su fiabilidad, ya que al incrementarse esta última disminuye la necesidad de reparaciones.

- Flexibilidad

Un sistema software es flexible si permite cambios que satisfagan los nuevos requerimientos, es decir, si puede evolucionar. Por naturaleza inmaterial, el software es mucho más fácil de cambiar o incrementar en lo que respecta a sus funciones que otros productos de naturaleza física (equipos hardware), pero esta flexibilidad se ve disminuida con cada nueva versión, ya que cada versión complica la estructura del software y, por tanto, las futuras modificaciones serán más difíciles.

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. En términos de estos últimos, los procesos deben poderse acomodar a nuevas técnicas de gestión y organización, a cambios en la forma de entender la ingeniería, etc.

Importancia de determinar la mantenibilidad antes y después de terminado el producto

El software no se deteriora con el uso ni con el paso del tiempo, a diferencia de los materiales mecánicos que son producto de otras actividades de ingeniería; es decir, no sufre de un deterioro físico. No obstante, se suele considerar que el software tiene un deterioro en su estructura, cuando a lo largo del tiempo se van incluyendo cuantiosos cambios que hacen que su estructura interna sea cada vez más difícil de entender. Esta idea del deterioro de la estructura da lugar a la idea relacionada con la mantenibilidad. “La mantenibilidad es una propiedad del diseño del software relativa a la facilidad de mantenimiento.” (ISO 9126-3). Esto lleva a que en ocasiones se introduzcan cambios en el software sólo para hacerlo más mantenible.

En un sistema que es fácil de mantener se puede implementar un cambio con un menor esfuerzo que en un sistema que es menos mantenible. Puede ser rentable hacer software mantenible si se producen cambios frecuentemente en el código, ya que el software evoluciona para adaptarse a las necesidades de los usuarios, por tanto, es de suma importancia garantizar la facilidad de mantenimiento antes de entregar el producto final.

Dependiendo de cómo se haya construido el software se puede aumentar la mantenibilidad. Los desarrolladores, por lo general, no producen un código claro ni fácil de comprender, por lo que proporcionar mantenimiento en esas condiciones resulta más trabajoso, por tal motivo se hace necesario realizar el análisis estático al código, determinando así cuán mantenible es el producto final. 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 produzcan, mucho más fáciles de llevar a cabo.

Prueba

“La prueba es un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática.” (Pressman, 1998)

Las pruebas deben centrarse en dos objetivos:

- Probar si el software no hace lo que debe hacer.

- Probar si el software hace lo que no debe hacer, es decir, si provoca efectos secundarios adversos.

¿Por qué probar?

Las fallas de software ocasionan graves pérdidas económicas; éstos son 100 a 1000 veces más costosos de encontrar y reparar después de la construcción.

Evitar plazos y presupuestos incumplidos, insatisfacción del usuario, escasa productividad y mala calidad en el software producido y finalmente la pérdida de clientes.

Automatizar el proceso de pruebas consigue reducciones de hasta un 75% en el costo de la fase de mantenimiento.

¿Quiénes deben de probar o qué se debe probar?

El desarrollador debe evitar probar sus propios programas, ya que desea demostrar que funcionan sin problemas de manera consciente o inconsciente, además de que:

- Se debe inspeccionar minuciosamente el resultado de cada prueba para poder descubrir posibles síntomas de defectos.
- Se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.
- No se deben hacer planes de prueba bajo la suposición de que, prácticamente, no hay defectos en los programas y, por tanto, dedicar pocos recursos a las pruebas.

Por tal motivo la persona capacitada para realizar esta tarea es el Probador, rol capacitado en un proyecto para ejecutar las pruebas y registrar los resultados de las mismas, además de no interactuar directamente con el código durante la etapa de desarrollo del mismo.

Prueba de Software

En la actualidad una de las actividades fundamentales en el proceso de desarrollo de un software son las pruebas. Estas se realizan dirigidas a componentes del software o al software en general, de manera que su resultado final es la medición del grado de cumplimiento del software con los requerimientos previamente definidos.

Las pruebas de software se pueden definir como el “proceso de validación y verificación de un programa de software, aplicación o producto que cumple con los requisitos que guiaron su diseño y desarrollo, funciona como se espera y se puede implementar con las mismas características.” (Pressman, 1998)

Las pruebas de software no son más que la ejecución de un programa con la intención de descubrir errores. Es un elemento crítico para la garantía de la calidad del mismo y representa una revisión final de las especificaciones, el diseño y la codificación. En resumen las pruebas de software no es más que una técnica experimental para la búsqueda de errores en los programas.

Estrategia de Pruebas

- Prueba de Unidad: se buscan errores en los componentes más pequeños del programa (módulos). Estos errores se detectan cuando dichos componentes no actúan como se ha especificado en el diseño detallado.
- Prueba de Integración: se prueban los distintos componentes que constituyen el software. Esta prueba está orientada a detectar fallos provocados por una incorrecta comunicación entre módulos.
- Prueba de Sistema: es la que se encarga de buscar errores en el ensamblaje software/hardware.
- Prueba de Regresión: se centra en la búsqueda de defectos después que se ha producido un importante cambio en el código.
- Finalmente, el usuario ha de realizar la Prueba de Aceptación final sobre el sistema completo.

En resumen las pruebas son un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática. Son el último eslabón en la evaluación de la calidad y el descubrimiento de errores. Pero no deben considerarse como una red de seguridad ya que no es posible probar la calidad. Si no está presente antes de que se empiece a probar, no estará cuando se culmine. La calidad debe hallarse en el software en todo el proceso de ingeniería.

Prueba de Caja Blanca

“Es la prueba que tenga en cuenta el mecanismo interno de un sistema o componente” (IEEE, 1990). Las pruebas de caja blanca también se conocen como pruebas estructurales, pruebas de caja clara, y pruebas de caja de cristal. (Beizer, 1995).

Las connotaciones de "caja transparente" y de “caja de cristal" de manera apropiada indican que tienen una visibilidad completa del funcionamiento interno del producto de software, en concreto, la lógica y la estructura del código. Consiste en realizar pruebas para verificar qué líneas específicas en el código funcionan tal como está definido.

Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Para esta prueba se consideran tres importantes puntos:

- Conocer el desarrollo interno del programa, determinante en el análisis de coherencia y consistencia del código.
- Considerar las reglas predefinidas por cada algoritmo.
- Comparar el desarrollo del programa en su código con la documentación pertinente.

Beneficios de las pruebas de caja blanca

- Facilitan la vida del desarrollador: La calidad del código mejora, se reducen los tiempos de depuración y la corrección de incidencias y por tanto el cliente está mucho más contento porque la aplicación hace lo que él quiere que haga, por lo que ha pagado.
- Fomentan el cambio y la refactorización: Si se considera que el código es mejorable se puede cambiar sin ningún problema. Si el cambio no estuviera realizado correctamente las pruebas avisarán de ello.
- Reducen drásticamente los problemas y tiempos dedicados a la integración: En las pruebas se simulan las dependencias, lo que permite que se pueda probar el código sin disponer del resto de los módulos.

- Ayudan a entender mejor el código: Sirven de documentación y a través de las pruebas se pueden comprender mejor qué hace un módulo y qué se espera de él.
- Permiten probar o depurar un módulo sin necesidad de disponer del sistema completo: En algunas situaciones montar un entorno para poder probar una incidencia es más costoso que corregir la incidencia propiamente dicha. Si se parte de la prueba de caja blanca se puede centrar toda la atención en corregir el error de una forma más rápida y lógicamente, asegurando posteriormente que todo funciona según lo esperado. (Raúl Expósito)

En resumen se denominan cajas blancas a un tipo de pruebas de software que se realiza sobre las funciones internas de un módulo. Se clasifican en estáticas y dinámicas; las primeras permiten realizar un análisis de errores sin ejecutar el código, pueden realizarse desde las primeras fases del ciclo de vida del software y están enfocadas a la validación y optimización del código. Las segundas se utilizan en las últimas fases del ciclo de vida. En conjunto tratan de prevenir errores, lo que resulta fundamental en la obtención de un producto con mayor calidad.

Prueba de Caja Blanca Estática

“Es el proceso que cuidadosa y metódicamente revisa el diseño del software, la arquitectura o el código para encontrar defectos sin necesidad de ejecutar el código.” (Patton, 2005)

Permiten realizar un análisis de fallos sin ejecutar el código y se pueden utilizar desde las primeras fases del ciclo de vida del software. Están enfocadas a optimizar el código y a su validación. Esta es una prueba que se realiza en el código fuente sin ejecutarla, por lo que se llaman "estáticas".

Clasificación del Análisis Estático

Manual: es realizado por una persona.

- Inspección: Determina si el código está completo y correcto, como también las especificaciones.
- Recorrido o Walkthrough: Interrelación informal entre probadores, creadores y usuarios del sistema.

Automático: es realizado por un programa de ordenador sobre el código.

- Verificación estática: Compara los valores generados por el programa con los rangos de valores predefinidos haciendo una descripción del funcionamiento de los procedimientos en términos booleanos (verdadero o falso) determinando los puntos de falla.
- Ejecución simbólica: Hace un seguimiento de la comunicación entre funciones, módulos, aplicaciones, luego de que todas las partes hayan sido verificadas por separado.

Cada uno de estos análisis persigue unos objetivos concretos:

El análisis realizado por un programa de ordenador (automático) reduce la complejidad que supone detectar problemas en la base del código ya que lo busca utilizando reglas predefinidas.

El análisis realizado por una persona (manual) se centra en porciones específicas de la aplicación como, por ejemplo, determinar si las librerías se están utilizando debidamente o si la arquitectura del software es la correcta.

Ambos análisis se complementan. El automático se centra únicamente en facetas de más bajo nivel como la sintaxis y la semántica del código, funcionando en cualquier tipo de aplicación, mientras que el manual se ocupa de facetas de alto nivel como la estructura de la aplicación o la manera de trabajar con otros elementos externos como las librerías. La unión de ambos permitirá identificar los problemas potenciales a distintos niveles que generen complicaciones técnicos en el producto. Por tanto se deben unificar para poder mejorar la base del código fuente, lo cual repercutirá en una mejora tanto del desarrollo como del mantenimiento del software antes incluso de llegar a ejecutarse.

Ventajas del análisis estático (Raúl Expósito)

En base al código fuente, se puede obtener información que permita mejorar la base del código manteniendo la semántica original.

En cuanto a la ganancia en facilidad de mantenimiento y desarrollo, el objetivo es minimizar los problemas técnicos de los proyectos.

Las funciones de los analizadores consisten en encontrar parte de código que pueda:

- Reducir el rendimiento.
- Provocar errores en el software.

- Complicar el flujo de datos.
- Tener una excesiva complejidad.
- Suponer un problema en la seguridad.

Limitaciones del análisis estático (Raúl Expósito)

No permite saber si el software va hacer lo que se espera de él o no. Se puede analizar el código fuente y saber cómo mejorarlo, pero no se sabe si hace lo que tiene que hacer u otra cosa totalmente distinta e inesperada.

Al utilizar analizadores automáticos pueden devolver falsos positivos. Es posible que el analizador detecte como error algo que se sabe no está bien y que por alguna buena razón está programado así.

Están limitados al análisis de código sin llegar a ejecutarlo, de tal modo que se necesita complementarlo con tests o con profiling para poder realizar mejoras en un ámbito mayor.

Los analizadores estáticos están muy ligados a lenguajes de programación concretos, e incluso entre lenguajes hay diferencias, ya que en lenguajes estáticos como Java es más sencillo hacer un análisis estático que en otros lenguajes dinámicos como Groovy.

Tipologías del Análisis Estático

- **Análisis basado en AST (Árbol de Sintaxis Abstracta).** Transformación del código en una representación en árbol que refleja la estructura del fichero.
- **Análisis style-checkers:** Chequeo de estilo. Se centran exclusivamente en la estructura léxica y sintáctica. Con este tipo de analizadores se detectan espacios en blanco, código demasiado extenso, nombres de variables que no siguen las convenciones indicadas, líneas de código mal indentadas.
- **Análisis bug-checker:** Chequeo de errores. Método basado en la búsqueda de patrones o reglas predefinidos por la herramienta o diseñados por el usuario de la misma.
- **Theorem proving:** Demostración del teorema. Construye una prueba de los requerimientos mediante inducción lógica sobre la estructura del programa.

- **Análisis dataflow.** Flujo de datos. Es la técnica más utilizada en el análisis estático. Un esquema de análisis de flujos de datos define un valor en cada punto en el programa. Las instrucciones del programa tienen funciones de transferencia asociadas que relacionan el valor antes de la instrucción con el valor después de la misma.

  Dentro de este tipo de análisis se encuentran varias categorías:

  - **Análisis intraprocedural vs interprocedural:** Un análisis intraprocedural opera a través de un método o procedimiento, mientras que un análisis interprocedural opera en todo el programa entre diferentes métodos o procedimientos.
  - **Análisis context-sensitive vs context-insensitive:** En este caso lo que se contabiliza es el número de llamadas a un procedimiento, método o función. Un análisis context-sensitive distingue varias llamadas un mismo método, sin embargo, un análisis context-insensitive contaría una única llamada a ese mismo método.
  - **Análisis path-sensitive vs path-insensitive:** Los analizadores path-sensitive tienen en cuenta distintas alternativas que se pueden tener en una sentencia de decisión o bifurcación, mientras que los analizadores path-insensitive no tendrían en cuenta las dos alternativas (x no es constante). Evidentemente, los primeros resultan más caros de implementar que los segundos.
  - **Backward dataflow analysis vs Forward dataflow analysis:** Estas técnicas se utilizan cuando es necesario investigar el flujo de datos en dos sentidos (pasado y futuro). La técnica forward analiza el código sobre el comportamiento pasado, alcance de las definiciones o expresiones definidas, mientras que en la técnica backward el análisis intenta predecir el comportamiento futuro, expresiones muy ocupadas o variables sin vida. Se pueden utilizar para descubrir redundancias en el código.

Herramientas de análisis estático

Se pueden clasificar las herramientas según los servicios que ofrece y/o la tarea a la que da soporte. Las herramientas de análisis estático están contenidas en el grupo de las de verificación y validación, las mismas se clasifican en:

- Análisis de consistencia.

- Detección de código no usado.
- Grafo de flujo de llamadas.
- Referencias cruzadas.
- Diagramas de estructura (dependencias entre módulos).
- Comprobador de normas.

En resumen las pruebas de caja blanca estática se pueden definir como un proceso que implica un examen metódico y cuidadoso de la arquitectura del software, el diseño básico o su código fuente, con el fin de encontrar fallos sin necesidad de ejecutarlo; el mismo se puede realizar de forma manual o automática y para ello existen herramientas diseñadas para automatizar este proceso, cada una centrada en la búsqueda de errores en una región específica de la estructura del programa (léxico, sintaxis o semántica).

Automatización de las pruebas

Para automatizar las pruebas se han creado una serie de herramientas que ayudan en gran medida a su realización, ya que en estos días los sistemas que se necesitan son cada vez más grandes y complejos y con estas herramientas se puede reforzar el desarrollo de todo el proceso de pruebas, disminuyendo tiempo, esfuerzo y costo. Entre las herramientas diseñadas para realizar análisis estático al código fuente de un sistema o componente de software se encuentran:

PMD

Es una herramienta muy conocida, tanto por su facilidad de uso como por su fácil integración con las tecnologías que se utilizan para desarrollar un proyecto. Su funcionamiento se basa en detectar patrones, los cuales son posibles errores que pueden aparecer en tiempo de ejecución: código no usado, código no óptimo, expresiones demasiado complicadas, código duplicado, código que no se puede ejecutar nunca porque no hay manera de llegar a él, expresiones lógicas que puedan ser simplificadas, malos usos del lenguaje, etc. Estos patrones se encuentran catalogados en distintas categorías, entre las cuales estarían las básicas: las de tamaño, acoplamiento, tratamiento de excepciones, nombrado, complejidad, optimización, seguridad, etc. Además va más allá de lo que únicamente es la detección de errores en el código ya que también encuentra errores en el uso que se haga con las tecnologías. Esta herramienta es

extensible y configurable ya que se pueden añadir nuevas reglas o configurar las que ya se incluyen, en caso de que esto fuera necesario. Además también se puede configurar el formato en el que se desea que devuelva los resultados: xml, html, txt, etc.

Limitaciones:

El análisis que realiza no incluye la documentación javadoc.

No realiza el análisis de flujo de datos.

No soporta análisis meta-data.

No puede ejecutarse en más de un fichero de cada vez.

CPD

Forma parte de PMD, aunque funciona de una manera autónoma, y es sin duda uno de los analizadores más útiles que se pueda encontrar. CPD son las siglas de *"*Copy Paste Detector*"* y con ese nombre queda clara cuál es su misión: buscar duplicidades en el código. Todo aquello que esté escrito más de una vez en el código será detectado por esta herramienta.

Pero, ¿qué sentido tiene encontrar código que haya sido copiado y pegado?

Cuando hay código duplicado, triplicado o n-plicado en una aplicación se está en presencia de un grave problema: si hay que hacer cambios en código duplicado de la aplicación habrá que recorrer todo el código de la misma para repercutir ese mismo cambio en todas las copias que se hayan hecho.

Los problemas derivados de la función "copiar y pegar" tienen su solución en su función hermana: "cortar y pegar": Si inicialmente sólo se necesita comprobar si un año es bisiesto en un punto y, posteriormente, se necesita hacer esta comprobación en más sitios lo que se tiene que hacer es mover la comprobación a un lugar más adecuado. Para ello se corta y pega el código en un sitio común a todos los puntos y, desde ellos, se invoca la validación. Si inicialmente la aplicación sólo tenía jefes y posteriormente se añadieron los empleados, lo que hay que hacer es crear un nuevo tipo y allí se corta y pega lo que ambos tengan en común. Al igual que PMD, CPD permite devolver los resultados de sus análisis en varios formatos, como por ejemplo xml o txt.

CheckStyle

Inicialmente se desarrolló con el objetivo de crear una herramienta que permitiese comprobar que el código de las aplicaciones se ajustase a los estándares dictados por Sun Microsystems, empresa creadora del lenguaje Java. Sin embargo, posteriormente se añadieron nuevas capacidades que han hecho que sea un producto muy similar a PMD. Es por ello que también busca patrones en el código que se ajustan a categorías muy similares a las de este analizador. Una ventaja que tiene con respecto a PMD es que además encuentra errores en la documentación que se puedan haber escrito en el javadoc. PMD no realiza ningún tipo de validación mientras que Checkstyle es muy riguroso en ese sentido. Gracias a ello, si se tiene una función que recibe dos parámetros y se documenta uno o ninguno, Checkstyle avisará de qué parte de la función está sin documentar. Además, en las últimas versiones se le han añadido nuevas características que permiten también encontrar problemas en el diseño de las clases, encontrar código duplicado o buscar patrones de error conocidos.

Limitaciones:

No hace un análisis de flujo de datos.

Classycle

Es especialmente útil para encontrar las dependencias cíclicas entre las clases y los paquetes. El analizador de Classycle analiza las dependencias estáticas de la clase y el paquete de aplicaciones Java o bibliotecas. Dependencias cíclicas de las clases o paquetes pueden ser un signo de mal diseño orientado a objetos. Una dependencia estática es una dependencia entre clases o paquetes que se puede detectar por el compilador. Esto tiene que ser distinguido de dependencia dinámica que es una dependencia entre objetos. Averiguar dependencia cíclica estática es el objetivo principal de Classycle. A partir del análisis de los archivos se generan gráficas y las dependencias de la clase y el paquete se calculan. Classycle puede instalarse como plug-in de Eclipse en una PC local.

Limitaciones:

Solamente analiza las dependencias entre clases y paquetes.

JLint

Es una herramienta que realiza análisis de flujo de datos, de sincronización de hilos de código y de herencia. Analiza el bytecode de Java para realizar análisis, sin embargo también incluye un componente que permite realizar análisis interprocedural.

El analizador está compuesto por dos programas:

- ✓ AntiC que realiza la verificación sintáctica. Es un programa que realiza análisis léxico y sintáctico de ficheros Java.
- ✓ JLint que efectúa la verificación semántica analizando ficheros .class. Puede detectar errores de sincronización de hilos de código, problemas que pueden darse en la jerarquía de herencia, errores en el flujo de datos.

Para realizar el análisis hace dos pasadas: en la primera se tratan los métodos de forma modular con interpretación abstracta y en la segunda se realiza el análisis de interbloqueos.

Limitaciones:

Es difícil de extender.

ESC/Java

El "Verificador de Extendido Estático para Java" es una herramienta de programación que trata de encontrar errores comunes de ejecución en Java y programas en tiempo de compilación. Utiliza la técnica “Demostración de teorema” para hacer comprobación de tipos: qué se encuentra en los límites de una matriz, si una variable de tipo entero es mayor que cero, etc. Y los errores derivados de la aritmética modular y/o multihilo. Es una herramienta fácil de utilizar, que se puede ejecutar desde la línea de comandos o bien integrada en el IDE Eclipse o Netbeans.

Limitaciones:

Los mensajes de descripción de fallos pueden resultar complicados de entender por los desarrolladores.

Un fallo puede referirse a varios ficheros de código ESC/Java.

FindBugs

Es un producto de la Universidad de Maryland que, como su nombre indica, está especializado en encontrar errores. Al igual que los demás tiene una serie de categorías donde poder catalogar dichos errores, y es que según FindBugs estos pueden ser malas prácticas, mal uso del lenguaje, internacionalización, posibles vulnerabilidades, mal uso del multihilo, rendimiento, seguridad, etc. También hay disponibles complementos para Eclipse, Netbeans, e IntelliJ IDEA. Es uno de los analizadores más utilizados y mejor documentados.

Limitaciones:

No proporciona un mecanismo estándar de análisis meta-datos que permita obtener mejores resultados.

No proporciona path-sensitive analysis.

No realiza un análisis sensitivo de contexto interprocedural.

El análisis que realiza no incluye la documentación javadoc.

Sonar

Sonar es una plataforma abierta para gestionar la calidad del código. Está diseñado para analizar código escrito bajo el lenguaje Java y los plugins comerciales permiten cubrir C, C #, Flex, Natural, PHP, PL/SQL, Cobol y Visual Basic. Sonar tiene una forma muy eficiente de navegar, esto permite descubrir rápidamente los proyectos y/o componentes para establecer planes de acción. Es una aplicación basada en web. Las reglas, alertas, umbrales, exclusiones, ajustes... se pueden configurar en línea. Al aprovechar su base de datos, Sonar no sólo permite combinar métricas por completo, sino también para que se mezclen con las medidas históricas. Internamente utiliza las herramientas: PMD, CPD, FindBugs y Checkstyle y unifica sus resultados mostrando, además, más información: complejidad, número de métodos, de clases, de líneas de código, mantenibilidad, etc. Todo de una manera muy visual y muy intuitiva.

Limitaciones

Necesita conexión a Internet para cargar los plugins necesarios para realizar el análisis estático al código.

Después de realizado el estudio de varias herramientas destinadas a realizar análisis estático al código fuente escrito en el lenguaje de programación Java se decide escoger Checkstyle ya que es un software de código libre, es fácil de utilizar y entender, una vez que se integra con el IDE Eclipse permite recrear el

ambiente de desarrollo utilizado en el proyecto del cual proviene el sistema o componente a probar. Además está disponible sin necesidad de conexión a Internet.

Tendencias Nacionales

En la universidad se han realizado varias investigaciones que han sido motivo de trabajos de diploma acerca de las pruebas de caja blanca como los son:

- Procedimiento para la realización de pruebas de caja blanca en la UCID. Facultad 8
- Pruebas unitarias de software en la plataforma J2EE. Facultad 4
- Procedimiento para la realización de pruebas unitarias dentro del proyecto Sistema Único de Aduanas. Facultad 3
- Propuesta de procedimiento y herramientas para realizar pruebas de caja blanca en el área temática APS. Facultad 7
- Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca. Facultad 7

Estas investigaciones están enfocadas al análisis dinámico del código, dejando a un lado el estático, motivo por el cual surge la necesidad de crear un procedimiento para realizarlo en le CEIGE, fomentando así los pocos conocimientos que se tienen respecto al tema, y por el hecho de que la corrección es más fácil y menos costosa durante la evaluación estática que durante la dinámica.

En la evaluación dinámica lo que se detecta es el fallo, no la causa que lo provoca y luego de ser detectada se procede a localizarla. Sin embargo con las técnicas estáticas se obtiene directamente dónde radica el problema, pudiendo ser corregido al instante obviando la tarea de localización. Esto significa, que las técnicas estáticas son más baratas que las dinámicas.

Conclusiones del capítulo

En el presente capítulo se realizó un análisis sobre las pruebas de caja blanca, específicamente las estáticas, lo que evidenció que la fase de pruebas lleva mucho tiempo en el desarrollo del software; por tal motivo se concluye que no se debe esperar al final para realizarlas sino que se debe probar según se va desarrollando. Evaluar las posibilidades de automatización de las pruebas de caja blanca estática, estudiar herramientas existentes y adaptarlas a las necesidades propias de cada proyecto ahorrará tiempo y esfuerzo al equipo de desarrollo garantizando así una mayor calidad para el producto final. Después de realizada la investigación sobre los tipos de análisis estático se deciden utilizar el chequeo de estilo y el chequeo de errores ya que son los que realiza la herramienta Checkstyle**,** elegida para llevar a cabo la automatización de las pruebas. Las métricas que se utilizarán para evaluar el procedimiento serán: Facilidad de prueba (Examinabilidad) y Facilidad de cambio (Cambiabilidad).

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Introducción

Las pruebas de caja blanca tienen un impacto importante en el código y la calidad del producto final, por tal razón en el presente capítulo se ofrece una propuesta de solución para garantizar la realización de este tipo de pruebas en el CEIGE. Se elabora un procedimiento para aplicar Pruebas de Caja Blanca Estática en el proyecto Sistema Automatizado de la Gestión Bancaria, donde se desarrolla bajo la tecnología Spring y el lenguaje de programación Java, evaluando así la mantenibilidad en los componentes desarrollados en el mismo. Se explican los diferentes métodos, técnicas y herramientas que se van a emplear con el objetivo de mostrar la magnitud que tendrá el mismo.

Aspectos correspondientes al procedimiento

Objetivos

- Analizar el código fuente en los sistemas desarrollados bajo la tecnología Spring y el lenguaje de programación Java para determinar el grado de mantenibilidad de los componetes.
- Utilizar la herramienta Checkstyle para automatizar las Pruebas de Caja Blanca Estática.

Alcance

La propuesta de solución comprende aquellos productos de software que se encuentren en desarrollo bajo la tecnología Spring y el lenguaje de programación Java.

Precondiciones

Para poder aplicar la propuesta de solución debe considerarse su puesta en funcionamiento sobre un módulo que esté en desarrollo.

Acceder al código del componente a probar de la misma forma que en el proyecto sin conectarse al repositorio.

La PC del Probador debe tener instalado el IDE Eclipse con el plugins de Checkstyle integrado y ya configurado.

Roles y Responsabilidades

Los roles que intervendrán en el proceso de Pruebas de Caja Blanca Estática son los definidos en el Proceso de Mejora (Ver Anexo 10). En este caso se ajustarán las responsabilidades según las necesidades que requiera el procedimiento.

- **Administrador de la Calidad**

Este rol asumirá todas las responsabilidades descritas en el Proceso de Mejora (Ver Anexo 10) sin modificaciones.

- **Analista del sistema**

Este rol asumirá todas las responsabilidades descritas en el Proceso de Mejora (Ver Anexo 10) además de añadírsele:

- ✓ Realizar la solicitud de las pruebas por ser el interesado en que el código fuente de la aplicación que se está desarrollando mantenga una buena estructura.
- ✓ Participar en las reuniones de inicio y cierre.
- ✓ Entregar las no conformidades detectadas al Desarrollador para que sean corregidas y posteriormente entregar el componente para nuevas iteraciones.

- **Desarrollador**

Este rol asumirá todas las responsabilidades descritas en el Proceso de Mejora (Ver Anexo 10) exceptuando:

- ✓ Elaborar las pruebas de unidad.
- ✓ Ejecutar los casos de prueba y generar no conformidades asociadas al mismo.
- ✓ Registrar y analizar los resultados de las pruebas.

Porque en este caso las pruebas son realizadas por una herramienta y no se diseñan. No se utilizan los diseños de casos de prueba ya que la técnica a utilizar no es la manual. El encargado de generar las no conformidades asociadas y registrar el análisis de los resultados de las pruebas es el Probador.

- **Probador**

Este rol asumirá todas las responsabilidades descritas en el Proceso de Mejora (Ver Anexo 10) exceptuando la ejecución de los diseños de casos de prueba porque las pruebas serán de forma automatizada y no manual.

Etapas del proceso de Prueba de Caja Blanca Estática

La propuesta de solución cuenta con cuatro etapas principales: Planificación, Diseño, Ejecución y Evaluación. Cada una de ellas posee: Artefactos de entrada y salida, Roles y Actividades. (Ver Figura 1)

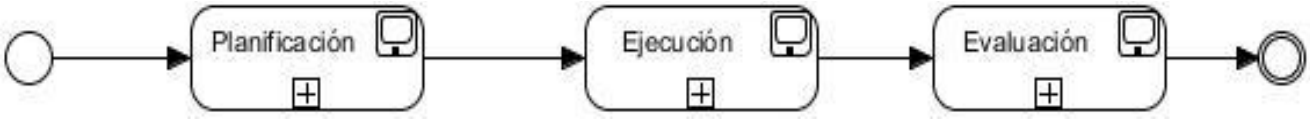


Figura 1 Etapas del proceso de Prueba de Caja Blanca Estática.

Tabla 1 Etapas del proceso de Prueba de Caja Blanca Estática.

Etapas	Descripción
Planificación	En esta etapa se origina un intercambio de artefactos entre el Analista del sistema y el Administrador de la calidad luego de realizar la reunión de inicion para pasar a diseñar el Plan de Prueba donde se describe la estrategia, los recursos y planificación de las Pruebas de Caja Blanca Estática, así como el entorno de desarrollo del proyecto Sistema Automatizado de la Gestión Bancaria.
Ejecución	Esta etapa tiene como objetivo fundamental aplicar las Pruebas de Caja Blanca Estática a los componentes desarrollados bajo la tecnología Spring y el lenguaje de programación Java y como resultado, encontrar los fallos que estos presentan, quedando descritos en el Registro de no conformidades.
Evaluación	En esta etapa se valoran las principales y más urgentes dificultades encontradas durante todo el proceso de Pruebas de Caja Blanca Estática.

Descripción de las Etapas del Proceso de Pruebas de Caja Blanca Estática

Planificación

Las pruebas de software deben ser planificadas con un tiempo anticipado antes de que comience su ejecución. En esta etapa se precisan el tipo de pruebas que se le realizarán al software, en este caso Caja Blanca Estática, se identifican los roles que intervendrán y sus responsabilidades, artefactos que serán consultados, generados o modificados, los recursos requeridos por el sistema y el ambiente para realizar las pruebas. Todas las planificaciones de las pruebas deben quedar reflejadas en el Plan de prueba ya que es el documento donde se describe y se deja claro cómo el equipo de calidad debe realizar las pruebas en un tiempo determinado según el cronograma de pruebas. Así como la definición de los elementos: escenario de pruebas, recursos del sistema, herramientas que serán utilizadas, estrategia de pruebas y evaluación de las mismas. (Ver Figura 2)

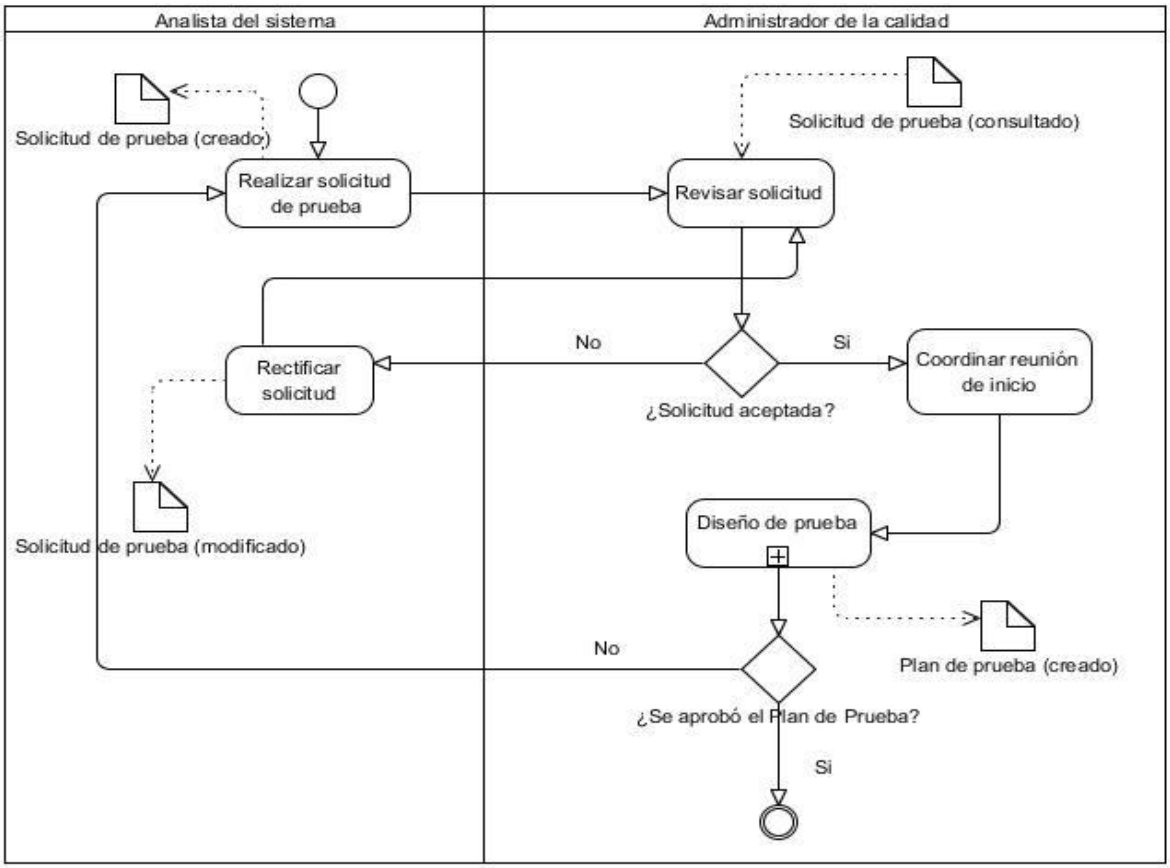


Figura 2 Diagrama de la Etapa Planificación.

Artefactos de la Etapa Planificación

- Solicitud de Prueba de Caja Blanca Estática

Este documento recoge la solicitud que realiza el Analista del sistema al Administrador de la calidad para la realización de las pruebas al producto seleccionado previamente. (Ver Anexo 2)

- Plan de Prueba de Caja Blanca Estática

Constituye el documento Rector de los procesos de pruebas y sirve como guía de referencia para las partes involucradas. Es el documento donde se definen estrategias y recursos necesarios para ejecutar una metodología de pruebas que incluye los objetivos de las pruebas, el alcance de las mismas, los roles involucrados y sus responsabilidades, el cronograma, los resultados de la evaluación de las pruebas y los criterios de criticidad. (Ver Anexo 3)

Diseño

Este subproceso de la etapa de Planificación tiene como objetivo principal diseñar el Plan de prueba que describe los recursos y la planificación de las mismas. El mismo recoge todo lo referente al ambiente de desarrollo del Proyecto SAGEB y debe ser aprobado por todos los implicados. El responsable de realizar esta actividad es el Administrador de la calidad quien elabora la plantilla del Plan de pruebas, identifica los objetivos, el ambiente de trabajo y cantidad de iteraciones que se realizarán. (Ver Figura 3)

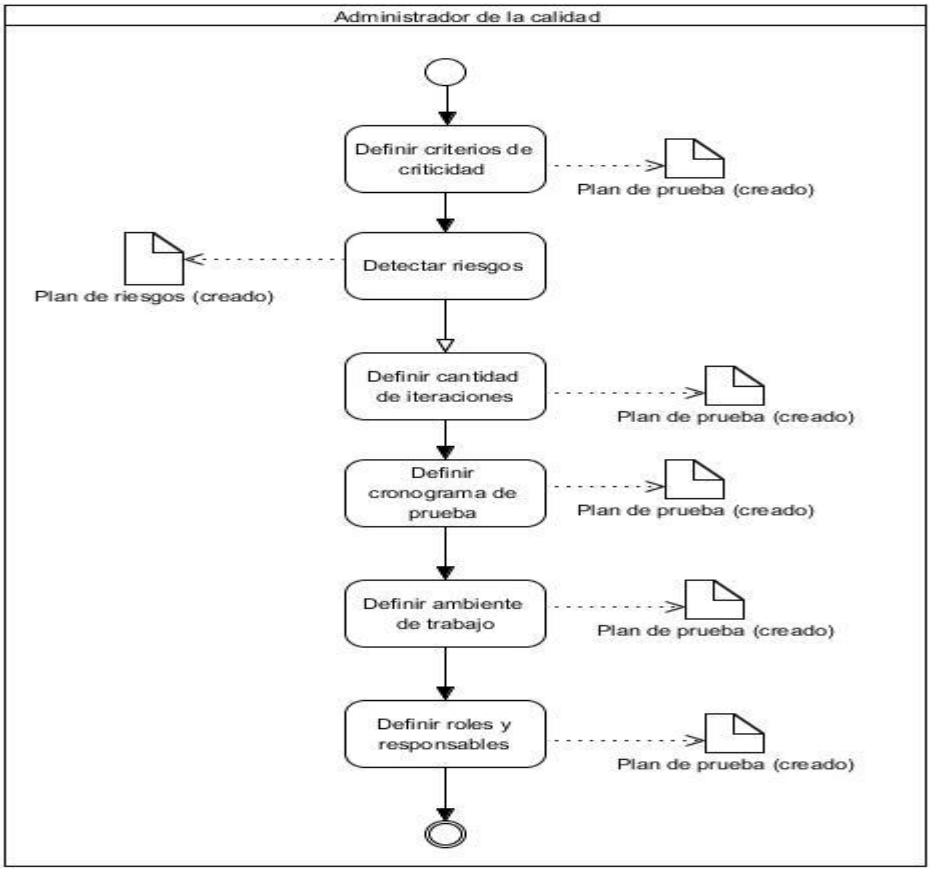


Figura 3 Diagrama del Diseño.

Artefactos de la Etapa Diseño

- Plan de Riesgos

Es el documento donde quedan plasmados todos los riesgos que pueden surgir durante el proceso de pruebas. En él se recogen el tipo de riesgo que puede ser clasificado en (tecnológico, personal, organización, herramientas, estimación, requerimientos), el impacto que pueda tener el mismo durante la ejecución de las pruebas, la probabilidad de que ocurra (alta, baja, media y muy alta) y los efectos (catastrófico, insignificante, tolerable, serio). Además de plantearse una estrategia para mitigar el impacto de los mismos. (Ver Anexo 4)

En el procedimiento se propone hacer una evaluación horizontal de los riesgos, es decir, evaluar en todas las etapas del Proceso de Pruebas de Caja Blanca Estática, aunque se haga un mayor énfasis en la etapa de Planificación.

Ejecución

Una vez entregado el Estándar de codificación y el componente a probar por el Analista del sistema y los datos de las pruebas por el Administrador de la calidad, el Probador puede proceder a la ejecución de las mismas haciendo uso de la herramienta Checkstyle, con el objetivo de descubrir defectos o no conformidades en el componente. Las pruebas se realizan en diferentes iteraciones en correspondencia con los defectos corregidos, verificando que las no conformidades detectadas sean resueltas con el fin de lograr una mayor estabilidad del software. Tiene como entrada el Estándar de codificación para Java y como salida el Registro de No Conformidades donde el Probador reflejará todas las anomalías encontradas. (Ver Figura 4).

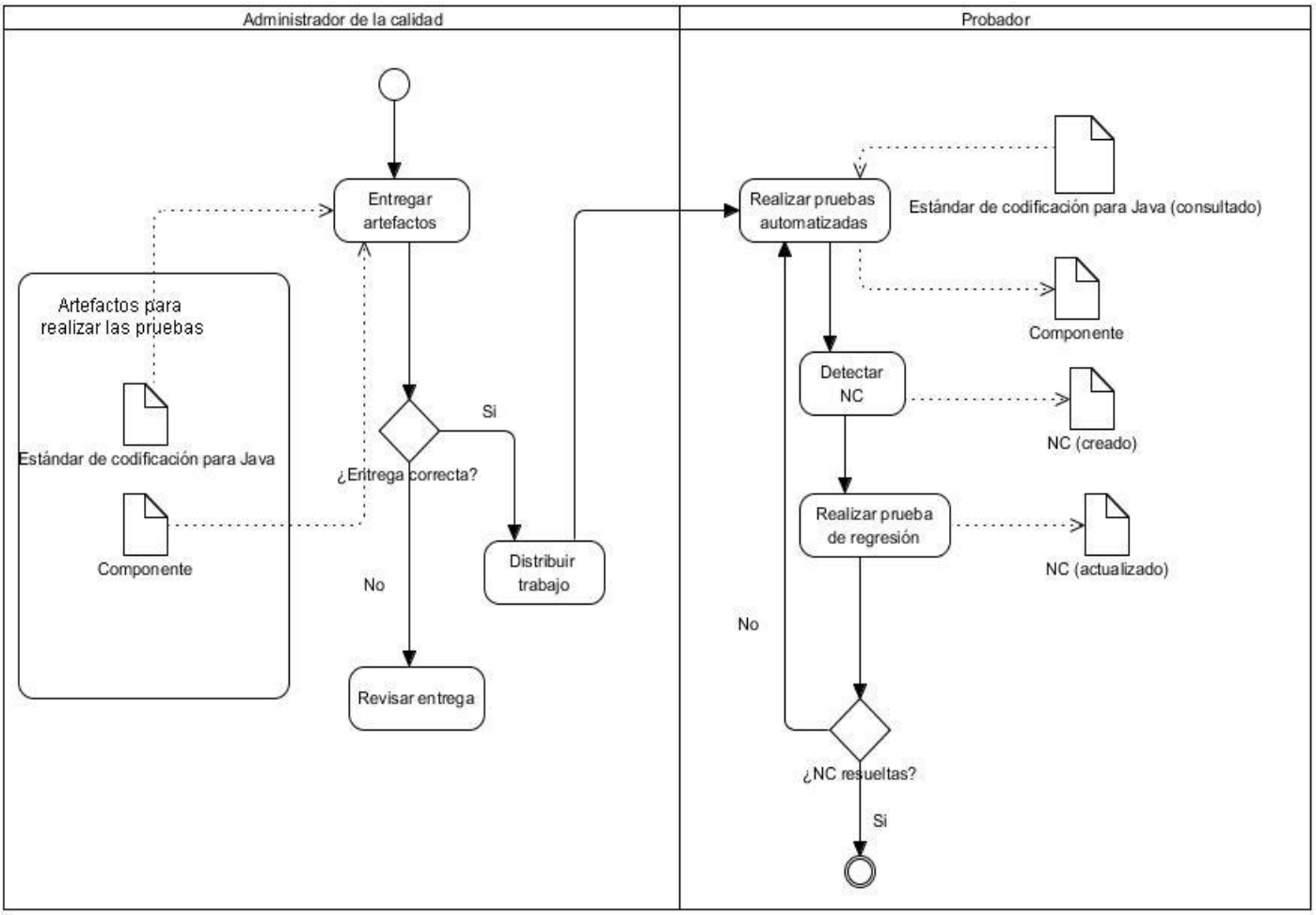


Figura 4 Diagrama de la Etapa de Ejecución.

Ambiente para Pruebas de Caja Blanca Estática

- Eclipse

Es un Entorno de Desarrollo Integrado (del inglés IDE), de código abierto, multiplataforma, para desarrollar lo que el proyecto que lleva su mismo nombre llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Provee soporte para Java y control de versiones. No tiene por qué ser usado únicamente para soportar otros lenguajes de programación. Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Eclipse dispone de un editor de texto con resaltado de sintaxis. La compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes para creación de proyectos, clases, tests, etc., y refactorización.

- Checkstyle

Es una herramienta que ayuda a los programadores a escribir código Java agregado a estándares de codificación establecidos, facilitando para ello la automatización del proceso de chequeo del código generado. Por defecto, incorpora las recomendaciones de Sun sobre el estilo de código, pero estas reglas pueden ser redefinidas e incluso creadas completamente desde cero por el usuario, lo que convierte este plug-in en adaptable al estilo de codificación interno del entorno que se esté utilizando, sea cual sea. Checkstyle puede instalarse como plug-in de Eclipse en una PC local.

- Configuración del Eclipse integrado con Checkstyle

Descargar el plug-in de Checkstyle para Eclipse y copiarlo en la carpeta plugins donde se encuentra el ejecutable de Eclipse. Luego se ejecuta el Eclipse.

Una vez ejecutado el Eclipse se accede al menú "Windows -> Preferences". (Ver Figura 6).

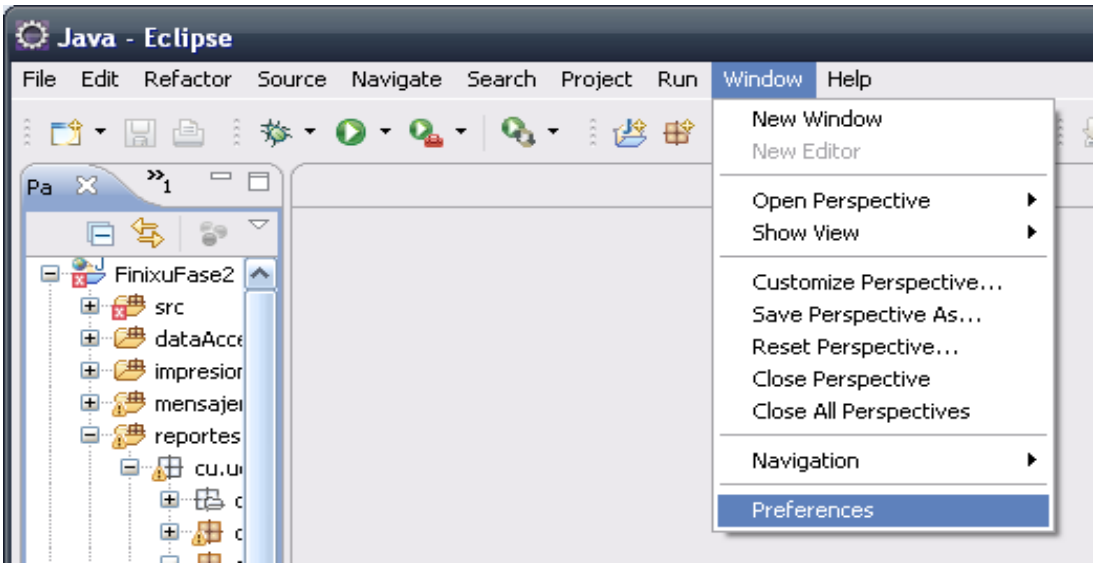


Figura 5 Ruta de localización de Checkstyle en Eclipse.

Aparece el nombre de la herramienta en el árbol de la sección izquierda. (Ver Figura 7). Presionar el botón "OK".

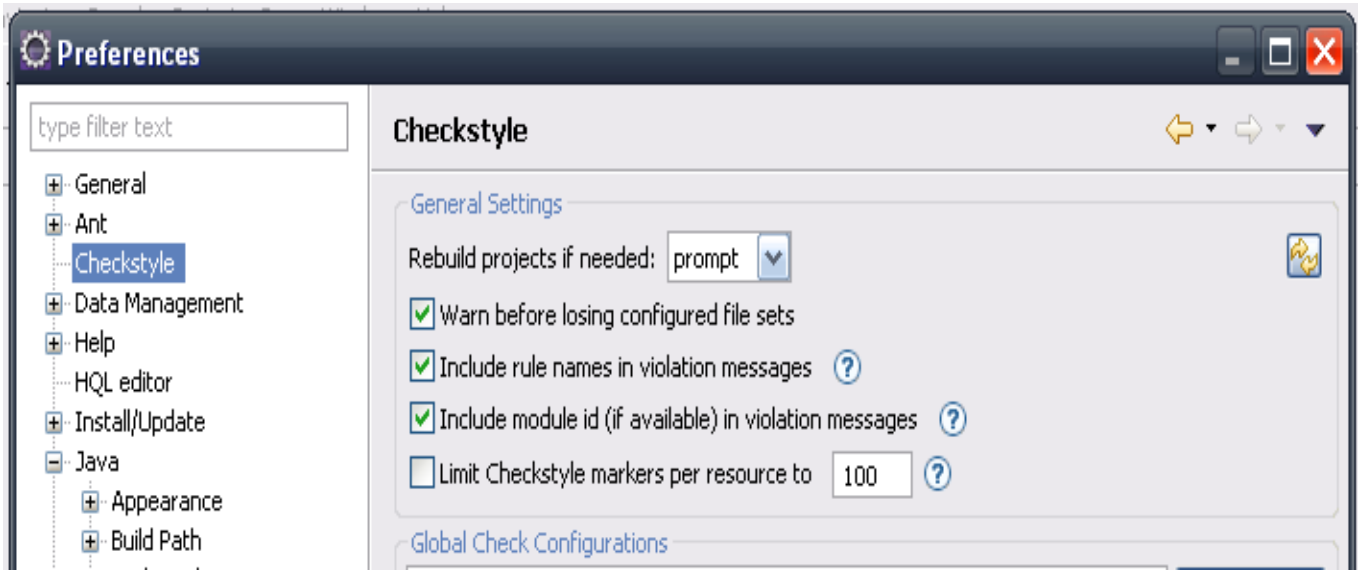


Figura 6 Localización de Checkstyle en el árbol de preferencias en Eclipse.

Cada vez que se quiera analizar el proyecto se presiona el clic derecho sobre el nombre del mismo y se selecciona la opción Checkstyle -> Activate Checkstyle y se mostrarán los errores en las clases señalados

en amarillo y los comentarios respectivos cuando se coloca el cursor sobre la línea señalada. (Ver Figura 7 y 8) respectivamente.

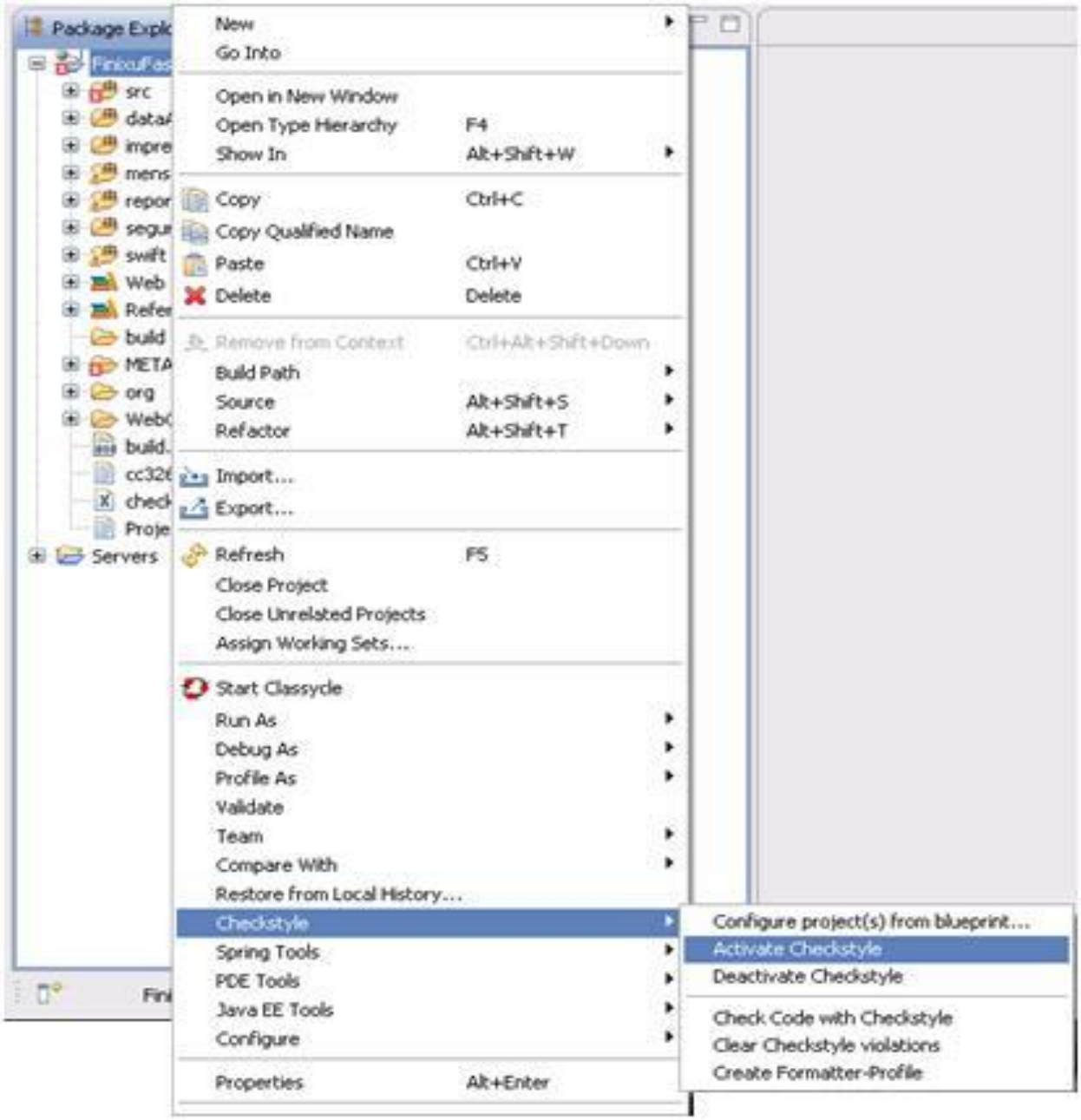


Figura 7 Activar la herramienta para el análisis.

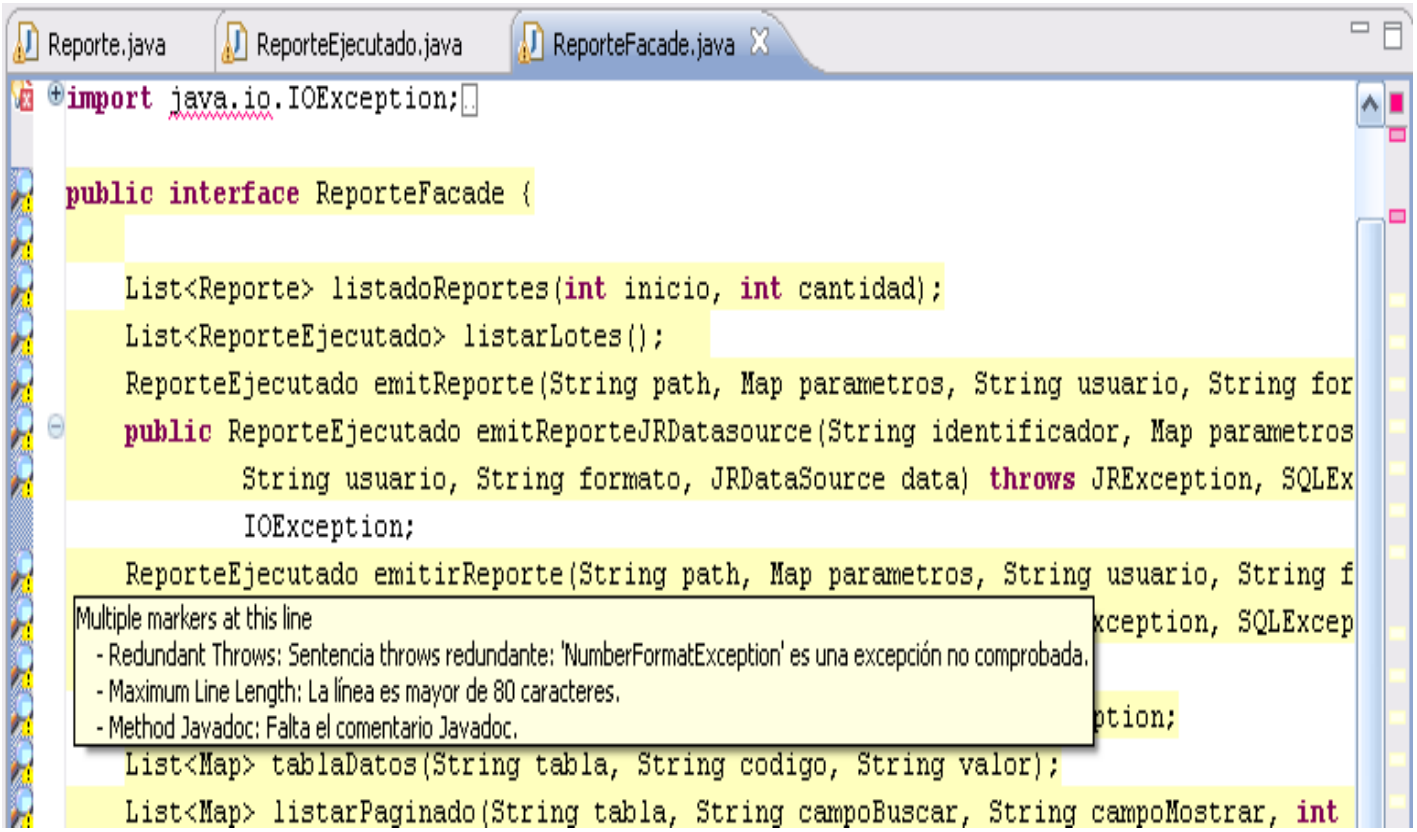


Figura 8 Comentarios de los errores en el código.

Artefactos de la Etapa Ejecución

- Estándar de codificación para Java

En un estándar de codificación se definen las reglas que se siguen para la escritura del código fuente, de tal manera que a otros programadores se les facilite entender el código ya implementado. Los estándares de codificación facilitan el aseguramiento de la calidad del producto a lo largo del ciclo de desarrollo del mismo. Cada proyecto define el que regirá en su entorno de trabajo según el lenguaje de programación en que se esté implementando y otras especificaciones propias del entorno de desarrollo.

En particular el de Java, tiene reglas como: Las clases inician en mayúsculas, los atributos y métodos inician con minúsculas, las constantes son todas en mayúsculas. También menciona la forma de abrir y cerrar bloques de código y la forma de poner comentarios. (Ver Anexo 5).

- Registro de No Conformidades

Es el documento donde se registran todas las no conformidades o defectos detectados durante todo el proceso de prueba. En dicho documento se especifica cuál fue el error detectado, localización del mismo, quien lo detectó, la etapa en que se detectó, la fecha en que se detectó y su estado (Pendiente o Resuelto). (Ver Anexo 6).

Evaluación

Una vez concluida la ejecución de las pruebas se convoca a una reunión de cierre donde se recopila toda la información referente al proceso de prueba que quedará plasmada en el Informe de evaluación. En esta reunión se verifica la conformidad de todos con el documento previamente discutido y analizado para concluir las pruebas. (Ver Figura 9).

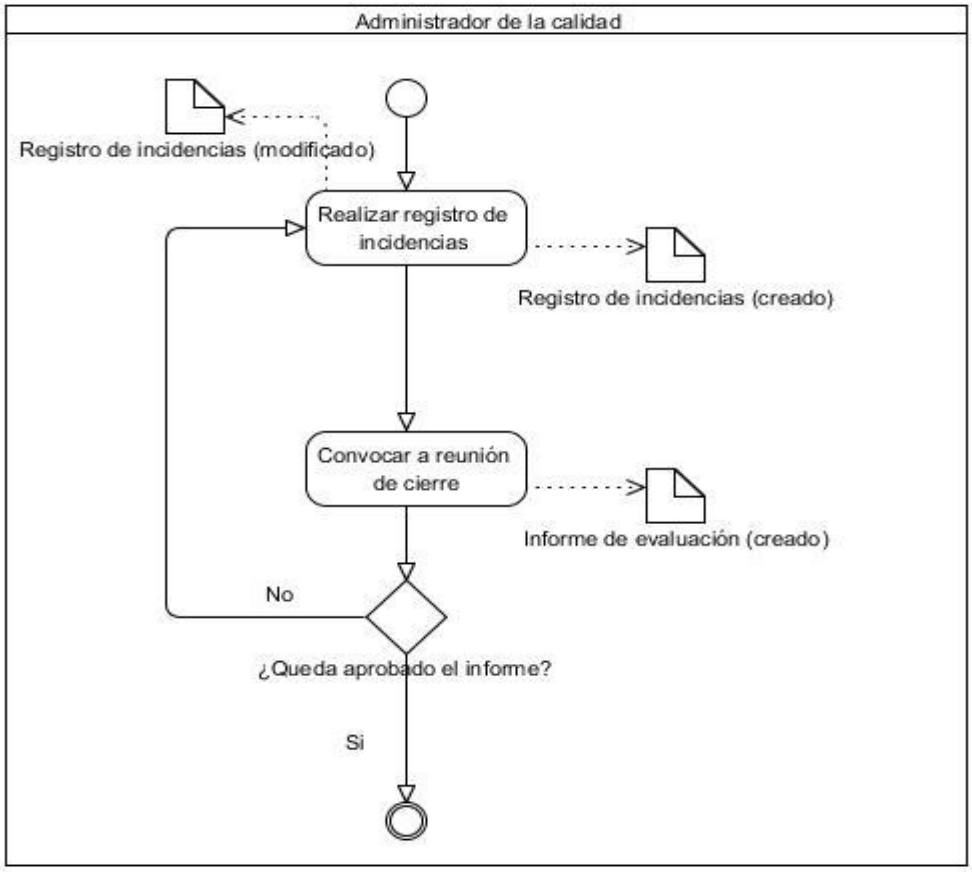


Figura 9 Diagrama de la Etapa Evaluación.

Artefactos de la Etapa Evaluación

- Registro de Incidencias

En este documento se recogen todos los acontecimientos transcurridos durante el proceso de pruebas. Se hace un resumen del total de horas aprovechadas y perdidas en las pruebas, ya sea por ausencia de los especialistas o por problemas tecnológicos. Además se recoge la cantidad de horas que demora realizar una actividad, el tipo de actividad, el responsable de realizarla y el tiempo mínimo que demora realizarla. Es conformado por el Administrador de la calidad. (Ver Anexo 7).

Técnica de prueba

Para realizar las Pruebas de Caja Blanca Estática se utilizará el análisis del código automático que no es más que el realizado por un programa de ordenador sobre el código; en este caso se utilizará la herramienta Checkstyle integrada con el IDE de desarrollo Eclipse para código escrito en el lenguaje de programación Java.

Documentos de referencia

Para conformar las plantillas que se utilizarán posteriormente en el capítulo 3 se tomaron como base los artefactos de Calisoft referenciados a continuación:

- ✓ Plantilla de Solicitud de pruebas de liberación_Calisoft_08 Septiembre 2011.dotx
- ✓ Plantilla de No conformidades_CAL.xls
- ✓ Plan de Mitigación de Riesgos_CAL.dotx
- ✓ 0516_Roles y responsabilidades del proceso de mejora.pdf
- ✓ Plan de Prueba de Liberación. Calisoft.pdf

Términos y definiciones

- Actividad: Tareas que tienen un propósito y se asigna a un rol.
- Artefacto: Productos tangibles del proyecto, que son producidos, modificados y usados por las actividades.

- Defectos: 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.
- Flujo de trabajo: Secuencia de actividades para producir determinados artefactos.
- No conformidades: Defectos y/o sugerencias detectadas por los probadores.
- Trabajador: Roles que definen el comportamiento y responsabilidad de los individuos.
- Validación: Proceso para asegurar que el producto del trabajo es el correcto, y corresponde perfectamente con la necesidad del cliente.

Conclusiones del capítulo

En este capítulo se definió que el procedimiento estará compuesto por tres etapas fundamentales: Planificación, Ejecución y Evaluación permitiendo una mayor organización en la ejecución de las actividades definidas en cada etapa. Los roles involucrados en el mismo serán el Administrador de la calidad, el Analista del sistema, el Desarrollador y el Probador, según los resultados arrojados en el diagnóstico inicial realizado en el Proyecto SAGEB (Ver Anexo 9) y los roles definidos en el Programa de Mejora (Ver Anexo 10). Asignar responsabilidades en el proceso de Pruebas de Caja Blanca Estática exigirá un mayor compromiso por parte de los involucrados para con el proceso. Mediante la integración de la herramienta Checkstyle con el IDE de desarrollo Eclipse se obtendrán datos estadísticos que ayudaran a obtener el grado de mantenibilidad que poseen los componentes probados.

CAPITULO 3: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

En este capítulo se realiza la validación de la propuesta de solución. Para ello se le aplica el proceso de Prueba de Caja Blanca Estática al subsistema Conciliaciones Bancarias, actualmente en desarrollo en el proyecto Sistema Automatizado de la Gestión Bancaria (SAGEB), bajo la tecnología Spring y el lenguaje de programación Java, utilizando la técnica automatizada mediante la herramienta Checkstyle integrada con el IDE de desarrollo Eclipse. Además se expondrán los resultados obtenidos aplicando las métricas de la mantenibilidad: Registrabilidad de cambios y Pruebas sin esfuerzo.

Encuesta realizada al grupo de calidad del CEIGE

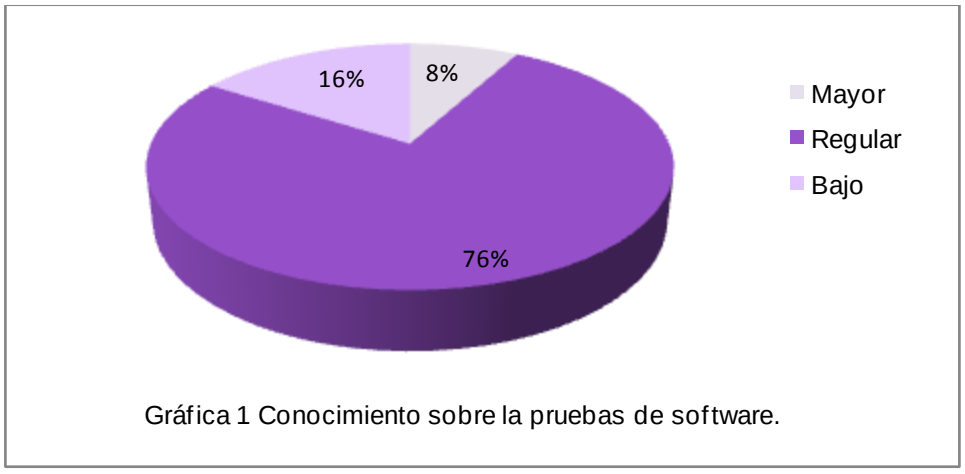
Con el objetivo de conocer y profundizar sobre el estado de la calidad y de las pruebas de software en el CEIGE, se seleccionó a 25 personas del grupo de calidad (estudiantes y profesionales), para realizarle una encuesta (Ver Anexo 8) con vista a recoger su opinión y para de esta manera tener una muestra del Estado del Arte a través del conocimiento de los mismos sobre del tema.

La encuesta arrojó los siguientes resultados. De un total de 25 personas encuestadas:

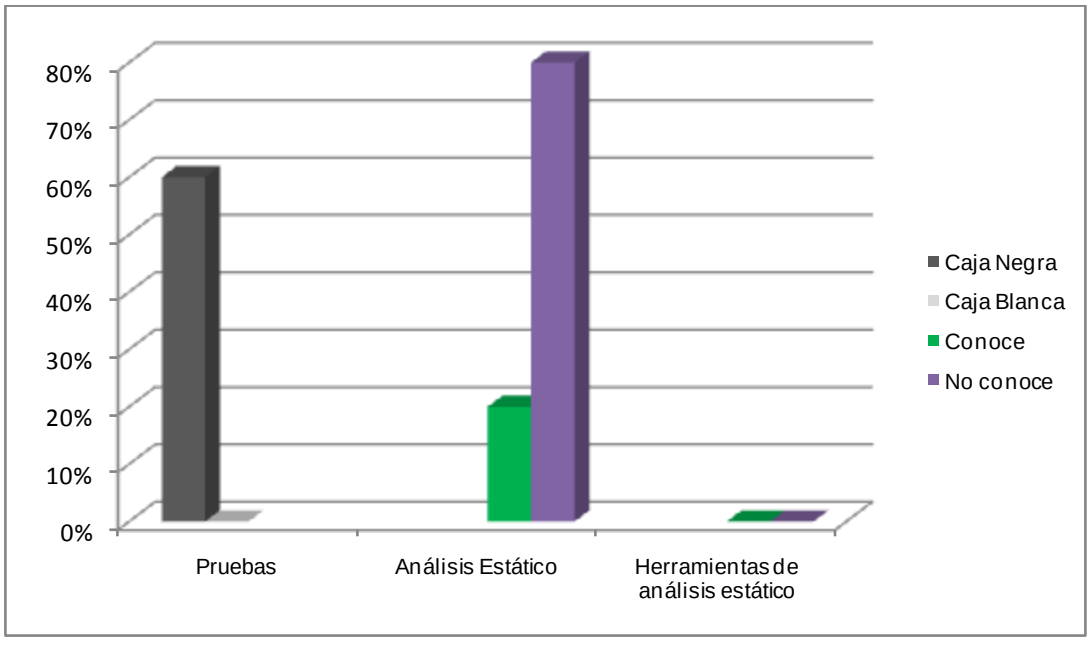
- Todas ellas poseen un conocimiento regular sobre el tema de calidad, lo que representa un 100 % de dominio regular del tema.
- Sólo 2 personas poseen mayor conocimiento sobre la aplicación de pruebas al software, 19 tienen un conocimiento medio y 4 tienen un conocimiento bajo, lo que representa un 8 % de personas con mayor conocimiento del tema, un 76 % de personas con conocimiento regular y un 16 % de personas con bajo conocimiento del tema.
- Las 15 personas que plantearon haber realizado pruebas al software solo hicieron pruebas de caja negra y el resto confiesa que nunca ha realizado ningún tipo de pruebas, representando un 60 % de personas que ha realizado pruebas funcionales quedando el 40 % restante que no ha realizado ningún tipo de pruebas.
- Solo 5 de ellas poseen conocimiento sobre el análisis estático de código, representando a un 20 % y el 80 % restante no tiene conocimiento alguno del tema.

- Ninguna de las 25 personas encuestadas posee conocimiento sobre herramientas existentes para realizar análisis estático al código.

Algunos de estos datos estadísticos se reflejan en los siguientes gráficos, para brindar una idea más completa acerca de la situación real existente:



Gráfica 1 Conocimiento sobre la pruebas de software.



Gráfica 2 Resultado del conocimiento sobre los tipos de pruebas, análisis estático y herramientas para realizar este tipo de análisis.

Descripción del Proyecto SAGEB

SAGEB es el Sistema Automatizado de la Gestión Bancaria. El mismo desarrolla bajo la tecnología Spring y el lenguaje de programación Java y su objetivo es informatizar el Sistema de Gestión del Banco Nacional de Cuba.

- Spring

Es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar, prácticas comunes en la industria. Está diseñado como una serie de módulos que pueden trabajar independientemente uno de otro. Además, intenta mantener un mínimo acoplamiento entre la aplicación y el propio framework de forma que podría ser desvinculada de él sin demasiada dificultad.

- Java

Es un lenguaje de programación orientado a objetos, robusto y fácil de aprender, permite la codificación de una manera rápida y flexible, al integrarse con los diferentes framework de desarrollo, permite programar páginas web dinámicas con accesos a bases de datos utilizando XML con cualquier tipo de conexión de red entre cualquier sistema. Es un lenguaje multiplataforma de código abierto, distribuido, que proporciona un conjunto de clases para su uso en aplicaciones de red. Es compilado en la medida en que su código fuente se convierte en una especie de código máquina. Por otra parte, es interpretado, ya que se puede ejecutar directamente sobre cualquier máquina a la cual se hayan trasladado el intérprete y el sistema de ejecución en tiempo real.

- Quarzo

Es un Sistema Informático para el Banco Nacional de Cuba. Su finalidad es permitir la informatización de operaciones, determinadas transacciones, así como gestionar los procesos bancarios del país. Permitiendo realizar una mejora significativa en cuanto al ahorro de recursos. Además como modalidad de proyecto, incluye funciones como la capacitación e impartición de cursos a los informáticos del banco para el manejo de la aplicación en Java, y otros lenguajes de dominio necesario. Esta solución está destinada a empresas nacionales, donde aporta mayor seguridad en las validaciones de datos, procesos mejor definidos y disminución del tiempo de gestión. Además permite tener un mejor control del dinero a la hora de ser utilizado para la compra y venta de artículos de primer orden.

- Conciliaciones Bancarias

Es un subsistema compuesto por dos módulos: Realizar conciliación y Gestionar partidas.

Realizar Conciliación

Es un módulo que le permite al conciliador realizar la conciliación de una cuenta específica de un determinado banco con una moneda asociada y una fecha de conciliación, primero de manera automática y luego manual en caso de quedar partidas pendientes y el conciliador así lo desee. Se puede cancelar la conciliación además de poder imprimir el Cierre de Conciliación.

Gestionar Partidas

El módulo Gestionar partidas le permite al conciliador ver todas las partidas pendientes y las partidas conciliadas en un rango de fecha determinado y de un banco especifico. Se pueden imprimir los listados de partidas pendientes y conciliadas.

Aplicación del procedimiento

Etapa de Planificación

Para la aplicación del proceso de pruebas propuesto en el capítulo anterior, se realizaron todas las actividades contempladas en cada etapa definida en el mismo. Para ello se comenzó por la primera etapa: Planificación. Como primera actividad de la misma, se realizó la reunión de inicio, donde se contó con la presencia del Administrador de la Calidad, el Analista del sistema, el Desarrollador y el Probador. En esta reunión se aprobó la Solicitud de pruebas enviada por el Analista del sistema al Administrador de la Calidad y posteriormente se comenzaron a definir puntos importantes recogidos en el Plan de prueba detallados a continuación:

Plan de Prueba

Proyecto: SAGEB

Producto: Conciliaciones Bancarias

Introducción

Este Plan de prueba se confecciona con el objetivo de planificar las Pruebas de Caja Blanca Estática realizadas al subsistema Conciliaciones Bancarias del proyecto Sistema Automatizado de la Gestión

Bancaria desarrollado bajo la tecnología Spring y el lenguaje de programación Java. El mismo ha sido solicitado por el Analista del sistema respectivamente.

Objetivo

Los objetivos del Plan de Prueba son:

- ✓ Comprobar la estructura del código en las clases que se encuentran implementadas hasta el momento en el subsistema Conciliaciones Bancarias del proyecto SAGEB.
- ✓ Verificar que la estructura del código cumpla con el estándar establecido en el SAGEB.
- ✓ Definir el cronograma de las pruebas de caja blanca estática.

Alcance y estrategia

El alcance de las pruebas está dado por la intención de evaluar la aplicación desde el punto de vista del código y verificar que cumpla con lo establecido en el Estándar de codificación del proyecto. La técnica de Prueba de Caja Blanca Estática que se seguirá es la automatizada, haciendo uso de la herramienta Checkstyle integrada al IDE de desarrollo Eclipse.

Diseño

En el Diseño se desarrollaron las actividades previstas para concluir la elaboración de Plan de Prueba. Se definieron los criterios de criticidad para detener y/o abortar las pruebas, se definió la cantidad de dos iteraciones de pruebas a realizar, se conformó el cronograma de pruebas, el ambiente de trabajo en cuanto a software y hardware quedó recogido en una tabal descrita posteriormente al igual que los roles involucrados en el proceso de pruebas y sus respectivas responsabilidades. Por último se detectaron posibles riesgos que pueden materializarse antes o durante la ejecución de las pruebas.

Roles y responsabilidades

Tabla 2 Roles y Responsabilidades.

Rol	Cantidad	Responsabilidad
Administrador de la calidad	1	✓ Realizar la reunión de inicio y cierre con todos los involucrados en el proceso. ✓ Confeccionar el Plan de pruebas y velar por su estricto

		cumplimiento. ✓ Revisar y asignar al proyecto las NC detectadas.
Analista del sistema	1	✓ Realizar la solicitud de las pruebas. ✓ Participar en las reuniones de inicio y cierre. ✓ Facilitar los artefactos necesarios al probador para la ejecución de las pruebas. ✓ Entregar las no conformidades detectadas al Desarrollador para que las corrija. ✓ Entregar el componente corregido para nuevas iteraciones.
Probador	1	✓ Ejecutar las pruebas de Caja Blanca Estática haciendo uso de la herramienta Checkstyle. ✓ Documentar la NC detectadas.

Recursos del sistema

Tabla 3 Recursos del sistema.

Hardware	Software
✓ 1 PC Cliente Pentium IV o superior. ✓ 1 GB de Memoria RAM o superior. ✓ 160 GB HDD.	✓ Sistema Operativo Windows XP SP 3. ✓ Eclipse 3.5 ✓ Plug-in de Checkstyle para Eclipse en su versión 5.5 (checkstyle-5.5-all.jar)

Cronograma

Tabla 4 Cronograma de pruebas.

No	Tarea	Fecha	Responsable	Participantes	Observaciones
1	Elaboración del Pre-Plan de prueba		Administrador de la calidad.	Administrador de la calidad	
2	Reunión de inicio.		Administrador de la calidad.	Administrador de la calidad, Analista del	

				sistema, Desarrollador y Probador.	
3	Preparación del entorno de pruebas.		Probador.	Probador.	
4	Ejecución de la Primera Iteración de las pruebas.		Probador.	Probador.	
5	Entrega de las NC al Analista.		Administrador de la calidad.	Administrador de la calidad y Analista del sistema.	
6	Respuesta de las NC.		Analista del sistema.	Analista del sistema y Desarrollador.	
7	Pruebas de regresión.		Probador.	Probador.	
8	Segunda Iteración de las pruebas.		Probador.	Probador.	
9	Evaluación de las pruebas.		Administrador de la calidad.	Administrador de la calidad.	
10	Reunión de cierre.		Administrador de la calidad.	Administrador de la calidad, Analista del sistema, Desarrollador y Probador.	

Criterios de criticidad

Criterios de Criticidad para Pruebas Detenidas (PD):

- ✓ No se configura el entorno debidamente para realizar las pruebas.
- ✓ No está lista la última versión del código que debe probarse.

- ✓ Se mantienen NC detectadas en iteraciones anteriores.

Criterios de Criticidad para Pruebas Abortadas (PA):

- ✓ No están presentes todos los elementos: componentes del sistema, producto o entregable (hardware, software, documentación y artefactos de apoyo).
- ✓ Excede el producto las 2 iteraciones establecidas en la planeación inicial.
- ✓ Se encuentran más de 50 No Conformidades.
- ✓ Se incumple con lo pactado en el Plan de Pruebas.
- ✓ Se excede el tiempo total de la prueba según lo planificado en el Cronograma pactado en el Plan de Pruebas.

Riesgos

Tabla 5 Riesgos.

Riesgo	**Tipo de Riesgo**	**Impacto**	**Descripción**	**Probabilidad**	**Efectos**
Personal mediocre.	Personal	Alto	Personal poco calificado para el rol que desempeña.	Media	Serio
Síndrome de la panacea.	Herramienta	Alto	“Esta herramienta ahorrará la mitad del trabajo.”	Alta	Serio
Desarrollo de las funciones incorrectas en el software.	Requerimiento	Alto	Implementar funcionalidades en las clases equivocadas.	Muy alta	Catastrófico
Continuos cambios en los requerimientos.	Organizacional	Medio	Se procede a implementar antes de diseñar el sistema.	Alta	Catastrófico

Introducción de una nueva tecnología.	Tecnológico	Alto	Migración hacia otro sistema operativo. Cambio del IDE de desarrollo.	Baja	Serio
Desarrollo orientado a la investigación.	Requerimiento	Medio	El proceso de desarrollo no se realiza de acuerdo a bases sólidas.	Muy alta	Catastrófico
Escatimar en la calidad.	Organizacional	Alto	No se tiene en cuenta la calidad durante el proceso de desarrollo del sistema.	Alta	Serio
Desvinculación del grupo de calidad con el proyecto.	Organizacional	Bajo	Los desarrolladores trabajan independientemente del grupo de calidad.	Bajo	Tolerable
"Maratones" de desarrollo.	Organizacional	Medio	Muchos desarrolladores trabajando sobre el mismo componente.	Medio	Tolerable
Ritmo de trabajo extremo.	Personal	Alto	Trabajar más horas de lo establecido sin descansar.	Alto	Catastrófico

Etapa de Ejecución

En la Ejecución de las pruebas, se tenía concebido realizar un máximo de 2 iteraciones, para ello se consultaron primeramente los criterios de criticidad. De este número de iteraciones previstas, ambas fueron realizadas con éxito, en la primera, se encontraron una serie de errores, que fueron corregidos posteriormente por el Desarrollador, de modo tal, que al realizar la segunda iteración, el código alcanzó un resultado de 0 No Conformidades, quedando ejecutadas las pruebas de regresión satisfactoriamente.

Análisis del código con Checkstyle

Del módulo Conciliaciones Bancarias se analizaron las 27 clases que lo componen, a 235 funcionalidades se le detectaron errores con un total de 843 alertas devueltas por la herramienta. Ver Tabla 9.

El módulo posee 27 javadocs y 85 comentarios en total.

Tabla 6 Clases analizadas por paquetes y cantidad de métodos y alertas encontradas.

Paquetes	**Clases analizadas**	**Cantidad de métodos y aletas detectadas**
Conciliación/dao/impl	EstadoNuestrasCuentasDaoStoreProcedure	1 método y 1 alerta
	MT950DAO	0 métodos y 13 alertas
Conciliación/domain	MT950	3 métodos y 24 alertas
Conciliación/facade/impl	GestionarMT950FacadeImpl	19 métodos y 49 alertas
	GestionarMT950Facade	4 métodos y 7 alertas
Conciliación/manager/impl	MT950ManagerImpl	20 métodos-51 alertas
	MT950Manager	5 métodos y 8 alertas
Conciliación/web/mvc/controller	MT950MultiActionController	12 métodos y 251 alertas
Gestionar partidas/dao	PartidasConciliadasDAOImpl	6 métodos y 19 alertas
	PartidasPendientesDAOImpl	3 métodos y 17 alertas
	PartidasConciliadasDAO	20 métodos y 31 alertas30
	PartidasPendientesDAO	18 métodos y 30 alertas
Gestionar partidas/domain	PartidasConciliadas	20 métodos y 43 alertas

	PartidasPendientes	18 métodos y 30 alertas
Gestionar partidas/facade/impl	GestionarPartidasFacadeImpl	24 métodos y 30 alertas
	GestionarPartidasFacade	4 métodos y 6 alertas
Gestionar partidas/manager/impl	GestionarPartidasManagerImpl	4 métodos y 6 alertas
	GestionarPartidasManager	21 métodos y 54 alertas
Gestionar partidas/report	CierreDataSource	5 métodos y 53 alertas
	ImprimirCierre	0 métodos y 1 alerta
	ImprimirPartidasConciliadas	1 método y 1 alerta
	ImprimirPartidasPendientes	1 método y 2 alertas
	PartidasConciliadasDataSource	5 métodos y 41 alertas
	PartidasPendientesDataSource	5 métodos y 44 alertas
Gestionar partidas/web	GestionarPartidasMultiActionController	12 métodos y 31 alertas

Etapa de Evaluación

Para evaluar los resultados de la ejecución de las Pruebas de Caja Blanca Estática en el Subsistema Conciliaciones Bancarias, se efectuó la aplicación de métricas obteniendo así el grado de mantenibilidad del código y finalmente se recogieron los resultados obtenidos. A continuación se muestra la aplicación de las métricas y posteriormente los resultados finales de la evaluación.

Métricas de Mantenibilidad

- De 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.

Tiempo medio en analizar un fallo

Fórmula: $X = \sum (Tout - Tin) / N$

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.

N = número total de fallos registrados.

- De 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.

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.

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

Frecuencia de fallos debidos a efectos colaterales producidos después de una modificación.

Fórmula: X = 1 - A / B

Siendo:

A = número de fallos debidos a efectos colaterales detectados y corregidos.

B = número total de fallos corregidos.

- De Facilidad de prueba o Examinabilidad: 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.

Pruebas sin esfuerzo

Fórmula: ∑(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 <= 300, la conducta del mantenedor, el usuario o el sistema de software es Correcta.

Si X > 300, 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.

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.

- De Conformidad de la mantenibilidad: 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.

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.

Métricas aplicadas

Después de analizar las métricas anteriormente expuestas se decidió aplicar Registrabilidad de cambios y Pruebas sin esfuerzo para evaluar la mantenibilidad del código probado. Para obtener mayor precisión en los resultados del análisis se utilizó el plug-in Classycle en su versión 1.4, facilitando así el cálculo de las métricas seleccionadas.

- Cambiabilidad o Facilidad de Cambio

Registrabilidad de cambios

La métrica Registrabilidad de cambios arroja resultados, que al ser interpretados, pueden medir la cambiabilidad y la estabilidad del sistema al mismo tiempo. Esta métrica será aplicada para medir el control de cambios ocurridos y determinar a partir de esto si el sistema se considera estable o no.

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.

Datos

A = 843 cambios a funciones

B = 235 funciones modificadas

Cálculo

X = A / B

X = 843 / 235

X = 3.6 aproximadamente.

Interpretación

0 <= X <= 1

En este caso X = 3.6 (fuera de rango) por tanto el sistema es inestable.

- Examinabilidad o Facilidad de Prueba

Pruebas sin esfuerzo

La métrica Pruebas sin esfuerzo arroja resultados que permiten medir la Facilidad de prueba en el sistema. Esta métrica será aplicada para medir la conducta del mantenedor o el sistema de software cuando se realizan las pruebas a este último.

Fórmula:

∑ (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 <= 300 seg / fallos, la conducta del mantenedor, el usuario o el sistema de software es Correcta.

Si X > 300 seg / fallos, la conducta del mantenedor, el usuario o el sistema de software es Incorrecta.

Datos

T = 1764 clases probadas * 5 segundos que tarda probar cada clase aproximadamente.

N = 122 fallos resueltos

Cálculo

∑ (T) / N = 8820 / 122 = 72.3 seg/fallos

∑ (T) = 1764 clases * 5 seg = 8820 seg * clases

Interpretación

0 <= 72.3 <= 300

La conducta del sistema de software es Correcta.

Informe final de evaluación

La última actividad en el proceso de pruebas es elaborar el Informe final de evaluación. En este informe quedaron reflejadas las No Conformidades registradas en la primera iteración de las pruebas y el resultado de la segunda iteración, donde no se registraron No Conformidades. Este informe incluyó el resultado de las métricas de mantenibilidad aplicadas, obteniendo a través de ello, el grado de mantenibilidad del subsistema Conciliaciones Bancarias del SAGEB, desarrollado bajo la tecnología Spring y el lenguaje de programación Java.

Resultados de la evaluación

Para la culminación del proceso de pruebas, se realizó la reunión de cierre, donde estuvieron presentes, nuevamente, todos los roles involucrados en la propuesta de solución. En esta reunión se hizo entrega por parte del Administrador de la Calidad del Informe final de evaluación al Analista del sistema y se concluyó que el resultado del proceso de pruebas fue satisfactorio.

Conclusiones del capítulo

Después de aplicada la propuesta de solución a las funcionalidades del Subsistema Conciliaciones Bancarias del proyecto SAGEB haciendo uso de la herramienta Checkstyle integrada al Eclipse se concluye que, según los datos arrojados tras aplicar las métricas escogidas para medir la mantenibilidad, el sistema es inestable en cuanto a su facilidad de cambio y posee una actitud correcta en cuanto a la facilidad de prueba. Por tal motivo queda demostrado que la propuesta de solución posee un aporte práctico y puede ser aplicable a los sistemas desarrollados bajo la tecnología Spring y el lenguaje de programación Java.

CONCLUSIONES GENERALES

Asignar responsabilidades a la hora de realizar las pruebas exigirá un mayor compromiso por parte de los involucrados para con el proceso de revisión del código fuente.

La integración de herramientas destinadas a analizar el código fuente estáticamente con los IDE de desarrollo utilizados en los proyectos, ahorrará tiempo y esfuerzo al equipo de desarrollo garantizando así una mayor calidad en el producto final.

La aplicación de métricas ofrecerá a los desarrolladores el nivel de complejidad, mantenibilidad y estandarización que poseen sus componentes.

La propuesta de solución tiene un aporte práctico y puede ser aplicable a los sistemas desarrollados bajo la tecnología Spring y el lenguaje de programación Java.

RECOMENDACIONES

Mantener un seguimiento activo sobre las actualizaciones de la herramienta que se propone en la solución.

Incentivar el estudio sobre otras herramientas para realizar Pruebas de Caja Blanca Estática sobre otros lenguajes de programación.

Aplicar la propuesta a los demás módulos del proyecto SAGEB y a otros proyectos que presenten semejantes características.

No se debe esperar al final para realizar las pruebas al software sino que se debe probar según se va desarrollando.

GLOSARIO DE TÉRMINOS

CALISOFT: Centro de Soluciones Informáticas.

CEIGE: Centro de Informatización de la Gestión de Entidades.

CVN: Control de versiones.

Deadlock: Punto muerto. En programación se refiere a los Interbloqueos.

Debugueo: Esta acción se denomina desensamblado. También hace posible la investigación y el estudio detallado del contenido interno de cualquier fichero.

Defecto: Es la manifestación de un error en el software.

Depuración: Proceso de corrección de los errores detectados en la revisión.

Escenario de prueba: Es el ambiente en el cual operará el software eventualmente.

Error: En la informática, un error de software (o *bug*, en inglés) es un fallo que se produce durante la creación de un programa de computadora. Esa equivocación, con el tiempo, puede aparecer en cualquier momento del uso del software, impidiendo su correcto funcionamiento.

Framework: Marco de trabajo

IDE: Entorno de Desarrollo Integrado.

Identación: Es un anglicismo (de la palabra inglesa *indentation*) de uso común en informática que significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente, lo que en el ámbito de la imprenta se ha denominado siempre como *sangrado* o *sangría*.

JDK: Máquina virtual de Java.

Legibilidad: Capacidad o posibilidad de ser leído, por su claridad.

Módulo: Un módulo se entiende como un componente software que cumple las siguientes características:

- ✓ Debe ser un bloque básico de construcción de programas.
- ✓ Debe implementar una función independiente simple.
- ✓ Podrá ser probado al cien por cien por separado.

- ✓ No deberá tener más de 500 líneas de código.

Monolítico: Conformado por una sola pieza.

No conformidades (NC): Problemas detectados en un artefacto según insatisfacción con el resultado final de un Elemento de Configuración, lo pactado con anterioridad con el cliente, o no cumplimiento de un requisito.

Plug-in: Módulos que se integran a un IDE de desarrollo.

Profiling: En ingeniería de software el análisis de rendimiento, comúnmente llamado *profiling*, es la investigación del comportamiento de un programa de computadora usando información reunida del análisis dinámico del programa (cuando este está corriendo) en oposición al análisis estático (análisis de código). La meta del análisis de rendimiento es determinar qué partes del programa se pueden optimizar para ganar velocidad u optimizar el uso de memoria.

Refactorización: "Refactorizar un software es modificar su estructura interna con el objeto de que sea más fácil de entender y de modificar a futuro, tal que el comportamiento observable del software al ejecutarse no se vea afectado." (Fowler).

SAGEB: Sistema Automatizado de la Gestión Bancaria.

Threads: En programación esta terminología se utiliza para referirse a los llamados *hilos de código.*

UCI: Universidad de las Ciencias Informáticas

Validación: Evaluación de un sistema o uno de sus componentes durante o al final de su desarrollo para determinar si satisface los requisitos.

Verificación: Determinar si los productos de una fase dada satisfacen las condiciones impuestas al inicio de la fase.

BIBLIOGRAFÍA

Artículos de revista

Yolanda González Fernando y Fernando de Cuadra García. “Calidad del software 1*”, Calidad de software (I), 2001, Pág. 54-60.

Elena Raja Prado. Actas de Talleres de Ingeniería del Software y Bases de Datos. “Casi todas las pruebas del software”. Vol. 1, No. 4, 2007, Pág.1-4.

Raúl Marticorena, Carlos López, Yania Crespo. Ingeniería de software. “Estudio de la distribución docente de pruebas del software y re-factoring para la incorporación de metodologías ágiles”. Pág. 247-254

Luis Vinicio León Carrillo. PRUEBA DE SOFTWARE. “Caracterización de la Prueba de Software”. Pág. 49.

Libros

Kan, S. (2002) Software Quality Metrics Overview. Metrics and Models in Software Quality Engineering, 2nd Edition. Addison Wesley Professional

Institute of Electrical and Electronics Engineers. (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.IEEE Std. 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, Los Alamitos, CA.

Roger S. Pressman (2002) Ingeniería del Software. Un enfoque práctico

IEEE (1998) IEEE Std. 1219-1998, Standard for Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA, 1998.

Fuente electrónica

Pressman, “Validación y Verificación”, Capítulo 13: Estrategia de Prueba, 7ma Edición, Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Bibliografia_general/Textos_Basicos/Ediciones_del_Pressman/Pressman_6ta_edicion/Pressman_Cap_13_Estrategia_Prueba.pdf

Pressman, “Validación y Verificación”, Capítulo 1: Software e Ingeniería de Software, 6ta Edición, Disponible en:

http://eva.uci.cu/file.php/161/Documentos/Materiales_basicos/Materiales_basicos_de_la_Unidad_1/Pressman_6ta_Edicion/Pressman_Cap_01_SW_e_ISW.pdf

Pressman, Capítulo 14: Conceptos de calidad, 7ma Edición, Disponible en:

http://eva.uci.cu/file.php/161/Documentos/Bibliografia_general/Pressman_7ma_edicion/18_Chapter_14._Quality_Concepts.pdf

Pressman, Capítulo 26: Gestión de la calidad, 6ta Edición, Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Bibliografia_general/Textos_Basicos/Ediciones_del_Pressman/Pressman_6ta_edicion/Pressman_Cap_26_Gestion_Calidad.pdf

Pressman, Capítulo 27: Software e Ingeniería de Software, 8va Edición, Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Bibliografia_general/Textos_Complementarios/Ediciones_del_Sommerville/Sommerville_8va_edicion/Cap_27_Gestion_de_la_Calidad.pdf

Pressman, “Verificación y Validación”, Parte 5, 8va Edición, Disponible en:

http://eva.uci.cu/file.php/161/Documentos/Bibliografia_general/Sommerville_8va_edicion/05_Part_5._Verification_and_Validation.pdf

Pressman, “Validation, verification and testing of computer software ”. Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_2/Comun/validation_verification_and_testing_of_computer_software_.pdf

Pressman, Capítulo 26: Gestión de la calidad, 7ma Edición, Disponible en:

http://eva.uci.cu/file.php/161/Documentos/Bibliografia_general/Pressman_7ma_edicion/20_Chapter_16._Software_Quality_Assurance.pdf

Pressman, Capítulo 24: Prueba de software, 8va Edición, Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Bibliografia_general/Textos_Complementarios/Ediciones_del_Sommerville/Sommerville_8va_edicion/Cap_23_Prueba_de_Software.pdf

Fillottrani, Pablo R., “Calidad en el desarrollo de Software”. Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_2/Comun/Calidad_en_el_Desarrollo_de_Software_Pablo_R._Fillottrani.pdf

Moreno, Jurisco, “Técnicas de Evaluación de Software”. Disponible en:

http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_2/Comun/Tecnicas_de_evaluacion_de_software_Jurisco-Moreno.pdf

Herramienta JLint. Disponible en: http://it.toolbox.com/wiki/index.php/Jlint

Herramienta Findbugs. Disponible en: http://findbugs.sourceforge.net/

Herramienta Sonar. Disponible en: http://www.sonarsource.org/downloads/

Ingeniería de pruebas de software. Disponible en: http://ingenieriadepruebasdelsoftware.blogspot.com/

Métricas de Mantenibilidad. Disponible en: http://cnx.org/content/m17464/latest/

Sociedad chilena de Ciencia de la computación. Disponible en:

http://www.dcc.uchile.cl/~mmarin/revista-sccc/sccc-web/Vol6/Art09.pdf

Folksemantic. Disponible en: http://www.folksemantic.com/visits/78640

Mantenibilidad del software. Disponible en:

http://es.scribd.com/doc/44878992/Mantenibilidad-Del-Software

Métricas de mantenibilidad. Disponible en:

http://ldc.usb.ve/~abianc/materias/ci4712/metricas.pdf

ISO 9126-3, http://mena.com.mx/gonzalo/maestria/calidad/presenta/iso_9126-3/

Raúl Expósito. ¿Qué es el análisis estático del código?, 2012, Disponible en: http://www.raulexposito.com

Miguel Matas, “Cómo definir un estándar de codificación y/o trabajo”, 2008. Disponible en: http://www.miguelmatas.es/blog/

Revisiones de código y estándares de codificación. Disponible en:

http://msdn.microsoft.com/es-es/default.aspx

Tesis

Ruiz, F. y Polo, M., Grupo Alarcos, Dpto. Informática, Escuela Superior de Informática, Universidad de Castilla - La Mancha, Ciudad Real, 2000-2001, http://alarcos.esi.uclm.es/per/fruiz/cur/mso/trans/s3.pdf

Elizabeth Quintas Sánchez. "Procedimiento para la realización de pruebas de unidad dentro el proyecto Sistema Único de Aduanas". Universidad de las Ciencias Informáticas. La Habana, 2010. 80 pág.

Dayanis Isaac Morales. "Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS". Universidad de las Ciencias Informáticas. La Habana, 2009. 115 pág.

Franklin Rivero Duharte. "Pruebas Unitarias de Software en la Plataforma J2EE". Universidad de las Ciencias Informáticas. La Habana, 2008. 120 pág.

Yurién Ricardo Fuentes Guerra y Ernesto Jordán Borjas. "Implementación de una herramienta para viabilizar el proceso de pruebas de caja blanca". Universidad de las Ciencias Informáticas. La Habana, 2008. 115 pág.

Lizzet Cabrera Montoya y Virgen Yuliet Acosta Hinojosa. "MÉTRICAS ESTANDARIZADAS INTERNACIONALMENTE, PROPUESTAS PARA EVALUAR LA CALIDAD DE LOS PRODUCTOS DE SOFTWARE". Universidad de las Ciencias Informáticas. La Habana, 2008. 106 pág.

Anel Alfonso Febles y Denia Francisca Alomá Benítez. "Perfeccionamiento de una estrategia basada en pruebas de Caja Blanca para los sistemas Registros Principales y Notarías Públicas del proyecto Registros y Notarías Fase II." Universidad de las Ciencias Informáticas. La Habana, 2010. 80 pág.

Andy Abelarde Silveira. "Procedimiento para la realización de pruebas de caja blanca en la UCID". Universidad de las Ciencias Informáticas. La Habana, 2010. 75 pág.

Yaimí Márquez Alpízar y Yenni Valdés Hechavarría. "Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico." Universidad de las Ciencias Informáticas. La Habana, 2007. 130 pág.

Francisco Ruiz y Macario Polo. "MANTENIMIENTO DEL SOFTWARE". UPM. 2007. 109 pág. MASTER EN INGENIERÍA DEL SOFTWARE.

Natalia Juristo, Ana M. Moreno, Sira Vegas. "TÉCNICAS DE EVALUACIÓN DE SOFTWARE". 2005. 126 pág.

Lic. Manuel José Barranco García. “APORTACIÓN A LAS TÉCNICAS DE EVALUACIÓN DE LA PRODUCTIVIDAD DEL SOFTWARE EN LOS ENTORNOS DE PROGRAMACIÓN”. Universidad Politécnica de Madrid, 1995. 151 pág. Disponible en: http://sinbad2.ujaen.es/sinbad2/files/TesisGrupo/TesisManuelBarranco.pdf

Arturo Henry Torres Zenteno. “Método de pruebas de sistema basado en modelos navegacionales en un contexto MDWE”. Universidad de Sevilla, 2008, 89 pág.

Pedro González Serrano. “Procedimiento para la aplicación de Pruebas de Caja Blanca en el grupo de calidad de FORTES.” Universidad de las Ciencias Informáticas. La Habana, 2011. 80 pág.

Dianelys Caridad Castañeda Orozco. “Definición del proceso de pruebas para los productos del CEIGE.” Universidad de las Ciencias Informáticas. La Habana, 2011. 84 pág.

Conferencias

Pablo R. Fillottrani. “Calidad en el Desarrollo de Software”, Segundo cuatrimestre 2007.

Ing. Roberto Hugo Vázquez, “Introducción a la calidad del software”, 2006.

María Clara Choucair Cárdenas. “Pruebas de software ¿la salvación, un proceso sin utilidad, trivial, simplemente una moda, o....?”, 2007

Dr. Eduardo A. RODRÍGUEZ TELLO. “Importancia de las pruebas de software”, 2011

Carlos Blanco Bueno. “Construcción y Pruebas de Software”, 2011.

Isabel Lamas Codesido. "Comparación de analizadores estáticos para código java", 2011.

Tayche Capote García (tcapote) Jefe Dpto. Docente. Subdirección Dpto. Pruebas de ISW_ Calisoft

ANEXOS

Anexo 2: Formato para la Solicitud de Prueba de Caja Blanca Estática

Fecha de la solicitud:

Centro:

Nombre del Proyecto:

Nombre del Producto(s):

Nombre del Componente:

Versión <>

Datos para el entorno de pruebas:

Características	PC Cliente
Capacidad disco duro (en GB)	
Memoria RAM	
Sistema Operativo	

Fecha de aceptación del producto según cronograma:

Breve descripción del proyecto:

Nombre y apellidos del Probador que revisará el componente:

Nombre y apellidos del solicitante:

Cargo:

Teléfono de localización:

Firma del Solicitante: _____

Anexo 4: Formato para el Plan de Riesgos

<Proyecto>

< Módulo>

Tabla de contenido

1. Alcance

2. Riesgos

Riesgo	Tipo de Riesgo	Impacto	Descripción	Probabilidad	Efectos
[Riesgo detectado.]	*[Clasificación del riesgo detectado (tecnológico, personal, herramienta, requerimiento, organizacional)]*	*[Grado de impacto del riesgo durante la ejecución de las pruebas. (alto, bajo o medio)]*	*[Detalles del riego detectado.]*	*[Probabilidad de ocurrencia del riesgo. (muy alta, alta, media o baja)]*	*[Clasificación del impacto del riesgo en (catastrófico, serio, tolerable e insignificante).]*

3. Gestión de Riesgos

Riesgo	Mitigación del riesgo	Monitoreo del riesgo	Administración del riesgo
[Riesgo detectado.]	*[¿Cómo se puede evitar el riesgo?]*	*[¿Qué factores se pueden vigilar que permitan determinar si el riesgo es probable?]*	*[¿Con que planes de contingencia se cuenta si el riesgo se vuelve realidad?]*

Anexo 6: Formato para el Registro de No Conformidades

Tipo
Anotaciones
Comentarios en el Javadoc
Convenciones de nombres
Cabeceras
Importaciones
Violaciones de tamaño
Modificadores
Bloques
Codificación
Diseño de clases
Duplicaciones
Filtros
Misceláneo
Otros

Fecha	Probador	Elemento	Etapa de detección	No	No Conformidad	Aspecto correspondiente	Tipo	Estado
[Fecha en que se realizó el registro.]	*[Nombre del probador que realizó el registro.]*	*[Elemento que se probó.]*	*[Número de iteración en que fue detectada la No Conformidad.]*	*[Número de la No Conformidad detectada].*	*[Descripción de la No Conformidad detectada.]*	*[Localización exacta donde fue encontrada la No Conformidad.]*	*[Clasificación de la No Conformidad según la herramienta.]*	*[Estado actual de la No Conformidad. Pendiente]*

Anexo 7: Formato para el Registro de Incidencias

<Centro>

<Departamento>

<Nombre del documento>

<Mes>

UCI	REGISTRO DE INCIDENCIAS		
Bitácora			
Período			
Resumen			
			% Uso
	Total horas aprovechadas en pruebas	0:00:00	0,00%
	Total horas perdidas	0:00:00	0,00%
	Horas perdidas por problemas tecnológicos	*0:00:00*	*0,00%*
	Horas perdidas por ausencia de especialistas	*0:00:00*	*0,00%*

Hora	Actividad	Tiempo (min	Tipo	Responsable

Anexo 8: Encuesta realizada a los Probadores del grupo de calidad del CEIGE

¿Es necesario corregir errores en el código? ¿Por qué?

¿Cómo se analiza el código?

¿Qué son las pruebas de caja blanca?

¿Qué es el análisis estático del código?

¿Qué herramientas se utilizan para el análisis estático del código?

¿Qué sabes de la calidad del código?

Anexo 10: Roles definidos en el Proceso de Mejoras

Roles	**Responsabilidades**
Administrador de la calidad	Elabora el Plan de Aseguramiento de la calidad. Elabora el plan de Mediciones. Participa en la elaboración del Plan de Monitoreo y en el monitoreo y análisis de las áreas de procesos. Elabora los planes de prueba. Participa en las revisiones técnicas formales de los artefactos. Participa en las revisiones con el cliente de los entregables. - Guía el diseño y ejecución de las pruebas internas. - Participa en el análisis y recolección de los datos para las mediciones. - Vela por el cumplimiento de las políticas de la organización y reglas bases del proyecto. - Colabora en las auditorías que se les realicen al proyecto. - Coordina y colabora con las pruebas de liberación externa al proyecto. - Crea una cultura de calidad en el proyecto. - Participa en las revisiones de inconsistencias y las monitorea hasta su cierre (REQM).
Analista del sistema	- Participa con el cliente y el usuario final recogiendo las entradas de los

	involucrados relevantes. - Captura los requisitos y definir las prioridades. - Lleva a cabo las actividades del análisis. - Realiza la especificación de requisitos. - Desarrolla el modelo de análisis del sistema. - Documenta el flujo de análisis. - Realiza el seguimiento de los requisitos durante todo el desarrollo del proyecto. - Participa en el diseño de la solución. - Diseña las pruebas. -Participa en la elaboración del Plan de Administración de Requisitos (REQM) - Determina los proveedores válidos de requisitos (REQM) - Crea y actualiza Matriz de Trazabilidad (REQM)
Desarrollador	- Convierte la especificación del sistema en código fuente ejecutable. - Desarrolla el diseño teniendo en cuenta la arquitectura. - Elabora las pruebas de unidad. - Desarrolla el prototipo de la interfaz de usuario. - Integra los componentes que forman parte de la solución. - Ejecuta los casos de prueba y generar no conformidades asociadas al mismo. - Registra y analiza los resultados de las pruebas.
Probador	Encargado de seguir los planes de pruebas. Ejecuta los casos de prueba y genera no conformidades asociadas al mismo. Registra los resultados de las pruebas. Analiza los resultados de las pruebas realizadas.