Universidad de las Ciencias Informáticas
Facultad 3

Título: “Evaluación estática del código para los proyectos desarrollados bajo las tecnologías Symfony y ExtJS en el CEIGE.”

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

Autora: Nailet Hernández Rodríguez

Tutor: Ing. Cristian Fernández López
Co-tutora: Ing. Iliannis Pupo Leyva

La Habana. Junio, 2012

DECLARACIÓN DE AUTORÍA

Declaro ser la autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Firma de la Autora

Nailet Hernández Rodríguez

_____ _____

Firma del Tutor Firma de la Co-tutora

Cristian Fernández López **Iliannis Pupo Leyva**

RESUMEN

El presente trabajo se realizó con el propósito de obtener una solución que permita evaluar el código, mediante pruebas de Caja Blanca Estática, en los proyectos desarrollados bajo las tecnologías Symfony y ExtJS en el CEIGE, de modo que contribuya a elevar la mantenibilidad de sus subsistemas. La base de la calidad del software es lograr la satisfacción del cliente. Con este propósito, se le deben realizar pruebas a cada producto de software para encontrar la mayor cantidad de defectos posibles y que este llegue a las manos del cliente con la calidad requerida. Este trabajo se centra en la elaboración de un procedimiento, que comienza con el establecimiento de precondiciones durante la implementación del código en los proyectos desarrollados en Symfony y ExtJS, seguido de un proceso de pruebas de Caja Blanca Estática, en el que se definen los subprocesos que guiarán la realización de las pruebas de Caja Blanca Estática, para que estas se efectúen con la mayor calidad posible. Para ello, se realizó una selección de los roles que deben intervenir en cada uno de los subprocesos, así como los artefactos necesarios para que estos se encuentren debidamente documentados, dando paso a cada una de las actividades a desarrollar. El procedimiento propuesto fue validado a través del criterio de expertos y de su aplicación en el subsistema Despacho No Comercial perteneciente al Sistema GINA. Se realizó un análisis de los resultados reales y se aplicaron métricas para medir el grado de mantenibilidad del subsistema evaluado en cuanto a la cambiabilidad, la estabilidad y la facilidad de pruebas. Mediante la correcta aplicación de este procedimiento, se garantiza que se realice la detección de errores en el código durante la fase de implementación, para ser posteriormente erradicados.

Palabras Claves

Proceso de pruebas, pruebas de Caja Blanca Estática, mantenibilidad.

Tabla de Contenido

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC´s) están en constante evolución y desarrollo y son las encargadas de estudiar, desarrollar, implementar, almacenar y distribuir la información utilizando hardware y software como medios informáticos. Las TIC´s, debido al desarrollo de las ciencias de la computación, brindan una gran variedad de herramientas de software que han hecho posible evaluar la calidad de software a través de una serie de pruebas realizadas durante el ciclo de vida de los mismos.

Las pruebas de software se llevan a cabo metódicamente, pudiendo ser planificadas por adelantado y ejecutadas una vez construido el código para asegurar que este se encuentre libre de errores.

Cuba también ha dedicado atención a la industria del software y cuenta actualmente con un Centro de Calidad para Soluciones Informáticas, llamado Calisoft, que se encarga de evaluar la calidad de este tipo de productos. En una entrevista realizada a la Jefa del Dpto. de pruebas de Ingeniería de Software (ISW) de Calisoft, se obtuvo la información de que en dicho centro no se realizan pruebas de Caja Blanca, sino que solo se tienen investigaciones sobre estas, específicamente de las pruebas de Caja Blanca Dinámicas, no siendo así el caso de las pruebas de Caja Blanca Estática, en las cuales estará centrado este trabajo. (Ver diagnóstico 1)

Básicamente, al aplicar pruebas de Caja Blanca Estática, se evalúan características de la calidad de software, una de ellas es la mantenibilidad, que está estrechamente relacionada con la facilidad de mantenimiento.

En Cuba, también existe la Universidad de las Ciencias Informáticas (UCI), donde se vincula estudio y producción de software y se cuenta con diferentes centros productivos, entre los que se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE). Dentro del CEIGE se encuentra el Dpto. de Soluciones para la Aduana, donde desarrollan varios proyectos informáticos, tal es el caso del Sistema de Gestión Integral de Aduanas (GINA), que se compone por diferentes subsistemas, y que se encuentra desarrollado bajo las tecnologías Symfony y ExtJS.

En el Dpto. de Soluciones para la Aduana, se realizó un diagnóstico (Ver diagnóstico 2) donde se detectó que en el código de estos aparecen con frecuencia referencias a variables con valor indefinido y violaciones de sintaxis y modelos de software, constituyendo, ambas, errores en el software. Además, se detectaron variables que nunca se usan (o código muerto), lo que provoca la reducción del rendimiento del

software y violaciones de los estándares de codificación que implican que el código alcance mayor complejidad y constituye un problema para la manteniblidad del mismo.

A partir de la problemática anterior, en el presente trabajo de diploma, se plantea el siguiente **problema a resolver:** En los proyectos del CEIGE, desarrollados bajo las tecnologías Symfony y ExtJS, existen dificultades con el rendimiento, la complejidad y la estandarización del código, lo cual afecta la mantenibilidad de los subsistemas.

Basado en el planteamiento del problema mencionado anteriormente, la investigación tiene como **objeto de estudio:** las Pruebas de Software de Caja Blanca Estática en el CEIGE, tomando como **campo de acción:** las Pruebas de Software de Caja Blanca Estática en sistemas desarrollados en Symfony y ExtJS.

El **objetivo general** de este trabajo es**:** Evaluar el código, mediante pruebas de Caja Blanca Estática, en los proyectos desarrollados bajo las tecnologías Symfony y ExtJS, de modo que contribuya a elevar la mantenibilidad de sus subsistemas. Por esta razón, la **idea a defender** que se plantea es la siguiente: Evaluar el código de proyectos desarrollados bajo las tecnologías Symfony y ExtJS, mediante pruebas de Caja Blanca Estática, permitirá elevar el grado de mantenibilidad de los mismos.

A modo de desglose del objetivo general de la investigación y en base a la idea a defender, se trazan los siguientes **objetivos específicos**:

- Identificar técnicas y herramientas para la realización de pruebas de Caja Blanca Estática a nivel nacional e internacional.
- Describir actividades y artefactos a desarrollar, y las herramientas a utilizar, para aplicar pruebas de Caja Blanca Estática a los subsistemas desarrollados en Symfony y ExtJS.
- Validar la solución propuesta a través del método de expertos y de su aplicación en uno de los productos del CEIGE.

Finalmente, se espera obtener como **posibles resultados:**

- Un procedimiento para la evaluación de la mantenibilidad en proyectos desarrollados bajo las tecnologías Symfony y ExtJS, mediante pruebas de Caja Blanca Estática.

- El grado de mantenibilidad de los subsistemas de GINA, desarrollados bajo las tecnologías Symfony y ExtJS, que hayan sido evaluados mediante pruebas de Caja Blanca Estática, en cuanto a la cambiabilidad, la estabilidad y la facilidad de prueba.

La presente investigación presenta la siguiente **estructura por capítulos:**

Capítulo 1: Fundamentación teórica.

En este capítulo se realizará una fundamentación teórica y/o estado del arte de las pruebas de software, particularmente las pruebas de Caja Blanca Estática. También serán abordados temas de interés que se encuentran relacionados con las pruebas de Caja Blanca Estática y la mantenibilidad.

Capítulo 2: Propuesta de solución.

En esta sección se lanzará, partiendo de la investigación realizada, la propuesta de solución que permita aplicar pruebas de Caja Blanca Estática y evaluar la mantenibilidad de los proyectos desarrollados en Symfony y ExtJS.

Capítulo 3: Validación de la propuesta de solución.

En este capítulo se validará la propuesta de solución a partir del criterio de expertos y la aplicación real de la solución propuesta en el capítulo anterior en el subsistema Despacho No Comercial de GINA, desarrollado en Symfony y ExtJS, y, posteriormente, se evaluará la mantenibilidad de dicho subsistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción al capítulo

En este capítulo, se realizará un análisis de conceptos y elementos que son de gran utilidad para elaborar una estrategia para el desarrollo y aplicación de las pruebas de Caja Blanca Estática en los subsistemas de GINA, desarrollados en los marcos de trabajo Symfony 1.2.8 y Ext 3.0, en el CEIGE. Estos conceptos y elementos son los referentes a: la calidad de software, las pruebas de software, las pruebas de Caja Blanca y, en específico, las de Caja Blanca Estática. Se tendrán en cuenta, además, otros aspectos de interés, como la mantenibilidad, las métricas, los estándares de codificación y la automatización de pruebas.

1.2 Calidad de software

La calidad del software puede definirse de muchas maneras. Una de las más limitadas, conocida como "calidad pequeña", define la calidad como la ausencia de defectos [1]. Para evaluarla de esta forma se emplean procedimientos estadísticos a partir de las tendencias de aparición de fallas durante la prueba de software. La calidad debe ser especificada, planificada, administrada, medida y certificada. Esto implica una visión integral que arroja la comprobación del software, con el fin de lograr un mayor grado de satisfacción y confianza del cliente hacia la organización productora de software. Constituye entonces las pruebas del software, tarea de alta prioridad para las empresas productoras. [2]

Al obtener la calidad requerida del software, se logra reducir su número de errores, o que estos queden eliminados totalmente, por lo que se alcanza una mayor fiabilidad en cuanto a las funciones que debe realizar el mismo, y se logra a la vez una mayor eficiencia e integridad de los datos, así como una mayor flexibilidad y reusabilidad. En la siguiente figura se muestran diferentes características del software que influyen sobre la calidad de software.

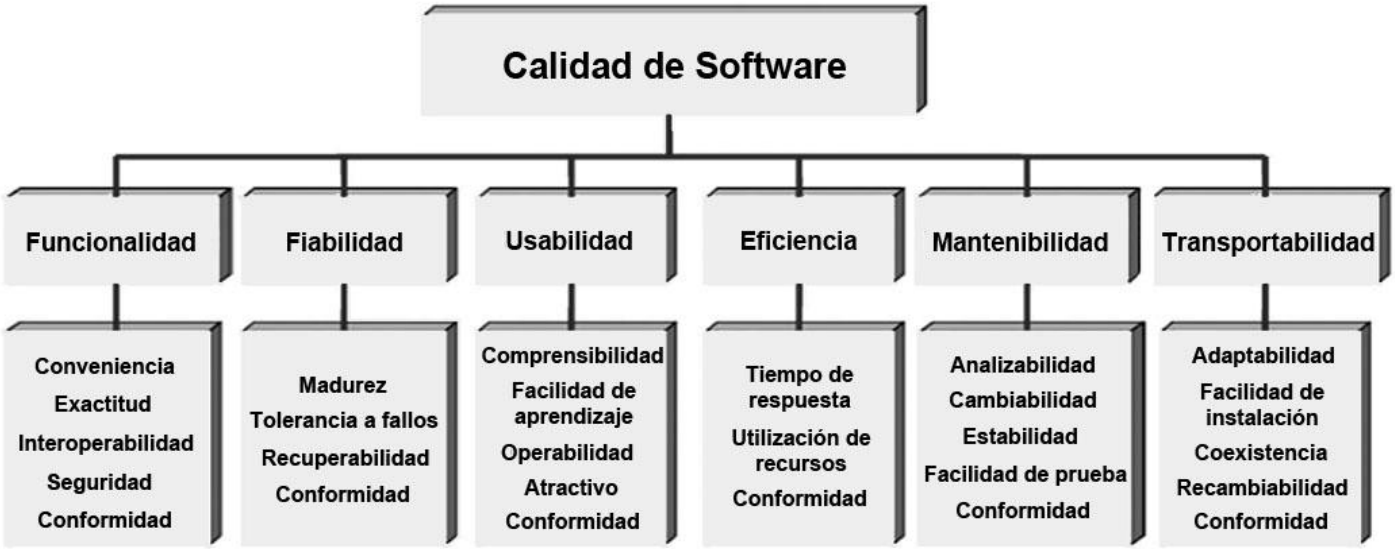


Fig.1 Calidad de un producto de software (modelo ISO 9126)

La calidad de software constituye un problema actual que afecta a los productores de software, así como a sus clientes. La demanda de este tipo de productos crece exponencialmente a escala mundial, debido al aumento de la informatización, y, lamentablemente, los desarrolladores le han brindado poco interés a la calidad de sus productos, pues en muchas ocasiones los clientes reciben el software sin que este haya pasado por la etapa de pruebas.

1.3 Pruebas de software

Para determinar el estado de la calidad de un producto de software es casi imprescindible llevar a cabo el proceso de pruebas. El proceso de pruebas, sus objetivos, métodos y técnicas a usar se describen en el Plan de prueba. En este proceso se ejecutan pruebas dirigidas a subsistemas o componentes del software, o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. Para realizar dichas pruebas, generalmente, se usan casos de prueba, especificados de forma estructurada mediante técnicas de prueba. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces. [3]

La prueba es una actividad fundamental en el desarrollo del software, así como en muchos otros procesos de desarrollo, ya que permite detectar la presencia de errores que pudieran generar salidas o

comportamientos indebidos durante la ejecución del software. Un concepto más específico dado por algunos desarrolladores de software es que las pruebas son:

Cualquier intento de demostrar que el software tiene propiedades por debajo de la calidad requerida. [4]

Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente. [5]

Otro concepto, importante a tomar en consideración, es el emitido por Pressman en su edición de 1998, que plantea lo siguiente: La prueba del software es un elemento crítico para la garantía de calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación. [6]

Partiendo de las definiciones anteriores, se puede ultimar que la prueba de software es una actividad en la que el sistema se ejecuta bajo condiciones específicas para demostrar si el software tiene, o no, la madurez necesaria para ser implantado.

Para obtener un software con la madurez necesaria se deben realizar las siguientes actividades:

- Revisiones: cada integrante del equipo de desarrollo revisa el producto que va generando.
- Inspecciones: revisión de cada producto por parte de colegas.
- Validaciones: el cliente revisa el producto para determinar si cumple con sus necesidades.

No se puede garantizar ni probar que un sistema jamás falle, sino que solo se puede demostrar que contiene faltas. No encontrar faltas no significa que la prueba haya sido exitosa. Sólo lo es si se han encontrado faltas. [3] Las faltas pueden estar presentes en el código o en el modelado, esto se sabrá en dependencia del tipo de pruebas que se le apliquen al software.

1.3.1 Estrategias de prueba

En el desarrollo de las pruebas se toma un conjunto de estrategias a seguir para lograr la mayor calidad que requiera el software y el cumplimiento de sus objetivos. Entre los diversos tipos de pruebas que existen se encuentran las pruebas unitarias, la cual contiene de forma general dos estrategias, por estructura (Caja Blanca), lo cual requiere conocer el diseño interno de la unidad, y por especificaciones (Caja Negra), la cual se basa sólo en especificaciones del comportamiento externo de la unidad. La mayoría de los autores recomiendan emplear pruebas de ambos grupos, cuando sea posible.

Una estrategia de prueba de software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a una construcción correcta del software. [7]

1.3.2 Personal responsable de realizar las pruebas

Para conocer quién debe responsabilizarse por crear las pruebas unitarias dentro el desarrollo del producto, deben tomarse en cuenta los criterios siguientes:

La prueba unitaria es la más “micro” escala de las pruebas; para probar funciones particulares o módulos de código. Típicamente hechas por desarrolladores y no por probadores (…). [8]

Mientras otras formas de probar pueden ser realizadas por probadores especializados, las pruebas unitarias deben ser escritas por el desarrollador (o una pareja de estos) que escriba cada clase, junto al código de la clase en sí. Un par de programadores es particularmente más efectivo, porque dos desarrolladores pueden pensar en un rango más amplio de escenarios de pruebas que uno. [9]

Independientemente de la forma en que sean escritas las pruebas unitarias, quien las haga debe tener las siguientes cualidades:

- Escéptico, especialmente sobre suposiciones (requiere evidencia concreta).
- Capaz de notar y seguir detalles extraños.
- Metódico y sistemático.
- Voluntad de experimentar, intentar cosas, ver lo que pasa.
- Habilidades orales y escritas que le permitan transmitir sus ideas.
- Capacidad de anticipar lo que otros no entenderán.

De modo independiente, las pruebas estructurales, generalmente, son la responsabilidad de los ingenieros de software que han desarrollado el producto, pero estos no deben hacerse cargo de las pruebas funcionales. En proyectos a gran escala las pruebas funcionales son la responsabilidad de un equipo de pruebas, formado por uno o varios técnicos, un coordinador de pruebas y un gestor de pruebas.

1.3.3 Métodos de prueba

Existen diferentes métodos para realizar las pruebas de software, entre los más importantes se encuentran la prueba de Caja Blanca, la prueba de Caja Negra y la prueba de la Estructura de Control.

La prueba de Caja Blanca es la mejor de su tipo para verificar que se recorran todos los caminos y detectar un mayor número de errores. Por su parte, las pruebas de la Estructura de Control amplían la cobertura de la prueba y mejoran la calidad de las pruebas de Caja Blanca. La Caja Negra brinda la posibilidad de cubrir la mayor parte de las combinaciones de entradas y lograr con ello un juego de pruebas más eficaz. Las pruebas mencionadas permiten probar cada una de las condiciones existentes en el programa, identificar claramente las entradas, salidas y estudiar las relaciones que existen entre ellas, permitiendo así maximizar la calidad de las pruebas y en dependencia del resultado se constará con un sistema más estable y confiable.

1.4 Pruebas de Caja Blanca

En las pruebas de Caja Blanca siempre se está observando el código, se conoce el contenido escrito dentro de la unidad a probar y se formaliza lo que sería la "cobertura del código" a probar. Para este tipo de pruebas existen métodos para diseñar los casos de prueba, con los que se pretende demostrar que las funciones del software son operativas, aceptando correctamente los datos de entrada y produciendo los resultados esperados, donde la estructura interna se comporta de la manera requerida.

Considerando una ruta como una combinación específica de condiciones manejadas por un programa, hay que señalar que no todos los errores de software se pueden descubrir verificando todas las rutas de un programa, hay errores que se descubren al integrar unidades del sistema y pueden existir errores que no tengan relación con el código específicamente.

1.4.1 Características de las pruebas de Caja Blanca.

En las pruebas de Caja Blanca, se pretende indagar sobre la estructura interna del código, omitiendo detalles referidos a datos de entrada o salida. Su objetivo principal es probar la lógica del programa desde el punto de vista algorítmico o estructural.

La prueba de Caja Blanca se basa en el diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante la prueba de la Caja Blanca el ingeniero de software puede obtener casos de prueba que: [2]

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdaderas y falsas.

- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que la prueba de Caja Blanca es considerada uno de los tipos de pruebas más importantes aplicables al software, pues arroja como resultado la disminución del número de errores existentes en los sistemas y, por ende, logra que exista una mayor calidad y confiabilidad en la codificación.

1.4.2 Ventajas y limitaciones de las pruebas de Caja Blanca

Las pruebas de Caja Blanca, proveen cinco ventajas básicas: [10]

- Fomentan el cambio: Las pruebas de Caja Blanca facilitan que el programador cambie el código para mejorar su estructura. Esto permite hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- Simplifican la integración: Las pruebas de Caja Blanca permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.
- Documentan el código: Las propias pruebas son documentación del código y permiten ver cómo utilizarlo.
- Separan la interfaz y la implementación.
- Los errores están más acotados y son más fáciles de localizar.

Como es lógico este tipo de pruebas no solo posee ventajas, también tiene las siguientes limitaciones:

- Las pruebas de Caja Blanca comprueban la estructura no la funcionalidad.
- Un programa puede estar bien, y sin embargo no servir a la función que pretende (lo que hace, lo hace bien, pero no es lo que se quería que hiciese).
- Se necesita comprobar la funcionalidad con pruebas de Caja Negra.

Se considera que las pruebas de Caja Blanca tienen una importancia trascendental en el ciclo de pruebas de un producto. Estas hacen que el programador se vuelva más consciente de sus decisiones de implementación, posibilitan la optimización del código y consiguen encontrar defectos u operaciones que podrían proporcionar problemas futuros.

1.4.3 Técnicas de pruebas de Caja Blanca.

Las técnicas utilizadas en pruebas de Caja Blanca son las siguientes:

- De Estructura de Datos Locales: Se centran en el estudio de las variables del programa. Buscan que toda variable esté declarada y que no existan variables con el mismo nombre, ni declaradas local y globalmente, que haya referencias a todas las variables y, para cada variable, analiza su comportamiento mediante comparaciones.

- De Cobertura Lógica: [11]

- *De Cobertura de Sentencias:* Comprueba que todas las sentencias se ejecuten al menos una vez.

- *De Cobertura de Decisión:* Ejecuta casos de prueba de forma tal que cada decisión se prueba al menos una vez a Verdadero (True) y otra a Falso (False).

- *De Cobertura de Condición:* Ejecuta un caso de prueba a True y otro a False por cada condición, teniendo en cuenta que una decisión puede estar formada por varias condiciones.

- *De Cobertura de Condición/Decisión:* Se realizan las pruebas de cobertura de condición y las de decisión a la vez.

- *De Condición Múltiple:* Cada decisión multicondición se traduce a una condición simple, aplicando posteriormente la cobertura de decisión.

- *De Cobertura de Caminos:* Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Los caminos en este caso no son más que una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

- *De Cobertura de bucles:* Los bucles son una fuente inagotable de errores, puesto que no son más que segmentos controlados por decisiones. Un bucle se ejecuta un cierto número de veces; pero ese número de veces debe ser muy preciso, y lo más normal es que, ejecutarlo una vez de menos o una vez de más, tenga consecuencias indeseables. Y, sin embargo, es extremadamente fácil equivocarse y redactar un bucle que se ejecute una vez de más o de menos. [12]

1.4.3.1 Prueba del Camino Básico

La prueba del camino básico es una técnica de prueba de Caja Blanca que permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control.

Un camino independiente es aquel que introduce por lo menos una sentencia de procesamiento (o valor de condición) que no estaba considerada. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. [13] Por último se diseñan los casos de prueba y se ejecutan los mismos.

- Grafo de flujo o grafo del programa: representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de éste. (Cada nodo representa una o más sentencias procedimentales y cada arista representa el flujo de control).
- Complejidad ciclomática: es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto de las pruebas, el cálculo de la complejidad ciclomática representa el número de caminos independientes del conjunto básico de un programa. Esta medida ofrece al probador de software un límite superior para el número de pruebas que debe realizar para garantizar que se ejecutan por lo menos una vez cada sentencia.

Con respecto a otras de su tipo, la técnica del Camino Básico ofrece una gran ventaja, puesto que el número requerido de pruebas es conocido por adelantado, por lo que el caso de prueba puede ser planeado y supervisado en mayores detalles, es válido aclarar que esto no sucede con la mayoría de las otras técnicas. [14]

1.4.3.2 Prueba de la estructura de control

Dentro de la prueba de la Estructura de Control se contempla la técnica del Camino Básico, mencionado anteriormente, pero, además, existen otras pruebas asociadas que permiten ampliar la cobertura de la prueba y mejorar su calidad. [15] Estas son:

- **Prueba de condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. En general los tipos de errores que se buscan en una prueba de condición, son los siguientes:

- Error en operador lógico (existencia de operadores lógicos incorrectos, desaparecidos, sobrantes).
- Error en variable lógica.
- Error en paréntesis lógico.
- Error en operador relacional.
- Error en expresión aritmética.

- **Prueba del flujo de datos:** [16] Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **Prueba de bucles:** [17] Es una técnica que se centra exclusivamente en la validez de las construcciones de bucles (bucles simples, anidados, concatenados y no estructurados).

Bucles simples: Se les aplica el siguiente conjunto de pruebas:

- Pasar por alto totalmente el bucle.
- Pasar una sola vez por el bucle.
- Pasar dos veces por el bucle.
- Hacer m pasos por el bucle con $m < n$ (donde n es el número máximo de pasos permitidos por el bucle).
- Hacer n - 1, n y n + 1 pasos por el bucle.

Bucles anidados: Si se empleara el mismo enfoque de prueba de bucles simples a los bucles anidados, el número de pruebas aumentaría considerablemente, por lo cual, se sugiere emplear el siguiente enfoque:

- Comenzar por el bucle más interior. Establecer o configurar los demás bucles con sus valores mínimos.
- Llevar a cabo las pruebas de bucles simples para el bucle más interior, mientras se mantienen los parámetros de iteración de los bucles externos en sus valores mínimos. Añadir otras pruebas para valores fuera de rango o excluidos.
- Progresar hacia fuera, llevando a cabo pruebas para el siguiente bucle, pero manteniendo todos los bucles externos en sus valores mínimos y los demás bucles anidados en sus valores típicos.

- Continuar hasta que se hayan probado todos los bucles.

Bucles concatenados: Estos bucles se pueden probar utilizando el enfoque de bucles simples, siempre y cuando cada uno de los bucles sea independiente del resto, de lo contrario, se debe emplear el enfoque de bucles anidados.

Bucles no estructurados: Siempre que sea posible estos bucles deben rediseñarse.

1.5 Pruebas de Caja Blanca Estática o Análisis Estático del Código

El análisis estático del código, tiene como objetivo mejorar la calidad del código fuente del software.

El análisis estático del código es el proceso de evaluar el software sin ejecutarlo. Este, por tanto, se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo. La idea es que, en base a ese código fuente, podamos obtener información que nos permita mejorar la base de código manteniendo la semántica original. El analizador estático de código, para ello, recibirá el código fuente de nuestro programa, lo procesará intentando averiguar qué es lo que queremos que haga y nos dará sugerencias con las que poder mejorar ese código.

1.5.1 Tipos de análisis estáticos del código

El análisis estático del código puede ser ejecutado de dos formas. Por un lado, está el análisis automático que realiza un programa de ordenador sobre el código y, por otro, está el análisis manual que realiza una persona. Cada uno de estos análisis persigue objetivos concretos:

- El análisis realizado por un programa de ordenador, o análisis automático, reduce la complejidad que supone detectar problemas en la base del código, ya que los busca, utilizando a unas reglas que tiene predefinidas.
- El análisis realizado por una persona, o análisis manual, se centra en apartados propios de la aplicación en concreto como, por ejemplo, determinar si las librerías que está utilizando el programa se están utilizando debidamente o si la arquitectura del software es la correcta.

Ambos, por tanto, son complementarios. El análisis automático se centra únicamente en facetas de más bajo nivel como la sintaxis y la semántica del código, funcionando este análisis en cualquier tipo de aplicación, mientras que el análisis manual se ocupa de facetas de más alto nivel como, por ejemplo, la estructura de la aplicación o su manera de trabajar con otros elementos externos como las librerías.

La unión de ambos tipos de análisis permitirá identificar los potenciales problemas a distintos niveles que generen la deuda técnica del proyecto. Se debe, por tanto, unificar ambas para poder mejorar la base del código fuente, lo cual repercutirá en una mejora tanto del desarrollo como del mantenimiento del software, incluso antes de llegar a ejecutarlo. [18]

1.5.2 Limitaciones del análisis estático del código

Los analizadores estáticos del código tienen sus limitaciones [18]:

- No nos permiten saber si el software va a hacer lo que se espera de él o no. Podemos analizar el código fuente y saber cómo mejorarlo muchísimo, pero no sabremos si hace lo que tiene que hacer u otra cosa totalmente distinta e inesperada.
- Al utilizar analizadores automáticos estos nos pueden devolver falsos positivos. Es posible que el analizador detecte como error algo que nosotros sabemos que está bien y que por alguna buena razón está programado así aposta.
- Están limitados al análisis de código sin llegar a ejecutarlo, de tal modo que necesitamos complementarlo con tests o con profiling para poder realizar mejoras en un ámbito mayor.

1.6 Mantenibilidad

Las pruebas de Caja Blanca Estática le proporcionan a los productos de software mayor facilidad de mantenimiento y de desarrollo, ya que minimizan la deuda técnica de cada proyecto, lo que contribuye a lograr una mantenibilidad adecuada para el software, partiendo de que esta última puede ser considerada una actividad de mantenimiento. Por tanto se puede afirmar que a mayor mantenibilidad, menores costes de mantenimiento y viceversa.

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 [19] Esta propiedad del software se puede definir como la medida cualitativa de la facilidad de comprender, corregir, adaptar y/o mejorar el software [20] Es un factor de calidad de revisión. Esfuerzo requerido para localizar y corregir fallas. Conjunto de atributos que se relacionan con el esfuerzo en realizar modificaciones. La mantenibilidad es la característica inherente de un elemento, asociada a su capacidad de ser recuperado para el servicio cuando se realiza la tarea de mantenimiento necesaria según se especifica. [21]

La mantenibilidad se define como la capacidad de un producto software para ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, en los requerimientos o en las especificaciones funcionales. Se subdivide en cinco subcaracterísticas: [22]

- Analizabilidad: Capacidad del producto software de diagnosticar sus deficiencias o causas de fallos, o de identificar las partes que deben ser modificadas.
- Cambiabilidad: Capacidad del producto software de permitir implementar una modificación especificada previamente. La implementación incluye los cambios en el diseño, el código y la documentación. Si el software es modificado por el usuario final, entonces, la cambiabilidad puede afectar a la operabilidad.
- Estabilidad: Capacidad del producto software de minimizar los efectos inesperados de las modificaciones.
- Facilidad de prueba: Capacidad del producto software de permitir evaluar las partes modificadas.
- Conformidad: Capacidad del producto software de satisfacer los estándares o convenciones relativas con la mantenibilidad.

Categorías de la mantenibilidad [23]

- 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 solo mejorar los algoritmos usados para hacerlos más eficientes.

Aspectos que influyen en la mantenibilidad del software: [24]

- *Proceso de desarrollo:*

La mantenibilidad debe formar parte del proceso de desarrollo del software, las técnicas utilizadas deben intervenir lo menos posible.

- *Documentación:*

En múltiples ocasiones ni la documentación, ni las especificaciones de diseño están disponibles, y por tanto, los costos del mantenimiento del software se incrementan debido al tiempo requerido para que un mantenedor[1] entienda el diseño del software antes de poder ponerse a modificarlo.

- *Comprensión de Programas:*

- La información disponible es incomprensible, incorrecta o insuficiente.
- La complejidad del software, de la naturaleza de la aplicación o de ambos.
- La confusión, mala interpretación u olvidos sobre el programa o sistema.

Propiedades de la mantenibilidad:

La mantenibilidad se puede considerar como la combinación de dos propiedades diferentes:

- Reparabilidad:

- Un sistema software es reparable si permite la corrección de sus defectos con una cantidad de trabajo limitada y razonable.
- Un producto software que consiste en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico.
- La reparabilidad de un producto software está influida por su fiabilidad, ya que al incrementarse esta última disminuye la necesidad de reparaciones.

- Flexibilidad: La flexibilidad es una característica tanto del producto software como de los procesos relacionados. El término de estos últimos significa que los procesos deben poder acomodarse a nuevas técnicas de gestión y organización, a cambios en la forma de entender la ingeniería, etc.

Efectos de los cambios en el software:

Después de aprobar el cambio solicitado en el software, el equipo de mantenimiento deberá implementar dicho cambio, las tareas a realizar son:

- Modificar los documentos y el código.
- Revisar los documentos y código modificados.

[1] Persona encargada de dar mantenimiento.

- Probar el código modificado.

Efectos de los cambios en el software sobre la mantenibilidad:

Algunos cambios en el software pueden reducir la mantenibilidad. Los que producen este efecto con mayor frecuencia son:

- Violar los estándares de codificación.
- Reducir la cohesión.
- Incrementar el acoplamiento.
- Incrementar la complejidad (Esencial).

El principal objetivo de la existencia de cualquier elemento/sistema realizado por el hombre es proporcionar utilidad, mediante la realización de una función requerida. De aquí que, una vez que se proporciona la funcionabilidad, la principal preocupación del usuario es alcanzar la disponibilidad y seguridad más elevadas posibles, con la menor inversión en recursos. La realización de cualquier tarea está relacionada con unos costes asociados, tanto en términos de coste de recursos de mantenimiento, como de coste de las consecuencias de no tener el sistema disponible para la operación. Por esto, los departamentos de mantenimiento son unos de los centros de mayor coste, requiriendo a la industria miles de millones de pesetas cada año, y de esa forma se han convertido en un factor crítico en la ecuación de rentabilidad de muchas compañías. Por tanto, puesto que las acciones de mantenimiento se vuelven cada vez más costosas, la ingeniería de mantenibilidad gana reconocimiento día tras día.

Está claro del breve análisis anterior sobre el papel e importancia de la mantenibilidad, que ésta representa uno de los determinantes principales de la consecución de los objetivos de los usuarios en lo relativo a disponibilidad, fiabilidad, coste de propiedad, reputación, etc. [25]

1.7 Métricas

La medición es esencial para cualquier disciplina de ingeniería. Las métricas de software se refieren a un amplio rango de medidas para el software de computadoras dentro del contexto de la planificación del proyecto de software, las métricas de calidad pueden ser aplicadas a organizaciones, procesos y productos los cuales directamente afectan a la estimación de costos.

Una métrica es usada para caracterizar cierta propiedad de un objeto o una clase de objeto; en la ingeniería de software, esta caracterización es cuantitativa, por ejemplo la métrica "líneas de código" caracteriza la propiedad de tamaño de un código fuente asociando un número con él.

Las métricas de software permiten estimar costo y esfuerzo en realizar un proyecto, evaluar la calidad del software realizado, predecir el tiempo invertido en mantenimiento de un programa o validar las mejoras prácticas para el desarrollo del mismo.

Si se desea estimar el costo y el esfuerzo entonces se necesita conocer el tamaño de un programa, si lo que se requiere es medir la complejidad, existe diversidad de métricas como McCabe y Haltstead, por lo que no siempre una sola métrica brinda toda la información que se necesita.

Las métricas se miden en: [26]

- Métricas directas: Son aquellas que se obtienen a través de un proceso de medición directo, es decir, que no involucra a ningún otro atributo.
- Métricas indirectas: Son aquellas que se obtiene a partir de métricas directas.

Y su clasificación consta de tres aspectos:

- Del proceso: Son los atributos de actividades relacionadas con el software. (Duración, esfuerzo, número de incidentes...).
- Del producto: Son los componentes, entregas o documentos resultantes de una actividad de proceso.
- De los recursos: Son entidades requeridas por una actividad de proceso.

1.7.1 Métricas de Mantenibilidad

Al elegir entre dos sistemas diferentes, ambos, desarrollados con el mismo lenguaje y que tienen el mismo tamaño. La elección vendría determinada por el más fácil de mantener (lo que implica menores costos de mantenimiento), pero, para saberlo, habría que conocer su mantenibilidad. [24]

Aproximaciones para medir la mantenibilidad:

Existen dos aproximaciones, para medir la mantenibilidad:

- Externa: La aproximación externa más directa para medir la mantenibilidad consiste en medir el proceso de mantenimiento, si es efectivo, entonces se asume que el producto es mantenible.

Medidas externas para la mantenibilidad: La característica clave de la mantenibilidad será la velocidad de implementar un cambio una vez que la necesidad de su realización está definida. Por esta razón se define una medida llamada Tiempo Medio para Reparación (MTTR).

- Interna: La aproximación alternativa se utiliza para identificar atributos internos. Ésta aproximación es más práctica puesto que las medidas pueden realizarse mucho más fácilmente, es importante recordar que la mantenibilidad nunca se puede definir sólo en términos de medidas internas.

Medidas internas de la mantenibilidad: Para determinar las medidas que más afectan a la mantenibilidad, la selección se debe realizar en combinación con medidas técnicas estadísticas, para identificar las medidas de producto que son las mejores para predecir los errores de interfaz con probabilidad de aparecer durante el mantenimiento.

El número ciclomático o cualquier otra medida sencilla resultan insuficientes por sí mismas como indicadores de la mantenibilidad, ya que capturan una visión muy reducida de la estructura y complejidad del software. Se han realizado múltiples estudios para determinar valores límites para otras medidas más sofisticadas.

¿Cómo medir la mantenibilidad?

La respuesta se encuentra entre los conceptos de calidad del software y de mantenibilidad. Por esta razón, los estándares ISO e IEEE proponen métricas de calidad para medir la mantenibilidad del software.

Las métricas de mantenibilidad no pueden medir el costo de realizar un cambio particular al sistema, sino que miden aspectos de la complejidad y la calidad de los programas, ya que existe una alta correlación entre la complejidad y la mantenibilidad.

Las métricas de mantenibilidad miden atributos relacionados con la conducta del mantenedor, el usuario o el sistema software, cuando dicho software se mantiene o se modifica durante la realización de pruebas o el mantenimiento. Estas pueden ser: [27]

- *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. Ejemplo: [22]
  - 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. Ejemplo: [27]

  - 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. Ejemplo:

  - 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. Ejemplos:

  - Pruebas sin esfuerzo

  Fórmula: Σ (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 adecuadas.

  B = número de oportunidades de prueba.

  - Reiniciabilidad de pruebas

  Fórmula: X = A/B

  Siendo:

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

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

- *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. Ejemplo:

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

Fórmula: X = 1 – (A / B)

Siendo:

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

B = número de elementos de conformidad totales.

1.8 Estándares de codificación

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del mismo es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de 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. [28] Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

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 de forma continuada un estándar de codificación bien definido, y se utilizan técnicas de programación apropiadas y, posteriormente, se efectúan revisiones del código de rutinas, caben 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 reforzar 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 iniciado el trabajo.

En el Dpto. de Soluciones para la Aduana, existe un estándar definido para la implementación de código en Symfony y otro para la implementación en ExtJS. En el caso del estándar definido para Symfony, se definen reglas para los componentes del framework, como las aplicaciones, módulos, acciones, plugins, etc., también para los nombres de las clases, la realización de pruebas, los nombres de las funciones y la declaración de variables. Aparecen además, una serie de estilos de codificación, donde se tienen en cuenta las políticas de llaves, de paréntesis, de sangrías, tabulaciones y espacios y la utilización de sentencias múltiples, etc. En el caso del estándar definido para ExtJS se definen pautas para la codificación en cuanto a la estructura y notación de carpetas y ficheros, las interfaces JS, los prototipos de clases, la configuración de los YML, la codificación Javascript, entre otras.

1.8.1 Ventajas de los estándares de codificación

El uso de los estándares de codificación implica innumerables mejorías, entre ellas:

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

Es por esto que la codificación de los módulos de los sistemas a desarrollar debe cumplir ciertos requisitos. Existen una serie de aspectos a tener en cuenta para comprobar los estándares según los criterios de programadores, aunque ellos son los que definen cuáles estandarizar y el estilo a aplicar, además de tener presente las características propias del lenguaje de programación, los recursos del mismo que se utilizarán y el tipo de programa que se debe implementar.

1.9 Automatización de Prueba

Según la Guía definitiva de Symfony 1.2, cualquier programador con experiencia en el desarrollo de aplicaciones web conoce de sobra el esfuerzo que supone probar correctamente la aplicación. Crear casos de prueba, ejecutarlos y analizar sus resultados es una tarea tediosa. Además, es habitual que los requisitos de la aplicación varíen constantemente, con el consiguiente aumento del número de versiones

de la aplicación y la refactorización continua del código. En este contexto, es muy probable que aparezcan nuevos errores.

Este es el motivo por el que la automatización de pruebas es una recomendación, aunque no una obligación, útil para crear un entorno de desarrollo satisfactorio. Los conjuntos de casos de prueba garantizan que la aplicación hace lo que se supone que debe hacer. Incluso cuando el código interno de la aplicación cambia constantemente, las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación. Además, este tipo de pruebas obligan a los programadores a crear pruebas en un formato estandarizado y muy rígido que pueda ser procesado por un framework de pruebas.

En ocasiones, las pruebas automatizadas pueden reemplazar la documentación técnica de la aplicación, ya que ilustran de forma clara el funcionamiento de la aplicación. Un buen conjunto de pruebas muestra la salida que produce la aplicación para una serie de entradas de prueba, por lo que es suficiente para entender el propósito de cada método. [29]

La automatización de pruebas es uno de los mayores avances en la programación. A raíz de esto, se han creado un conjunto de herramientas que ayudan a realizar las pruebas al código y mediante estas disminuir tiempo y esfuerzo.

1.9.1 Selección de la herramienta a utilizar para probar el código

Algunas herramientas han surgido a nivel mundial para las revisiones más rápidas y perfeccionadas del software, creadas por grupos de desarrolladores para analizar códigos en diferentes lenguajes. Partiendo de que los frameworks Symfony y ExtJS, utilizan como lenguaje de programación JavaScript y PHP, se realizó una búsqueda exhaustiva de herramientas aplicables a este trabajo y se encontraron algunas que analizan el código de forma estática en el lenguaje JavaScript, ellas son:

- **JSLint:** Herramienta creada por Douglas Crockford para medir la calidad de los códigos JavaScript en cuanto a errores de sintaxis y estructura del mismo sin necesidad de ejecutarlo.
- **JSHint:** Herramienta creada por la comunidad de desarrolladores JavaScript que constituye una ramificación o versión mejorada del proyecto original JSLint.
- **JavaScript Lint:** Herramienta para comprobar el código fuente de JavaScript en busca de errores comunes sin tener que ejecutarlo.

- **Doctor JS:** Se trata de un conjunto de herramientas de análisis estático de código JavaScript.

Los analizadores de código son herramientas que realizan la lectura del código fuente y devuelven observaciones o puntos en los que el código puede mejorarse desde la percepción de buenas prácticas de programación y código limpio. Originalmente el término Lint fue acuñado en 1977 por Stephen C. Johnson para denominar este tipo de herramientas, aunque, en la actualidad, no se utiliza este término comúnmente. [30]

A continuación se realiza la comparación entre las herramientas JSLint, JSHint y JavaScript Lint con el objetivo de seleccionar, finalmente, la más adecuada para ser utilizada en la continuidad de este trabajo:

JSLint es un analizador de código que permite mostrar puntos en los que el código no cumpla con determinadas reglas establecidas. Es "online", así que es posible probar con esta herramienta sin necesidad de instalarla. El manejo es bastante intuitivo, simplemente incluir el código JavaScript que deseemos probar, marcar las opciones de comprobación y configuración deseadas para un determinado tipo de análisis. El funcionamiento de JSLint es el siguiente: toma nuestro código, lo escanea y, si encuentra un problema, devuelve un mensaje describiéndolo y mostrando su ubicación aproximada. El aviso no tiene que ser necesariamente un error de sintaxis, sino que puede ser de estilo o estructural. Esto no significa que el programa no sea correcto, sino que ofrece otro punto de vista para mejorar su construcción a partir de patrones y recomendaciones tomadas de las Convenciones de Código el Lenguaje de Programación Javascript definidas en la especificación ECMAScript. JSLint tiene algunos inconvenientes. El más importante es la rigidez de sus estructuras durante la evaluación del código; esto hace que, cómo indican sus detractores, las aplicaciones terminen siendo tiranizadas por la herramienta**.** Todas las directrices que utiliza, pese a participar de la especificación correspondiente, son el resultado del criterio personal de una única persona, aunque dado el caso, se hable, posiblemente, del mejor conocedor del lenguaje. Es por esto que muchas veces se tratan de validar patrones que han sido definidos por otros desarrolladores obteniendo un sin fin de advertencias que sabemos, positivamente, que no van a producir error. Esta circunstancia hace que en muchas ocasiones, JSLint no resulte de utilidad al no validar códigos contrastados. Además es válido aclarar que esta herramienta arroja con frecuencia muchos falsos – positivos. [31]

Partiendo de los problemas que presenta la herramienta JSLint, la comunidad de desarrolladores JavaScript ha creado una ramificación del proyecto original denominada JSHint cuya finalidad es medir la

calidad de un código en el mundo moderno actual. La idea es prescindir de algunas reglas demasiado estrictas presentes en JSLint para ofrecer un conjunto más flexible de estilos y convenciones. Además, permite configurar una serie de parámetros dependiendo de nuestro estilo de programación para que los pase por alto cuando realice la validación:

- Permitir estamentos de debug y logging (los típicos 'console.log()').
- Tolerar eval.
- Exigir siempre igualdad estricta '==='.
- Permitir asignaciones dentro de los bucles if/for/while/do.

JSHint es también una herramienta "online", Open Source, que de forma gratuita y sin necesidad de registro previo, analiza el código Javascript. Esto constituye una utilidad pensada para códigos JavaScript, dirigidos a entornos de producción como navegador, ECMAScript5, Node.js o Rhino.
El manejo de JSHint es bastante intuitivo, simplemente incluir el código JavaScript que deseemos probar, marcar las opciones de comprobación y configuración deseadas para un determinado tipo de análisis y que encontraremos en un menú situado a la izquierda, y esperar a que la herramienta haga su trabajo. En caso de encontrar algún problema, JSHint devuelve un mensaje describiendo el error y mostrando su ubicación aproximada. [32]

Javascript Lint deriva del motor Javascript SpiderMonkey de Mozilla. Esta herramienta puede utilizarse directamente y personalizar las opciones de detección modificando un sencillo archivo de configuración. También puede ser integrado como herramienta externa en Visual Studio 2003/2005 y en el editor de Visual C++ 6.0 [33]. Javascript Lint está disponible para Windows y Linux. Además, es muy versátil y te permite utilizarlo de diversas formas: [34] puedes integrarlo en un IDE, ejecutarlo desde el explorador de Windows, utilizar la línea de comandos de su sistema, integrarlo en algún programa para Windows o hacerlo correr desde una página PHP.

Doctor JS se encuentra online en la siguiente dirección: http://doctorjs.org/try.html. Al acceder a la página, se introduce el código en el área de análisis, y, al dar clic en la lupa para analizar, aparecen los errores en el área de al lado en formato JSON. En lugar de simplemente analizar el código JavaScript, que utiliza una forma simple de interpretación abstracta para determinar qué símbolos se exportan. Doctor JS utiliza

jsctags, escrito completamente en JavaScript, el uso de módulos CommonJS, el marco node.js y el motor de Narciso.

Luego de analizar las herramientas anteriores y tratar instalarlas para verificar su funcionamiento, se determinó que la más apropiada para ser utilizada en este trabajo es JSHint, debido a que es una versión mejorada de JSLint, por lo que posee más ventajas, además de ser mencionada en la Internet como una de las herramientas más utilizadas para realizar pruebas de Caja Blanca Estática. También fue escogida por la facilidad de uso y de instalación que brinda sobre JavaScript Lint. No se optó por la utilización de Doctor JS, ya que, pese a que esta asegura que respeta la privacidad y no va a compartir el código con nadie, ni almacenar el código más tiempo de lo necesario para diagnosticar sus propios problemas si se producen, no se encuentra disponible para su descarga e instalación, sino que solo se utiliza directamente online y esto podría potenciar un peligro para la seguridad del código probado.

Para la realización de pruebas automáticas de Caja Blanca Estática en PHP existen diversas herramientas, solo que estas se encargan de analizar la seguridad del código, pero no su estructura, por lo tanto, no se ajustan al desarrollo de este trabajo. Se determinó entonces que el código PHP será evaluado manualmente a través de Listas de chequeo.

1.10 Tendencias de las pruebas de Caja Blanca Estática

Actualmente, en la UCI, se cuentan con investigaciones realizadas con respecto a las pruebas de Caja Blanca, donde se han propuesto diferentes estrategias y procedimientos a seguir para llevar a cabo este tipo de pruebas. Algunas de estas investigaciones son: “Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías”, “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”, “Procedimiento general para pruebas de Caja Blanca aplicando la técnica del Camino Básico”, “Procedimiento para la realización de pruebas de caja blanca en la UCID”, “Procedimiento para la realización de pruebas de caja blanca para FORTES”. De ellas, solamente “Procedimiento general para pruebas de Caja Blanca aplicando la técnica del Camino Básico”, trata acerca de las pruebas de Caja Blanca Estática, pero no se ajusta al marco de esta investigación, ya que el procedimiento que describe no es aplicable de forma automática y resulta muy tedioso para aplicarlo en el Dpto. de Soluciones para la Aduana. De modo general, estas investigaciones tampoco se basan en la aplicación de pruebas de Caja Blanca Estática en proyectos desarrollados en Symfony y ExtJS y ni en

determinar el grado de mantenibilidad de un proyecto mediante la aplicación de este tipo de pruebas, de ahí la necesidad de realizar un procedimiento para evaluar estáticamente el código de los subsistemas desarrollados en Symfony y ExtJS.

1.11 Conclusiones del capítulo

En este capítulo se realizó un análisis sobre las pruebas de software y otros factores en relación a estas. A partir del mismo, se concluye que las pruebas pueden encontrar fallos pero no aseguran que éstos no existen y que, de modo particular, analizar las Pruebas de Caja Blanca Estática y las herramientas existentes para realizar dichas pruebas y adaptarlas a las necesidades propias de cada proyecto, ahorrará tiempo y esfuerzo al equipo de desarrollo. Se concluye, además, que la mantenibilidad es un atributo de calidad de software de gran importancia ya que influye directamente en los costes y necesidades de mantenimiento y que puede ser mejorada mediante pruebas de software. Tras haber realizado un estudio profundo sobre las diferentes herramientas para aplicar pruebas de Caja Blanca Estática y de acuerdo a las características de Symfony y ExtJS, los cuales trabajan con los lenguajes de programación PHP y JavaScript, se decide proponer, como herramienta a utilizar, la herramienta JSHint para el análisis estático del código JavaScript y aplicar Listas de chequeo para el análisis del código PHP. Para medir la mantenibilidad en cuanto a la cambiabilidad, la estabilidad y la facilidad de pruebas, a partir de los resultados de las pruebas de Caja Blanca Estática arrojados por las Listas de chequeo y la herramienta JSHint, se decidió emplear las métricas Registrabilidad de cambios y Pruebas sin esfuerzo.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción al Capítulo

En el presente capítulo se describe la propuesta de un procedimiento para realizar Pruebas de Caja Blanca Estática en proyectos desarrollados en Symfony y ExtJS. El mismo está integrado por precondiciones establecidas para la fase de implementación del software y por un proceso compuesto por varios subprocesos. Dicho procedimiento, fue creado para dar cumplimiento a los objetivos propuestos en la investigación y se centra en definir los roles y responsabilidades que participarán en el proceso de pruebas, sus actividades correspondientes y los artefactos de entrada y salida que intervienen en el mismo.

2.2 Solución propuesta

2.2.1 Nombre

Procedimiento para la aplicación de Pruebas de Caja Blanca Estática a los proyectos desarrollados bajo las tecnologías Symfony y ExtJS.

2.2.2 Objetivos

- Establecer pasos para realizar pruebas de Caja Blanca Estática a los proyectos que se desarrollan en el CEIGE bajo las tecnologías Symfony y ExtJS.
- Especificar los roles que intervendrán, las actividades que se realizarán y los artefactos que se generarán a partir de esta solución.
- Evaluar la mantenibilidad de los proyectos desarrollados en Symfony y ExtJS mediante su aplicación, en cuanto a cambiabilidad, estabilidad y facilidad de pruebas.

2.2.3 Alcance

La solución comprende aquellos productos de software que se encuentren en desarrollo y utilicen como marcos de trabajo Symfony y ExtJS.

2.2.4 Responsables

La metodología RUP plantea que, durante la etapa de pruebas, deben intervenir cuatro roles fundamentales, estos son el de Especialista de Pruebas, Analista de Pruebas, Desarrollador de Pruebas y

Probador. Cada rol tiene diferentes responsabilidades en el proceso de pruebas, pero teniendo en cuenta un diagnóstico realizado en el Dpto. de Soluciones para la Aduana (Ver Diagnóstico 3), acerca de los roles existentes en los proyectos desarrollados bajo las tecnologías, para la cual se propone esta solución, se definió que la presente propuesta no comprenderá todos los roles que propone RUP, sino, que los roles participantes en el mismo serán los siguientes:

- Planificador/Líder de gestión: Participa en la reunión de inicio y en la reunión de cierre de las pruebas. Es el responsable de administrar los recursos para las pruebas y monitorear las acciones correctivas luego del subproceso de evaluación. También es el encargado de enviar la Solicitud de pruebas (Ver Anexo 1) al Administrador de la Calidad.
- Administrador de la Calidad: Participa en la reunión de inicio y en la reunión de cierre de las pruebas. Es el responsable de planificar y diseñar las pruebas. Para lograr una correcta planificación y administración de los recursos, este deberá elaborar el Plan de Pruebas (Ver Anexo 2). Además, deberá aplicar las métricas de mantenibilidad y elaborar el Informe final de evaluación. Participa en cada iteración durante el proceso de pruebas y evalúa en cada subproceso los resultados de cada ciclo.
- Desarrollador de PHP y Desarrollador de Interfaz de Usuario (IU): Participan en la reunión de inicio y en la reunión de cierre de las pruebas. Participan también en la realización del Plan de pruebas. Deben estar presentes en cada subproceso de pruebas con el objetivo de apoyar el proceso y corregir las No Conformidades detectadas durante las iteraciones de las pruebas.
- Probador: Es el responsable de ejecutar las principales pruebas de software y obtener los resultados de estas directamente. Se encarga de evaluar el código de forma manual, utilizando las Listas de chequeo (Ver Anexo 3 y Anexo 4), elaboradas a partir de los estándares de codificación de ExtJS (Ver Anexo 5) y Symfony (Ver Anexo 6), y también evaluará el código de forma automatizada, utilizando la herramienta JSHint y aplicando los Criterios de criticidad que determinarán el estado de las pruebas. Deberá elaborar, posteriormente el Registro de No Conformidades (Ver Anexo 7), donde se plasman las No Conformidades detectadas durante la ejecución de las pruebas y un Registro de incidencias (Ver Anexo 8), donde se recogen las incidencias o afectaciones ocurridas durante las pruebas. El artefacto Registro de NC y la tabla mencionada anteriormente se utilizarán para determinar los valores de entrada para las métricas. El Probador participa en cada iteración durante el proceso de pruebas.

2.2.5 Términos

Artefacto: Producto tangible del proyecto que es producido, modificado y usado en las actividades.

Actividad: Tarea que tiene un propósito y es asignada a un rol.

Rol: Función que una persona desempeña en una situación determinada.

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

2.2.6 Referencias

- IPP 1000:2008 Elaboración y aprobación de los procedimientos y lineamientos para la actividad productiva.
- Estrategia para la aplicación de Pruebas de Caja Blanca y Caja Negra al proyecto Registros y Notarías.
- Procedimiento para la realización de pruebas de caja blanca en la UCID.
- Propuesta de Procedimiento y Herramientas para realizar Pruebas de Caja Blanca en el Área Temática APS.
- Definición del proceso de pruebas para los productos de CEIGE.

2.3 Procedimiento para la aplicación de pruebas de Caja Blanca Estática en los proyectos desarrollados bajo las tecnologías Symfony y ExtJS

Durante la fase de implementación del software, el desarrollador debe seguir las siguientes precondiciones:

- Aplicar los estándares de codificación especificados en su proyecto.
- Realizar evaluaciones internas al código antes de entregarlo al grupo de trabajo encargado de la calidad en su centro.

Posteriormente, el código debe pasar por un proceso de pruebas de Caja Blanca Estática que está compuesto por diferentes subprocesos que permiten que este sea realizado correctamente.

Los subprocesos principales que se proponen seguir en este proceso, definido para proyectos desarrollados en Symfony y ExtJS, son la Planificación, la Ejecución y la Evaluación de las pruebas de Caja Blanca Estática, tal y como se muestra a continuación:

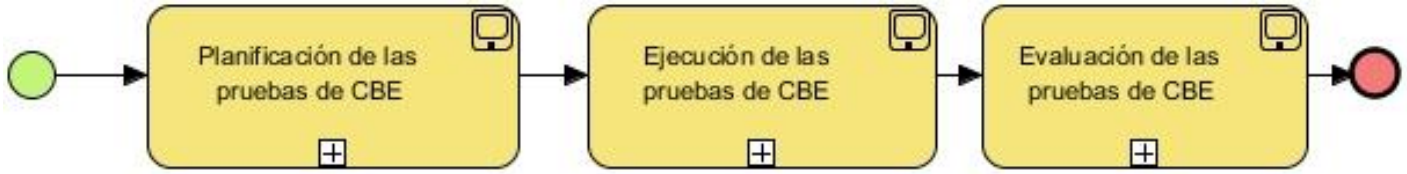


Fig. 2 Proceso de pruebas de Caja Blanca Estática para proyectos desarrollados en Symfony y ExtJS.

2.3.1 Planificación de las pruebas de Caja Blanca Estática en GINA

Subproceso de Planificación			
El objetivo principal de este subproceso es definir los roles y recursos del sistema para un apropiado entorno de pruebas, además de una estrategia adecuada.			
Roles	**Actividades**	**Artefactos de entrada**	**Artefactos de salida**
- **Planificador/Líder de gestión.** - **Administrador de la Calidad.** - Desarrolladores de IU. - Desarrollador de PHP. - Probador.	- Revisar solicitud. - Coordinar reunión de inicio de las pruebas. - Rectificar solicitud - Diseño de las pruebas (subproceso)	- Solicitud de pruebas.	- Plan de pruebas.

En el subproceso Planificación de las pruebas de Caja Blanca Estática en GINA, inicialmente, el Planificador/Líder de gestión envía la Solicitud de pruebas al Administrador de la Calidad, la cual consiste en un documento que especifica la fecha de la solicitud, el nombre del proyecto y del producto, los subsistemas a probar y la cantidad de clases o métodos a probar en cada subsistema, los requerimientos de software y hardware para realizar las pruebas y la firma del solicitante, etc. La Solicitud de pruebas es revisada por el Administrador de la Calidad, si este encuentra deficiencias en la misma le envía las deficiencias encontradas al Líder de Gestión nuevamente, de lo contrario, si la solicitud no presenta

deficiencias, el Administrador de la Calidad convoca a una reunión con todo el personal inmiscuido en la realización de las pruebas. En este caso, participan todos los roles definidos para este procedimiento, siendo ejercidas las funciones principales por el Líder de Gestión y el Administrador de la Calidad. Posteriormente se pasa al subproceso de Diseño, el cual servirá de apoyo a la Planificación, ya que en este, el Administrador de la Calidad realizará actividades muy importantes para conformar el Plan de pruebas, el cual debe ser aprobado por todos los presentes en la reunión. El Plan de pruebas consiste en un documento que contendrá una estrategia para la aplicación de las pruebas de Caja Blanca Estática, un cronograma correspondiente a su aplicación, los roles que intervendrán en el proceso con sus respectivas responsabilidades y las herramientas que serán empleadas para su ejecución.

Representación gráfica

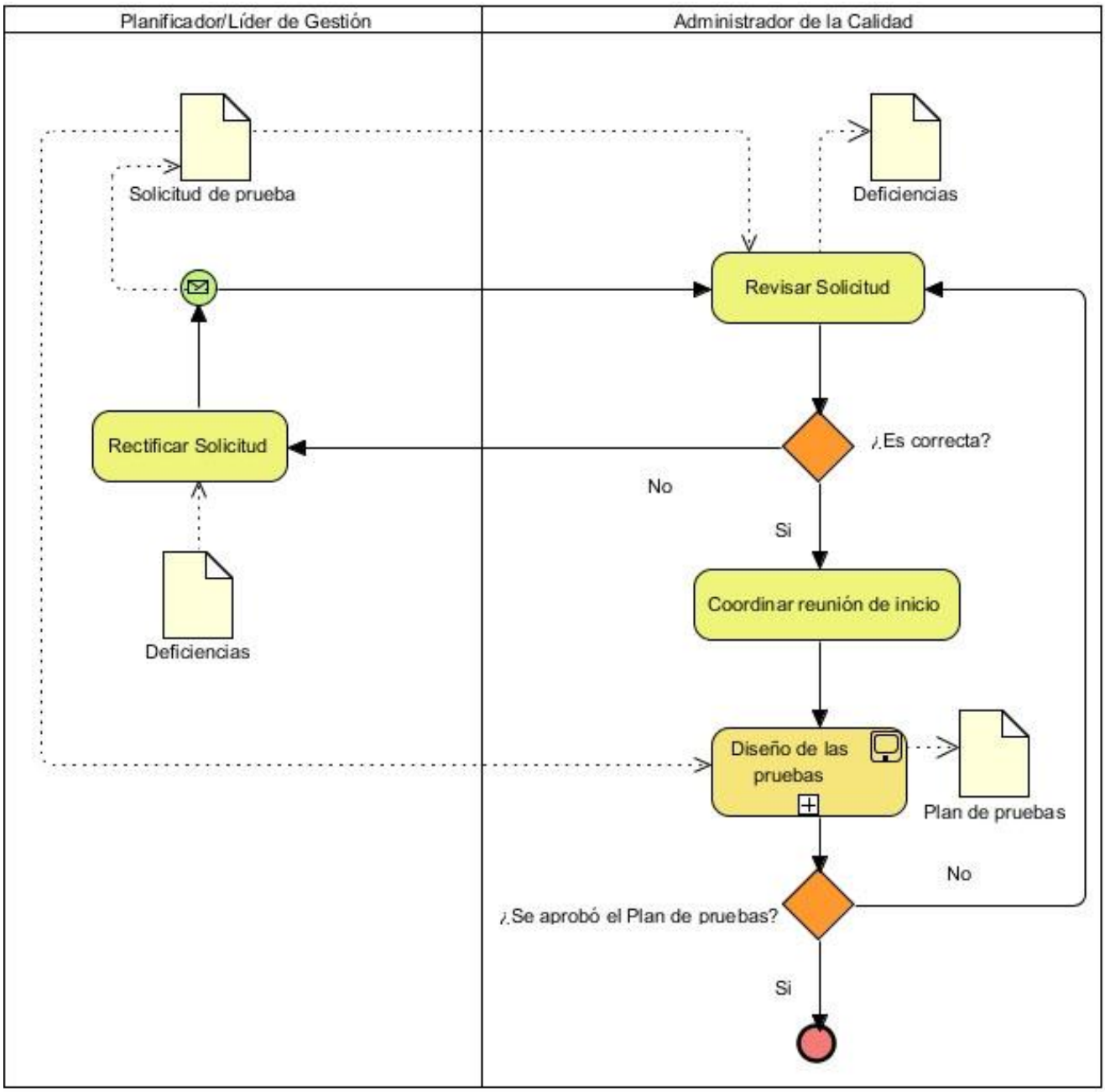


Fig. 3 Planificación de las pruebas de Caja Blanca Estática en GINA.

2.3.2 Diseño de las pruebas de Caja Blanca Estática en GINA

Subproceso de Diseño			
El objetivo principal de este subproceso es configurar el entorno de pruebas y elaborar el cronograma de pruebas.			
Roles	**Actividades**	**Artefactos de entrada**	**Artefactos de salida**
- **Administrador de la Calidad.**	- Identificar objetivos. - Definir si las pruebas serán realizadas manual o automáticamente (o ambas). - Configurar el entorno. - Elaborar Cronograma de pruebas.	- Requerimientos.	- Cronograma de pruebas.

En el subproceso Diseño de las pruebas de caja Blanca Estática en GINA, se identificarán los objetivos de las pruebas, se definirá si las pruebas serán realizadas manual o automáticamente (o de ambas formas) y se configurará el entorno de pruebas, teniendo en cuenta los requerimientos existentes para ello. Además, se elaborará el cronograma de pruebas, el cual no constituye un artefacto independiente, sino que estará contenido dentro del Plan de pruebas, y consiste en una planificación, que indicará el momento en que serán realizadas las pruebas.

Representación gráfica

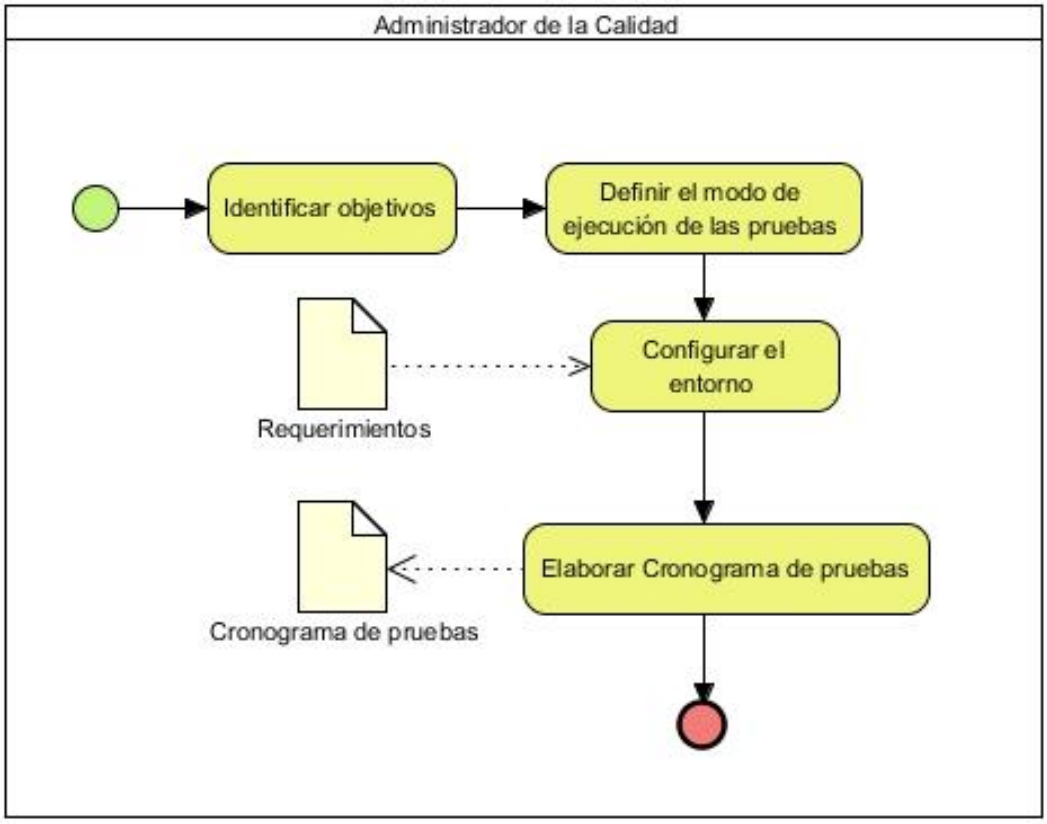


Fig. 4 Diseño de las pruebas de Caja Blanca Estática en GINA.

2.3.3 Ejecución de las pruebas de Caja Blanca Estática en GINA

Subproceso de Ejecución			
El objetivo principal de este subproceso es evaluar el código, manual y automáticamente y registrar las No Conformidades encontradas en el mismo.			
Roles	**Actividades**	**Artefactos de entrada**	**Artefactos de salida**

- **Administrador de la Calidad.** - **Desarrollador de IU.** - **Desarrollador de PHP.** - **Probador.**	- Evaluar el código manualmente mediante Listas de chequeo. - Evaluar el código automáticamente mediante la herramienta JSHint. - Realizar pruebas de regresión. - Corregir las No Conformidades registradas. - Elaborar Registro de incidencias.	- Cronograma de pruebas. - Estándares de codificación definidos en GINA para Symfony y ExtJS. - Listas de chequeo para cada código. - Criterios de criticidad. - Código de los subsistemas a prueba.	- Registro de No Conformidades. - Registro de incidencias de las pruebas.

En el subproceso Ejecución de las pruebas de Caja Blanca Estática en GINA, se procede a efectuar las pruebas de Caja Blanca Estática con el objetivo de detectar los errores existentes en el código. En este subproceso, participarán los desarrolladores de IU y PHP, el Administrador de la Calidad y el Probador, siendo el Probador el encargado de evaluar el código y elaborar el Registro de No Conformidades por cada iteración. Los desarrolladores de IU y PHP apoyarán el proceso mediante la aclaración de dudas en cuanto al código y la corrección de las No Conformidades detectadas y el Administrador de la Calidad elaborará el Registro de incidencias ocurridas durante la ejecución de las pruebas.

El Probador deberá evaluar el código de forma manual, mediante Listas de chequeo, las cuales estarán compuestas por una serie de interrogantes realizadas a partir de los estándares de codificación definidos para Symfony y ExtJS en GINA para verificar de forma manual si el código cumple o no con dichos estándares. Además, deberá evaluar el código mediante el uso de la herramienta JSHint, teniendo en cuenta los criterios de criticidad, los cuales establecerán parámetros para declarar el producto en estado crítico de terminación. Luego de cada iteración el Probador deberá elaborar el Registro de No Conformidades que contendrá los errores encontrados durante la ejecución de la pruebas, clasificados en

significativos o no significativos y, posteriormente, se determinará si será necesario que se realicen pruebas de regresión para verificar si los errores detectados anteriormente han sido corregidos y se repetirá esta parte del proceso según se defina en el Plan de Pruebas. En caso de que se deban realizar pruebas de regresión, los desarrolladores de IU y PHP serán los encargados de corregir las No Conformidades detectadas. Por último, el Administrador de la Calidad deberá elaborar un Registro de incidencias, que contendrá el total de horas perdidas por incidencias ocurridas durante la ejecución de las mismas.

Representación Gráfica

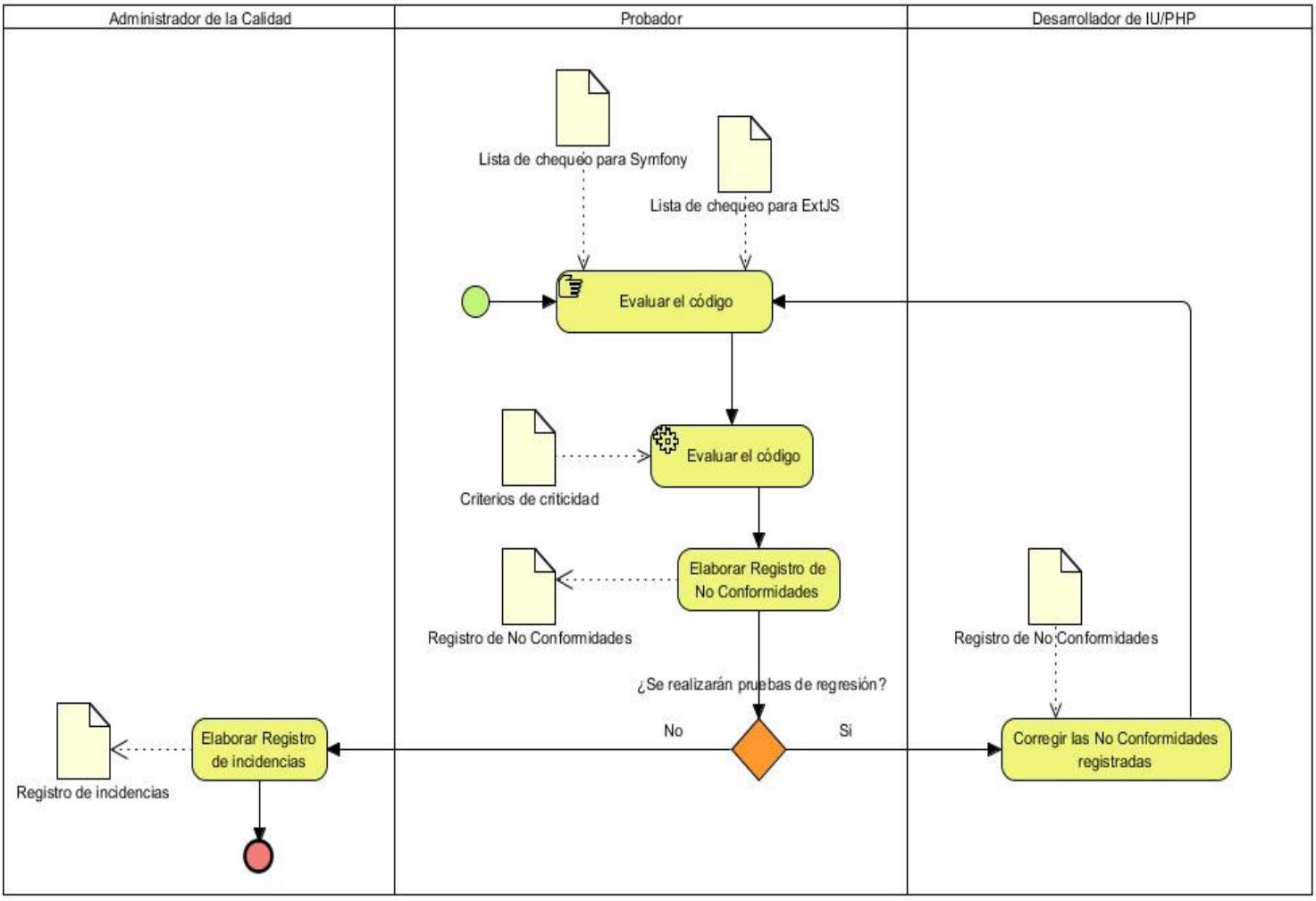


Fig. 5 Ejecución de las pruebas de Caja Blanca Estática en GINA.

2.3.3 Evaluación de las pruebas de Caja Blanca Estática en GINA

<table>
<tr><th colspan="4">Subproceso de Evaluación</th></tr>
<tr><td colspan="4">El objetivo de este subproceso es obtener el grado de mantenibilidad de los subsistemas probados a partir de la aplicación de métricas de mantenibilidad y reflejar este resultado en el Informe final de evaluación.</td></tr>
<tr><th>Roles</th><th>Actividades</th><th>Artefactos de entrada</th><th>Artefactos de salida</th></tr>
<tr><td>- Planificador/Líder de gestión.
- Administrador de la Calidad.
- Desarrollador de IU.
- Desarrollador de PHP.
- Probador.</td><td>- Aplicar métricas de mantenibilidad.
- Elaborar Informe de evaluación.
- Realizar la reunión de cierre de las pruebas.</td><td>- Registro de No Conformidades.</td><td>- Informe final de evaluación.</td></tr>
</table>

En el subproceso Evaluación de las pruebas de Caja Blanca Estática en GINA, a partir de los resultados de las pruebas y para hacer el proceso de pruebas de Caja Blanca Estática más completo y darle cumplimiento a uno de los objetivos de este trabajo, se propone aplicar métricas para evaluar la mantenibilidad del código probado. Las métricas que seleccionadas para ser aplicadas, en este caso, son las siguientes:

- Para medir la cambiabilidad y estabilidad del código se empleará la métrica Registrabilidad de cambios.
- Para medir la facilidad de prueba del código se empleará la métrica Pruebas sin esfuerzo.

Ambas métricas permiten evaluar subcaracterísticas de la mantenibilidad, a partir de las cuales se determinará si el sistema es mantenible o no. Estas métricas serán aplicadas a partir de los valores de entrada para las mismas, arrojados luego de ser ejecutadas las pruebas, que hayan sido registrados en el Registro de No Conformidades.

En este subproceso, se elaborará, además, el Informe final de evaluación de los subsistemas probados, teniendo en cuenta el Registro de No Conformidades. Dicho informe contendrá el resumen de evaluación, teniendo en cuenta todas las iteraciones realizadas, y el grado de mantenibilidad de los subsistemas probados. Todas las actividades mencionadas anteriormente, serán llevadas a cabo por el Administrador de la Calidad, quien, además, coordinará la reunión de cierre de las pruebas. En esta reunión, participarán, nuevamente, todos los roles inmersos en el proceso y el Administrador de la Calidad deberá entregar el Informe final de evaluación al Líder de gestión.

Representación Gráfica

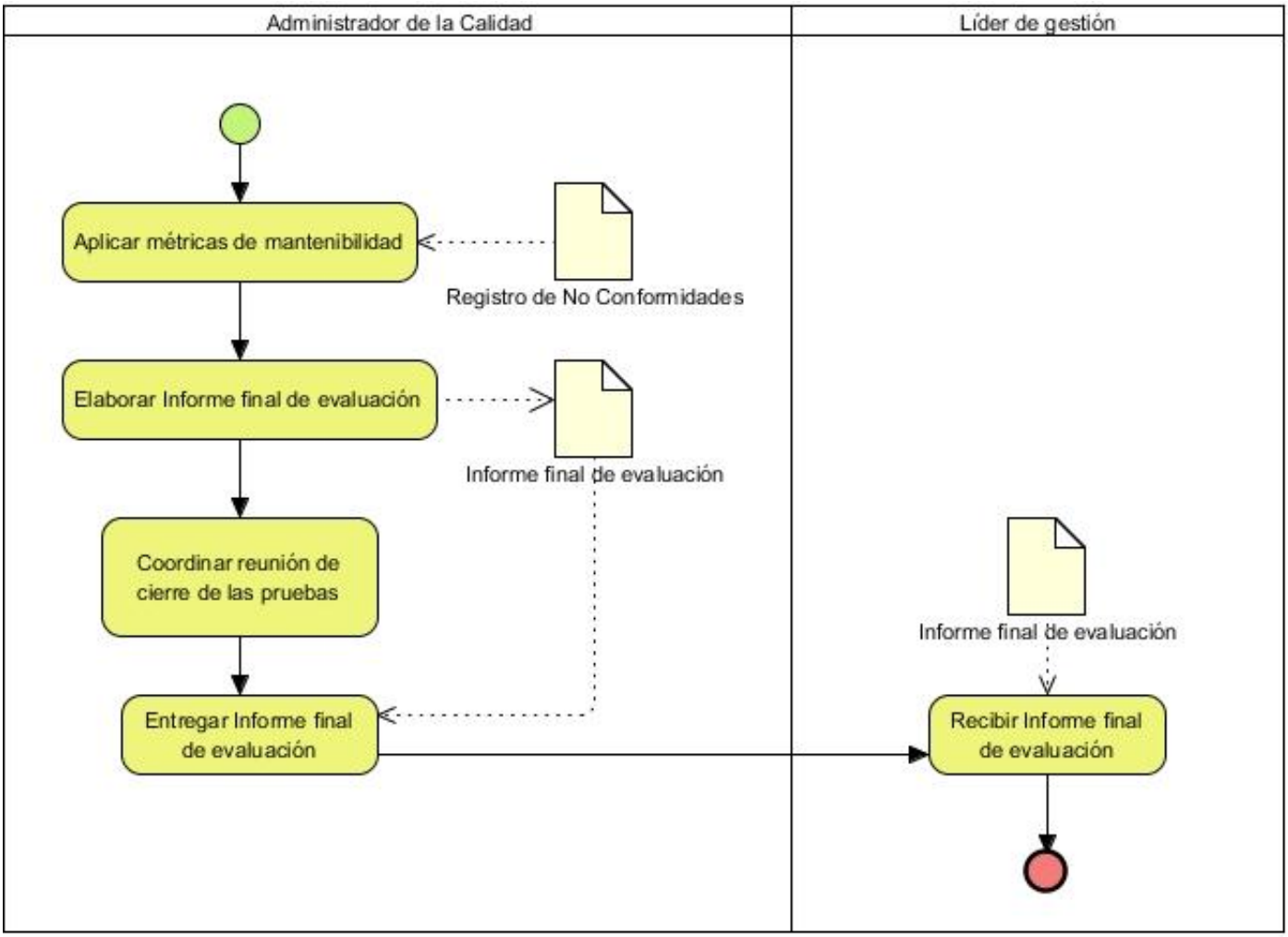


Fig. 6 Evaluación de las pruebas de Caja Blanca Estática en GINA.

2.4 ¿Cómo aplicar la métrica Registrabilidad de cambios?

Para aplicar la métrica Registrabilidad de cambios en función de evaluar la cambiabilidad y la estabilidad del software se aplica la siguiente fórmula: X = A/B, que consiste en dividir el número de cambios a funciones o módulos que tienen comentarios confirmados (A) entre el total de funciones o módulos modificados (B).

Los datos de las variables A y B se obtienen de los resultados que arrojen las pruebas de Caja Blanca Estática, que hayan sido registrados en el Registro de No Conformidades. Al obtener dichos datos y realizar la división correspondiente, el resultado será ubicado en un rango entre 0 y 1, donde su interpretación estará dada de la siguiente manera:

Mientras más cercano a 1 sea el valor, los cambios serán más registrables.

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

2.5 ¿Cómo aplicar la métrica Pruebas sin esfuerzo?

Para aplicar la métrica Pruebas sin esfuerzo en función de evaluar la facilidad del prueba del software se aplica la siguiente fórmula: Σ (T) / N, que consiste en sumar el tiempo (en seg) empleado en probar con el fin de asegurar si el informe de fallo ha sido resuelto o no en cada clase (T) y dividirlo entre el número de fallos resueltos en total (N).

Los datos T y N se obtienen de los resultados que arrojen las pruebas de Caja Blanca Estática. T y N se obtienen del Registro de No Conformidades. Al obtener dichos datos se realizan los cálculos correspondientes y el resultado obtenido, para ser interpretado, es ubicado en un rango de o a 300 seg/fallo, en caso de que este sea menor o igual que 300 seg/fallo, la conducta del mantenedor, el usuario o el sistema de software es Correcta, lo que significa que el software tiene una alta facilidad de prueba. En caso de que el resultado sea mayor que 300 seg/fallo, la conducta del mantenedor, el usuario o el sistema de software es Incorrecta, por lo que indicará que el software tiene una baja facilidad de prueba.

2.6 Uso de JSHint

La herramienta JSHint, que se utilizará para probar el código de forma automática, posee una gran facilidad de uso. Esta se puede utilizar tanto en línea (online), como instalada en un ordenador. En este

trabajo se decidió optar por su instalación. Para utilizarla de esta forma, se puede escoger como Sistema Operativo, tanto Windows como Linux, debido a que esta herramienta es multiplataforma.

Para trabajar con esta herramienta instalada en Windows, se instala el plugin JSHintNPP.dll, que se le configura al Notepad++, luego se abre el archivo JavaScript en Notepad++, se despliega el menú Plugins de la barra de menú y se selecciona la opción JSHint.

Todos los errores (si existen) que se encuentren en el archivo se mostrarán en un listado dentro de una ventana acoplable en la parte inferior de la ventana principal de Notepad++.

En el caso que se decida utilizar la herramienta instalada en Linux, tal y como se hizo en este trabajo, por las razones de ser un Sistema Operativo libre y de que en Cuba, incluyendo la UCI, se está migrando actualmente hacia esta plataforma.

Para la instalación de JSHint, debido a que esta herramienta es un plugin que se le configura al Gedit, se necesita instalar también node.js.

Para trabajar con JSHint, se abre el Gedit. Posteriormente, se debe acceder al menú Herramientas, que aparece en la barra de menú, luego se despliega la opción Clientside y por último se selecciona la opción JSHint. Al realizar los pasos anteriores, se mostrará una interfaz, donde dentro del área superior que contendrá esta, se coloca el código a analizar y luego se repiten los pasos anteriormente mencionados e instantáneamente la herramienta lo analiza. Si el código contiene algún tipo de error, este se muestra en la parte inferior de la interfaz, indicando la línea en que se encuentra, el número del caracter y un mensaje sobre el tipo de error encontrado. Véase el siguiente ejemplo:

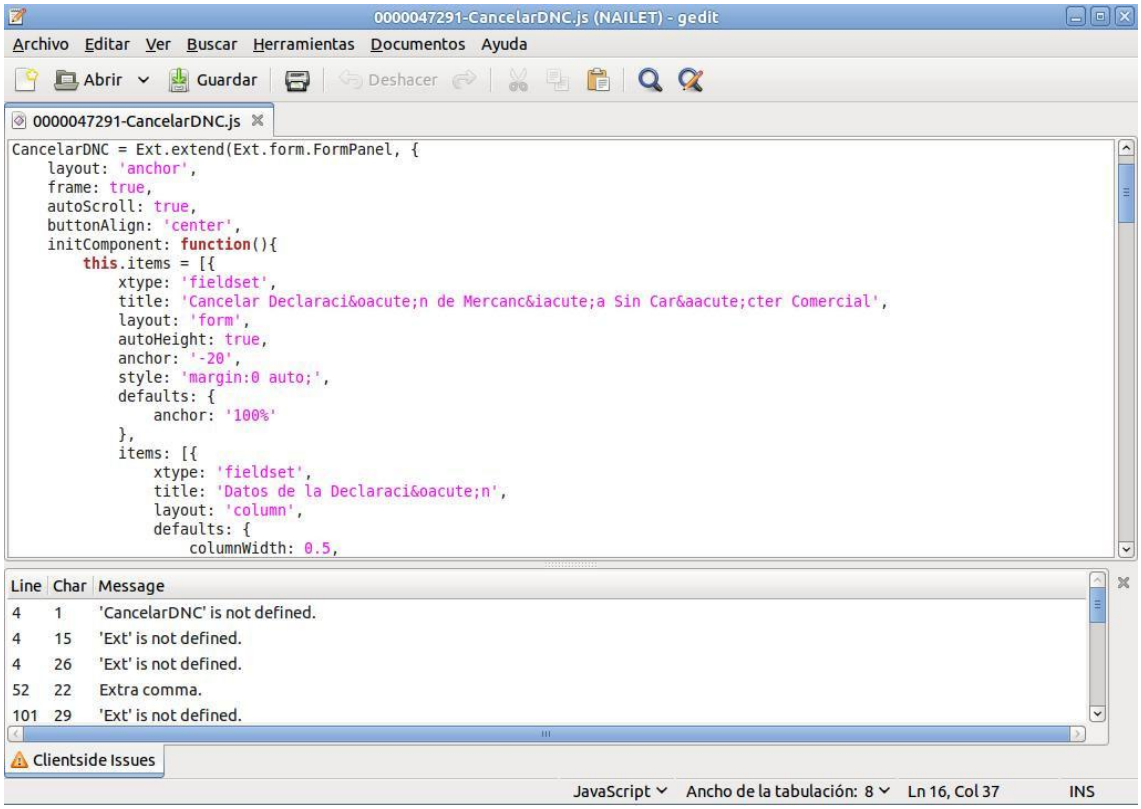


Fig. 7 Código probado en JSHint.

2.7 Conclusiones del Capítulo

En este capítulo se realizó la propuesta de un procedimiento para aplicar pruebas de Caja Blanca Estática en proyectos desarrollados sobre las tecnologías Symfony y ExtJS. Este se compuso de algunas precondiciones importantes a tener en cuenta por los desarrolladores durante la implementación de código y por un proceso de pruebas de Caja Blanca Estática, elaborado teniendo en cuenta las tecnologías Symfony y ExtJS, mediante el cual se pretende detectar la mayor cantidad de No Conformidades posibles al ser aplicado, permitiendo además una evaluación parcial de la mantenibilidad. La descripción realizada para cada subproceso del proceso de pruebas definido, siguió una serie de actividades sucedidas en orden lógico para cada rol, con sus respectivas entradas y salidas, lo que permite que el proceso de pruebas esté bien documentado. Las métricas seleccionadas permitirán evaluar la mantenibilidad para darle mayor completitud al proceso de pruebas definido y la especificación de uso de la herramienta JSHint facilitará el trabajo con la herramienta a la hora de realizar las pruebas.

CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

3.1 Introducción al capítulo

La validación de la solución propuesta en el capítulo anterior se realizará mediante el criterio de expertos, aplicando el método Delphi, y la aplicación del procedimiento en el subsistema Despacho No Comercial del Sistema GINA.

Para la aplicación del método Delphi, se utilizará como fuente de información un grupo de personas que deben presentar un conocimiento elevado del tema a tratar. Este grupo estará conformado por especialistas que poseen experiencias en temas relacionados con el presente trabajo. Se aplicarán cuestionarios a los expertos, a fin de poner de manifiesto convergencias de opiniones y deducir consensos eventuales.

Para la aplicación del procedimiento propuesto, se escogió el subsistema Despacho No Comercial perteneciente a GINA, el cual está implementado en Symfony y ExtJS. Para lograr la correcta realización de las pruebas se realizarán todas las actividades definidas en la propuesta, participarán todos los roles previstos y se generarán los artefactos pertinentes.

3.2 Método Delphi

El método Delphi es un método de estructuración de un proceso de comunicación grupal que es efectivo a la hora de permitir a un grupo de individuos, como un todo, tratar un problema complejo. Este método pretende extraer y maximizar las ventajas que presentan los métodos basados en grupos de expertos y minimizar sus inconvenientes. Para llevar a cabo esta técnica se selecciona un grupo de expertos que deben dar respuesta a un conjunto de preguntas que forman parte del cuestionario a realizar. La calidad de los resultados depende, sobre todo, del cuidado que se ponga en la elaboración del cuestionario y en la elección de los expertos a consultar. [35]

Para aplicar el método se siguen tres etapas fundamentales:

- Elección de expertos.

- Elaboración y lanzamiento del cuestionario.

- Desarrollo práctico y explotación de resultados.

3.2.1 Características del método Delphi

El método Delphi posee tres características fundamentales, ellas son:

- Anonimato: No debe existir contacto entre los participantes, pero el encargado de la encuesta sí puede identificar a cada participante y sus respuestas.

- Iteración y realimentación controlada: La iteración se consigue al presentar varias veces el mismo cuestionario. Como se van presentando los resultados obtenidos con los cuestionarios anteriores, se consigue que los expertos vayan conociendo los distintos puntos de vista y puedan ir modificando su opinión si los argumentos presentados les parecen más apropiados que los suyos.

- Respuesta del grupo en forma estadística: La información que se presenta a los expertos no es sólo el punto de vista de la mayoría, sino que se presentan todas las opiniones indicando el grado de acuerdo que se ha obtenido.

3.3 Proceso de selección de expertos

Uno de los aspectos fundamentales a considerar metodológicamente para la aplicación práctica de Delphi, es la selección del grupo de expertos a encuestar.

No hay forma de determinar el número óptimo de expertos para participar en una encuesta Delphi, pero aun así la selección de estos, así como la determinación de la cantidad para la aplicación del método Delphi, constituye una tarea fundamental.

Estudios realizados por investigadores de la Rand Corporation [36] señalan que es necesario un mínimo de siete expertos, teniendo en cuenta que el error disminuye notablemente por cada experto añadido hasta llegar a los siete, pero no es aconsejable recurrir a más de treinta expertos, pues la mejora en la suposición es muy pequeña y normalmente el incremento en coste y trabajo de investigación no compensa dicha mejora.

Para llevar a cabo este método y evaluar la calidad de la propuesta, se realizó la selección de un grupo expertos, integrantes de diferentes centros, en su mayoría del CEIGE que estuvieran dispuestos a participar en la encuesta y que ejercieran roles relacionados con el procedimiento propuesto. Además, que tuvieran experiencia como Desarrolladores o Administradores de la Calidad y que contaran con más de dos años de experiencia profesional. Elegir los expertos atendiendo a las características mencionadas

propicia obtener resultados con calidad, junto a otras cualidades propias de éstos como pueden ser: la seriedad, la honestidad, la responsabilidad y otras en este sentido, que hacen que las opiniones brindadas sean confiables y válidas para el objetivo propuesto.

Luego de hacer esta previa selección se aplicó una metodología usada en este método para determinar la competencia de los expertos. Para realizar las evaluaciones se realizó una encuesta (Ver Anexo 9) la cual fue resuelta por los expertos seleccionados.

La metodología establece que lo primero que el experto debe hace es marcar con una x, en una escala creciente de 0 a 10, el valor que se corresponde con el grado de conocimiento o información que tienen sobre el tema de investigación, como se muestra en el Anexo 9.

Con estos datos se calcula el Kc (coeficiente de conocimiento) para cada uno de los expertos. Donde:

Kc = El valor que marcó el experto en la tabla multiplicado por 0.1.

Posteriormente, se les pide a los expertos su autoevaluación de los niveles de argumentación o fundamentación sobre el tema de investigación, para esto deben marcar con una x en la tabla de coeficiente de argumentación que aparece en el Anexo 9.

A partir de la autoevaluación realizada y el patrón de valores para el coeficiente de argumentación (Ver Anexo 11), se calcula el coeficiente de argumentación Ka. Donde:

Ka = Σ de los valores que se obtienen de sustituir las cruces de las casillas marcadas, por los valores correspondientes en su posición de la Tabla de valores (Anexo 10) para el coeficiente de argumentación.

Luego de ser calculados los valores de Kc y Ka, se calcula el coeficiente de competencia K mediante la siguiente fórmula:

$$K = \frac{Kc + Ka}{2}$$

El resultado de K será interpretado de la siguiente forma:

Si 0,8 < K < 1: Coeficiente de competencia Alto.

Si 0,5 < K < 0,8: Coeficiente de competencia Medio.

Si K < 0,5: Coeficiente de competencia Bajo.

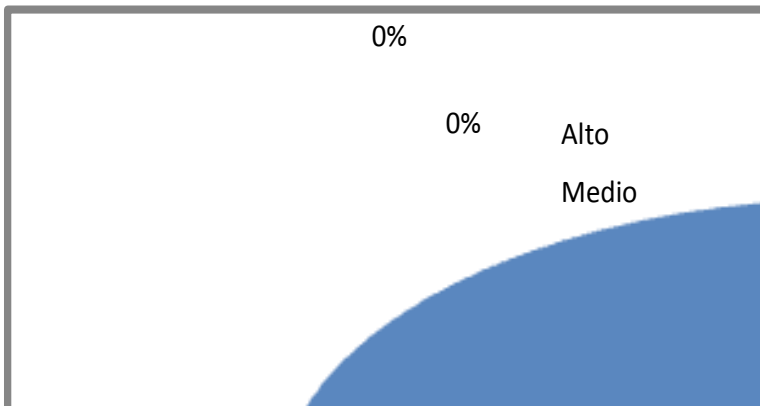
Para obtener mejores resultados en la aplicación del Delphi es conveniente utilizar los expertos cuyo coeficiente de competencia sea Alto o Medio.

Luego de realizada la encuesta de autoevaluación a ocho expertos, todos fueron seleccionados para formar parte del grupo de validación de la propuesta de solución, ya que todos arrojaron como resultado un coeficiente de competencia Alto o Medio. Los resultados del análisis del coeficiente de competencia se muestran a continuación:

Tabla 1: Coeficientes de competencia de los expertos

Experto	Kc	Ka	K	Coeficiente de competencia
E1	0.7	0.8	0.75	Medio
E2	0.8	0.99	0.89	Alto
E3	0.7	0.77	0.74	Medio
E4	0.4	0.70	0.55	Medio
E5	0.6	0.71	0.66	Medio
E6	0.6	0.795	0.69	Medio
E7	0.6	0.795	0.69	Medio
E8	0.6	0.8	0.7	Medio

En la gráfica que se muestra a continuación se representa el resultado de acuerdo al coeficiente de competencia del grupo resultante para la validación de la propuesta, donde los expertos con coeficiente Alto representan un 12.5% del total y un 87.5% los que poseen coeficiente Medio.



Gráfica 1: Porcentaje del coeficiente de competencia de los expertos.

3.4 Elaboración del cuestionario para la validación de la propuesta

Para validar la solución propuesta se utilizó un cuestionario (Ver Anexo 11) conformado por 10 preguntas orientadas a evaluar aspectos críticos de la solución en cuanto a la calidad de la propuesta, atendiendo a unidades de medida como la correctitud, la claridad de las actividades, la completitud y la capacidad que tiene la propuesta para ser aplicada. El cuestionario fue realizado por los expertos seleccionados, pidiéndoseles que evaluaran en las categorías ordinales de Muy Adecuado (MA), Bastante Adecuado (BA), Adecuado (A), Poco Adecuado (PA), y No Adecuado (NA). Con las respuestas dadas por los expertos se podrá obtener la concordancia entre ellos y la aceptación y validez de la propuesta.

3.5 Establecimiento de la concordancia entre los expertos

Cuando se tienen datos de tipo ordinal, se aplica el coeficiente de Kendall, el cual toma en consideración el orden. Este es un instrumento estadístico muy apropiado para indicar el grado de asociación de las evaluaciones ordinales hechas por evaluadores múltiples cuando se evalúa la misma muestra. Los valores del coeficiente deben oscilar entre 0 y 1. Entre mayor sea el valor, más fuerte será la asociación. [37]

Para que la propuesta tenga mayor validez debe existir un excelente acuerdo entre los expertos. El coeficiente de Kendall (W) posibilita comprobar el grado de coincidencia de las valoraciones realizadas por los expertos. Este se obtiene aplicando la siguiente fórmula:

$$W = \frac{12 * S}{K^2(N^3 - N)}$$

Donde la suma de los cuadrados de las desviaciones de la media (S) se obtiene de la sumatoria de los rangos (Sj) entre N, siendo N el total de aspectos a evaluar (los aspectos son las preguntas del cuestionario) y K el número total de expertos. A continuación se muestran los cálculos realizados para determinar la concordancia de los expertos:

Tabla 2: Concordancia entre los expertos

	E1	E2	E3	E4	E5	E6	E7	E8	Sj
P1	4	3	3	3	4	3	3	5	28
P2	4	5	4	3	4	3	3	3	29
P3	4	4	4	3	4	3	3	3	28
P4	4	4	4	4	3	4	3	5	31
P5	4	5	4	4	4	4	4	3	32
P6	4	4	3	3	4	4	4	3	29
P7	3	4	4	3	3	3	3	3	26
P8	3	3	3	3	3	4	4	3	26
P9	4	4	5	3	3	3	3	3	28
P10	4	5	4	3	3	3	4	3	29

K es el número de expertos que intervienen en el proceso de validación, por lo tanto, **K=8**.

N, es la cantidad de aspectos a validar, en este caso **N=10**.

Sj, es la suma de los rangos asignados a cada pregunta por parte de los expertos, en este caso **Sj = 286**.

$\bar{S}j$, es la media de los rangos y se determina a través de la fórmula:

$$\bar{S}j = \frac{\sum_{j=i}^{N} Sj}{N}$$

Obteniendo, en este caso, un valor de:

$$\bar{S}j = \frac{286}{10} = 28.6$$

Posteriormente, se determina la desviación media, aplicando la siguiente ecuación:

$$S = \sum_{J=1}^{N} (Sj - \bar{S})^2$$

El valor obtenido al aplicar la ecuación anterior fue **S = 32.40**.

Sustituyendo los valores en la fórmula general de Kendall:

$$W = \frac{12 * S}{K^2(N^3 - N)} = \frac{12 * 32.40}{64 - (1000 - 10)} = \frac{388.8}{63360} = 0.0061363$$

El coeficiente de W brinda el valor que permite decidir el nivel de concordancia entre los expertos. Este valor (W) siempre es positivo, y oscila entre 0 y 1. El coeficiente de Kendall obtenido permite calcular el Chi cuadrado real, el cual tiene el objetivo de medir si existe o no concordancia entre los expertos y se obtiene a través de la fórmula:

$$x^2 = K(N - 1)W$$

$$x^2 = 8 * (10 - 1) * 0.0061363$$

$$x^2 = 0.44$$

Para buscar el Chi cuadrado tabulado en la tabla de distribución, mostrada en el (Ver Anexo 12) se calcula el grado de libertad, siendo **N - 1 = 10 - 1 = 9**.

α es el nivel de significación utilizado para calcular el nivel de confianza. El nivel de confianza es igual a 100(1 - alfa) %, es decir, un alfa de 0,05 indica un nivel de confianza de 95%.

El Chi - Cuadrado real se compara con el de las tablas estadísticas. Si el Chi- Cuadrado real es menor que el Chi – Cuadrado de la tabla entonces hay concordancia:

$$X^2\ real < X^2(\alpha, N-1)$$

$$0.44 < (0.05, 9)$$

$$0.44 < 16.91$$

3.6 Desarrollo práctico y explotación de los resultados

Para la valoración de consensos para los expertos, existen distintas técnicas y modelos, pero es más conveniente aplicar el modelo matemático Torgerson, el cual es muy útil cuando las escalas empleadas en los instrumentos aplicados a los expertos son ordinales. [38] Además, una de las desventajas del Delphi radica en la subjetividad de los criterios emitidos, por lo que se opta, para tratar de resolver este problema, por emplear este modelo matemático que permite asignar un valor de escala a cada indicador y determinar límites entre cada categoría, y así, obtener los límites reales entre las categorías ordinales y sus correspondientes a escala de intervalo, y entre cada uno de los rangos que componen los criterios evaluativos dados por los expertos, para poder conocer hasta qué valores reales se puede considerar que la variable es, Muy Adecuado, Bastante Adecuado, Adecuado, Poco Adecuado o No Adecuado.

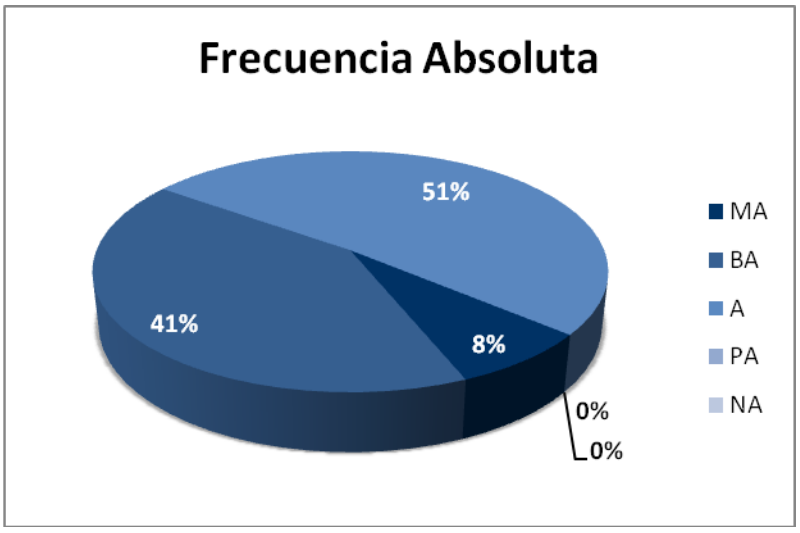
A partir de las respuestas dadas por cada experto se analizarán los resultados, los cuales se recogen en la siguiente tabla:

Tabla 3: Frecuencia Absoluta

	Frecuencia absoluta					
	MA	**BA**	**A**	**PA**	**NA**	**Total**
P1	1	2	5	0	0	8
P2	1	3	4	0	0	8
P3	0	4	4	0	0	8
P4	1	5	2	0	0	8
P5	1	6	1	0	0	8

P6	0	5	3	0	0	8
P7	0	2	6	0	0	8
P8	0	2	6	0	0	8
P9	1	2	5	0	0	8
P10	1	3	4	0	0	8
Total	6	32	40	0	0	

Los resultados recogidos en la tabla anterior quedan representados en la siguiente gráfica mostrando el grado de aceptación de cada uno de los aspectos y preguntas evaluadas por los expertos.



Gráfica 2: Frecuencia absoluta.

Para obtener resultados satisfactorios, una vez calculados los datos, se realizan los siguientes pasos:

1- Se construye una tabla de frecuencia absoluta acumulada donde cada número en la fila se obtiene sumándole el anterior, excepto el primero que se mantiene igual.

Tabla 4: Frecuencia absoluta acumulada

	Frecuencia absoluta acumulada				
	MA	**BA**	**A**	**PA**	**NA**
P1	1	3	8	8	8
P2	1	4	8	8	8
P3	0	4	8	8	8
P4	1	6	8	8	8
P5	1	7	8	8	8
P6	0	5	8	8	8
P7	0	2	8	8	8
P8	0	2	8	8	8
P9	1	3	8	8	8
P10	1	4	8	8	8

2- Se copia la tabla anterior y se borran los dígitos. En esta nueva tabla se construye la tabla de frecuencia relativa acumulada, la cual se logra dividiendo por 8 (número total de expertos), cada uno de los números de la tabla anterior. Dado que con cuatro puntos se obtienen tres intervalos y en la tabla hay cinco categorías, se eliminan dos columnas, pues no son necesarias.

Tabla 5: Frecuencia relativa acumulada

	Frecuencia relativa acumulada				
	MA	**BA**	**A**	**PA**	**NA**
P1	0.13	0.38	1	1	1
P2	0.13	0.5	1	1	1
P3	0	0.5	1	1	1
P4	0.13	0.75	1	1	1

P5	0.13	0.88	1	1	1
P6	0	0.63	1	1	1
P7	0	0.25	1	1	1
P8	0	0.25	1	1	1
P9	0.13	0.38	1	1	1
P10	0.13	0.5	1	1	1

3- Se buscan las imágenes de los elementos de la tabla anterior, y la nueva tabla que se obtiene cuenta con cuatro elementos nuevos.

- **Suma:** Es la suma de las columnas y de las filas.
- **P:** Promedio de las filas.
- **N:** Se obtiene al dividir la suma de las sumas de todas la filas entre el número que se ha obtenido de multiplicar el total de categorías por el de preguntas.
- **N-P:** Valor promedio que otorgan los expertos consultados a cada pregunta propuesta.

Tabla 6: Puntos de corte

						N = 0.33	
	MA	**BA**	**A**	**Suma**	**P**	**N-P**	**Nivel de adecuación**
P1	0.13	0.38	1	1.50	0.4	-0.07	Muy adecuado
P2	0.13	0.50	1	1.63	0.4	-0.07	Muy adecuado
P3	0	0.50	1	1.50	0.4	-0.07	Muy adecuado
P4	0.13	0.75	1	1.88	0.5	-0.17	Muy adecuado
P5	0.13	0.88	1	2.01	0.5	-0.17	Muy adecuado
P6	0	0.63	1	1.63	0.4	-0.07	Muy adecuado
P7	0	0.25	1	1.25	0.3	0.03	Muy Adecuado
P8	0	0.25	1	1.25	0.3	0.03	Muy adecuado
P9	0.13	0.38	1	1.51	0.4	-0.07	Muy adecuado

P10	0.13	0.50	1	1.63	0.4	-0.07	Muy adecuado
Suma	0.77	5.01	10	**15.78**			
Puntos de corte	0.06	0.42	0.83				

La suma obtenida de las 3 primeras columnas da los puntos de corte. Estos puntos de corte se utilizan para determinar el grado de adecuación o categoría de cada aspecto encuestado según los expertos.

Si el valor promedio de adecuación del elemento a evaluar es:

Menor o igual que 0.06 el nivel de adecuación es Muy adecuado.

Mayor que 0.06 y menor o igual que 0.42 el nivel de adecuación es Bastante adecuado.

Mayor que 0.42 y menor o igual que 0.83 el nivel de adecuación es Adecuado.

Mayor que 0.83 el nivel de adecuación es Poco adecuado o No adecuado.

Una vez aplicado el método Delphi, las encuestas realizadas a los 8 expertos arrojaron resultados satisfactorios:

Todas las preguntas fueron valoradas por los expertos de Muy adecuada, por lo que se deduce que el 100% de los expertos coinciden en la utilidad que tiene la puesta en práctica de la propuesta en el Sistema GINA.

3.7 Aplicación de la propuesta de solución el subsistema Despacho No Comercial de GINA

Partiendo de los resultados arrojados por el método de Expertos y considerando que estos fueron positivos, se decidió fortalecer la validación de la solución propuesta mediante su aplicación en el subsistema Despacho No Comercial (DNC) perteneciente al Sistema GINA, el cual se encuentra implementado bajo las tecnologías Symfony 1.2.8 y ExtJS 3.0. Este subsistema se encuentra actualmente en la fase de implementación, por lo que resulta idóneo para la aplicación de pruebas de Caja Blanca Estática. Del mismo se evaluaron un total de seis clases, siendo cuatro de ellas clases visuales (CancelarDNC, PresentarDNC, RegistrarDS y Prorroga), implementadas en ExtJS 3.0 y el resto constituido por la clase controladora (Actions) y una clase auxiliar a esta (sfAuxiliarDNCTC), ambas implementadas en Symfony 1.2.8.

3.7.1 Descripción del subsistema DNC

EL subsistema DNC debe ser asimilado por todas las aduanas del país donde se realice el control de las importaciones sin carácter comercial para personas jurídicas y naturales tanto por la vía marítima, como por la aérea, y para todos los tipos de operaciones que realicen.

Este subsistema abarcará el procesamiento automatizado desde el pre-despacho, si procede, hasta la fase de liquidación y caja, previendo su funcionamiento en línea, es decir, en el momento que se producen las operaciones.

El subsistema debe ser capaz de:

- Automatizar el despacho de las importaciones logrando un mayor control sobre los términos establecidos de artículos controlados y su valoración.
- Brindar un servicio aduanero con calidad, oportuno y confiable.
- Garantizar que la información global del proceso llegue a todas las unidades por las diferentes vías de conectividad.
- Lograr una recaudación y estadísticas confiables.
- Disminuir las posibilidades de cualquier actividad de fraude.

3.8 Planificación de las pruebas de Caja Blanca Estática en el subsistema DNC

Para la aplicación del procedimiento propuesto para las pruebas de Caja Blanca Estática en proyectos desarrollados en Symfony y ExtJS, se realizaron todas las actividades contempladas en cada subproceso del proceso de pruebas definido en el mismo. Por tanto, se comenzó por el subproceso Planificación de las pruebas. Como primera actividad de este subproceso, se realizó la reunión de inicio de las pruebas, donde se contó con la presencia del Líder de Gestión, el Administrador de la Calidad, el Desarrollador de IU y de PHP y el Probador. En esta reunión se aprobó la Solicitud de pruebas enviada por el Líder de Gestión al Administrador de la Calidad. Posteriormente se comenzaron a definir puntos importantes que deben ser recogidos en el Plan de pruebas. Los mismos aparecen a continuación:

- **Plan de Pruebas**

Introducción

En el presente Plan de pruebas se planifican las Pruebas de Caja Blanca Estática para el subsistema DNC, desarrollado en Symfony y ExtJS.

Alcance

Este Plan comprende la planificación de las pruebas de Caja Blanca Estática en el subsistema DNC, desarrollado bajo las tecnologías Symfony y ExtJS.

Definiciones, acrónimos y abreviaturas

DNC: Despacho No Comercial

GINA: Gestión Integral de Aduana

Objetivos

- Comprobar la estructura y sintaxis del código en las clases que se encuentran implementadas hasta el momento en el subsistema DNC.
- Verificar que la codificación cumpla con los estándares establecidos en el Sistema GINA.

Estrategia de prueba

Este plan contempla pruebas de Caja Blanca Estática, donde específicamente se verificó el cumplimiento estricto de los estándares de codificación de forma manual y la correcta implementación del código de forma automatizada.

Técnica

Para realizar esta investigación se estudiaron diferentes técnicas de pruebas de Caja Blanca. De ellas, se decidió aplicar la de Estructura de Datos Locales, la cual se centra en el estudio de las variables del programa y será llevada a cabo mediante la aplicación de las Listas de chequeo.

Herramientas

Se realizaron pruebas manuales y automáticas, para las pruebas manuales se utilizarán Listas de chequeo y para las pruebas automatizadas se utilizará la herramienta JSHint.

3.8.1 Diseño de las pruebas de Caja Blanca Estática en el subsistema DNC

En el diseño de las pruebas, se desarrollaron las actividades previstas en este subproceso para concluir la elaboración del Plan de pruebas. Se configuró el entorno, teniendo en cuenta los recursos del sistema que deben estar presentes, tanto en software como en hardware y los recursos humanos que deben participar en el proceso de pruebas de Caja Blanca Estática en GINA. Además, se elaboró el Cronograma de pruebas, para las actividades que conforman la estrategia de prueba.

- **Configuración del entorno de pruebas**

La configuración del entorno de pruebas queda determinada mediante la siguiente distribución de los recursos del sistema y los recursos humanos que serán utilizados para el desarrollo de las pruebas de Caja Blanca Estática:

Tabla 7: Recursos del sistema

Hardware	Software
1 PC Cliente Pentium 4 o superior, con 1 GB de RAM o superior.	- Sistema Operativo Windows (XP o superior) o Linux (Ubuntu 10.04 o superior). - Instalación de Microsoft Office (preferentemente 2007) u Open Office (preferentemente 3.2) y AcrobatPDF (3.03 o superior).

Tabla 8: Recursos humanos

Rol	Cantidad
Planificador/Líder de Gestión	1
Administrador de la Calidad	1
Desarrollador de IU	1
Desarrollador de PHP	1
Probador	1

- **Cronograma de pruebas**

Tabla 9: Cronograma de pruebas

No.	Tarea	Fecha	Responsable	Participantes	Observaciones
1	Evaluación del código mediante Listas de chequeo.	22/04/2012	Probador	Equipo de Calidad	
2	Evaluación del código mediante la herramienta JSHint.	28/04/2012	Probador	Equipo de Calidad	
3	Aplicación de Criterios de criticidad.	28/04/2012	Administrador de la Calidad	Equipo de Calidad	
4	Aplicación de pruebas de regresión.	04/05/2012	Probador	Equipo de Calidad	
5	Elaboración de Listado del No Conformidades.	10/05/2012	Probador	Equipo de Calidad	
6	Elaboración del Registro de incidencias.	10/05/2012	Administrador de la Calidad.	Equipo de Calidad	
7	Aplicación de métricas de mantenibilidad.	12/05/2012	Administrador de la Calidad	Equipo de Calidad	
8	Elaboración del Informe final de evaluación.	14/05/2012	Administrador de la Calidad	Equipo de Calidad	

3.9 Ejecución de las pruebas de Caja Blanca Estática en el subsistema DNC

En la ejecución de las pruebas, se tenía concebido realizar un máximo de tres iteraciones, tanto manuales, como automáticas. De este número iteraciones previstas, sólo hubo que realizar dos iteraciones de cada tipo, ya que en la primera iteración, se encontraron una serie de errores, que fueron corregidos posteriormente por el desarrollador del subsistema, de modo tal, que al realizar la segunda iteración, tanto manual como automática, el código alcanzó un resultado de 0 No Conformidades, quedando ejecutadas las pruebas de regresión satisfactoriamente.

- **Listas de chequeo**

Las Listas de chequeo fueron aplicadas como técnica manual para filtrar los errores introducidos en la generación del código, verificando así la estructura del código según los estándares de codificación definidos para Symfony y ExtJS en el Sistema GINA, encontrando en total un error en el código implementado en ExtJS y cuatro errores en el código implementado en Symfony durante la primera iteración, y obteniéndose como resultado de la segunda iteración, una calificación de Correcto en ambos códigos. Los resultados de la primera iteración manual, correspondientes a cada marco de trabajo, se muestran a continuación:

Tabla 10: Lista de chequeo para ExtJS - Primera iteración.

PROTOTIPO DE CLASES				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Las interfaces visuales están implementadas según el modelo de clases que propone ExtJS?	4		En la clase PresentarDNC aparece un paso que no está concebido en el modelo de clases.
CODIFICACIÓN JAVASCRIPT				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Se aplica el patrón *El nombre revela la intención*?	5		
	2. ¿Se hace uso indiscriminado de funciones públicas?	0		
VARIABLES				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Las variables locales, los atributos y métodos de una clase están denotadas con letras minúsculas?	5		
	2. ¿Las variables globales, las funciones y las	5		

	clases están denotadas con letras mayúsculas?			
	4. ¿Los nombres compuestos por más de una palabra se denotan sin separadores y usando la primera letra mayúscula a partir de la segunda palabra?	5		

Tabla 11: Lista de chequeo para Symfony - Primera iteración.

GENERAL				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**

	1. ¿Se utiliza la notación UpperCamelCase?	3		En el código de la clase Actions.class, aparecen:
				public function executeBuscardeclaranteporcodigo(){
				public function executeBuscarpersonaporci(){
				public function executeBuscarpersonaporDocumentoIdentificacion(){

	2. ¿Se utiliza la nomenclatura para las clases de las acciones y sus funciones definidas en el estándar para Symfony en GINA?	5		
	3. ¿Se utiliza la notación **sf** para los plugins?	5		
CLASES				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Los nombres de los archivos de las	5		

	clases están compuestos por el nombre de la clase seguido de un punto y la palabra “class” y la extensión del archivo “.php”?			
!	2. ¿Las clases siguen la estructura por niveles, como está definido en el estándar para Symfony en GINA?	5		
FUNCIONES				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Los nombres de las funciones reflejan claramente cuál es la acción que realizan las mismas?	5		
	2. ¿Se hace uso indiscriminado de funciones públicas?	0		.
VARIABLES				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Los nombres de las variables reflejan claramente cuál es el contenido de las mismas?	4		Hay variables con nombres incompletos, que no reflejan claramente cuál es su contenido.
ESTILOS DE CODIFICACIÓN				
Nivel	**Criterio de evaluación**	**Evaluación**	**N.P.**	**Observaciones**
	1. ¿Se siguen las políticas de llaves definidas en el estándar?	4		Se utilizan ambas políticas del estándar, cuando debe utilizarse solo una de ellas.
	2. ¿Se siguen las políticas de paréntesis definidas en el estándar?	5		
	3. ¿Se siguen las políticas de sangría, tabulación y espacios definidas en el	5		

	estándar?			

- **Criterios de criticidad**

Como apoyo a las pruebas automáticas que se realizaron posteriormente, fue necesario establecer Criterios de criticidad para determinar el estado de las pruebas en dependencia de los mismos. Los criterios establecidos para este proceso se encuentran en el Anexo 2.

- **Evaluación del código con JSHint**

El código de las clases visuales, implementadas en ExtJS, fue evaluado con la herramienta JSHint. Se evaluaron un total de cuatro clases visuales (PresentarDNC, CancelarDNC, RegistrarDS, y Prorroga), de ellas se obtuvieron los siguientes resultados:

- En la clase PresentarDNC, no se detectaron No Conformidades en el código.
- En la clase CancelarDNC se detectó la siguiente No Conformidad:

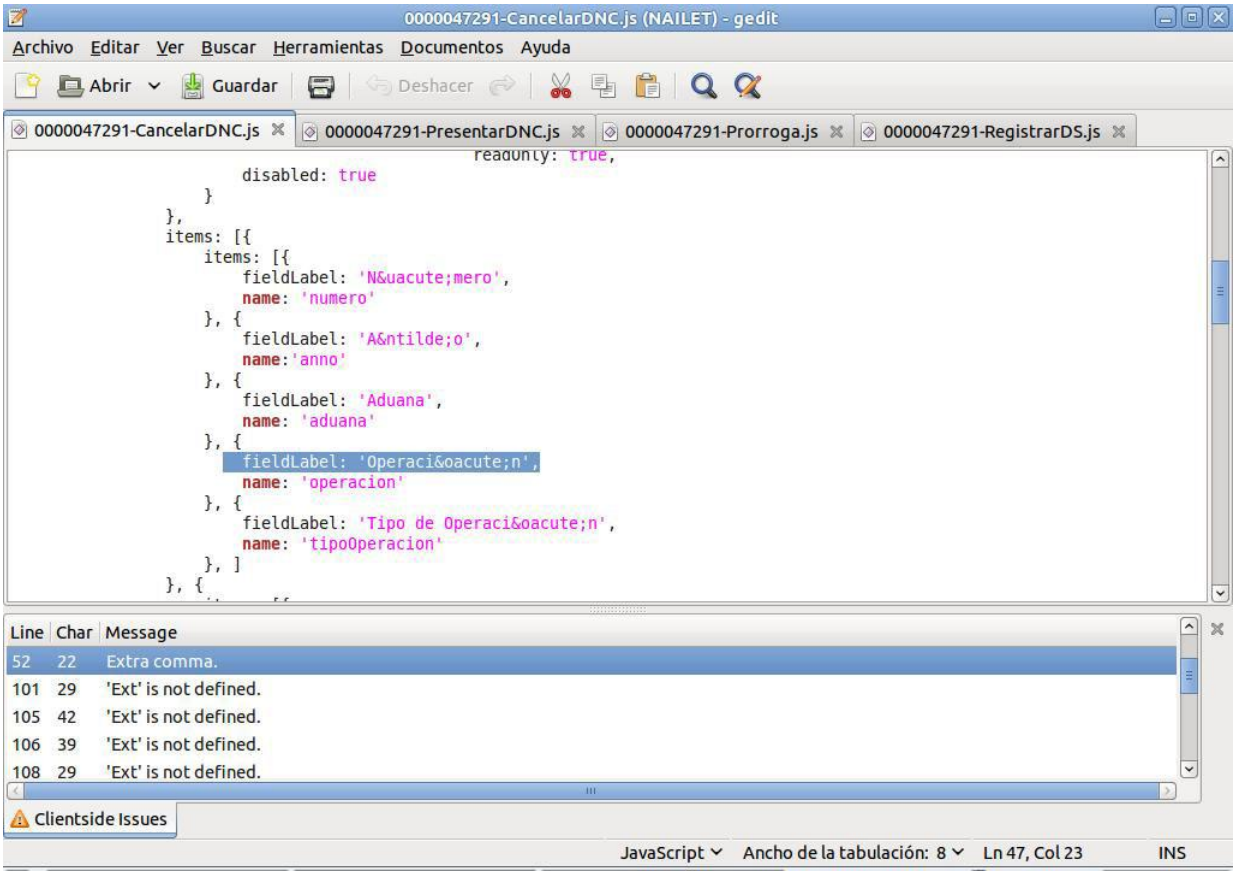


Fig. 8 Primera NC detectada por JSHint en la clase CancelarDNC.

- En la clase RegistrarDS se detectaron dos No Conformidades, a continuación se muestra una de ellas (la otra se muestra en el Anexo 13):

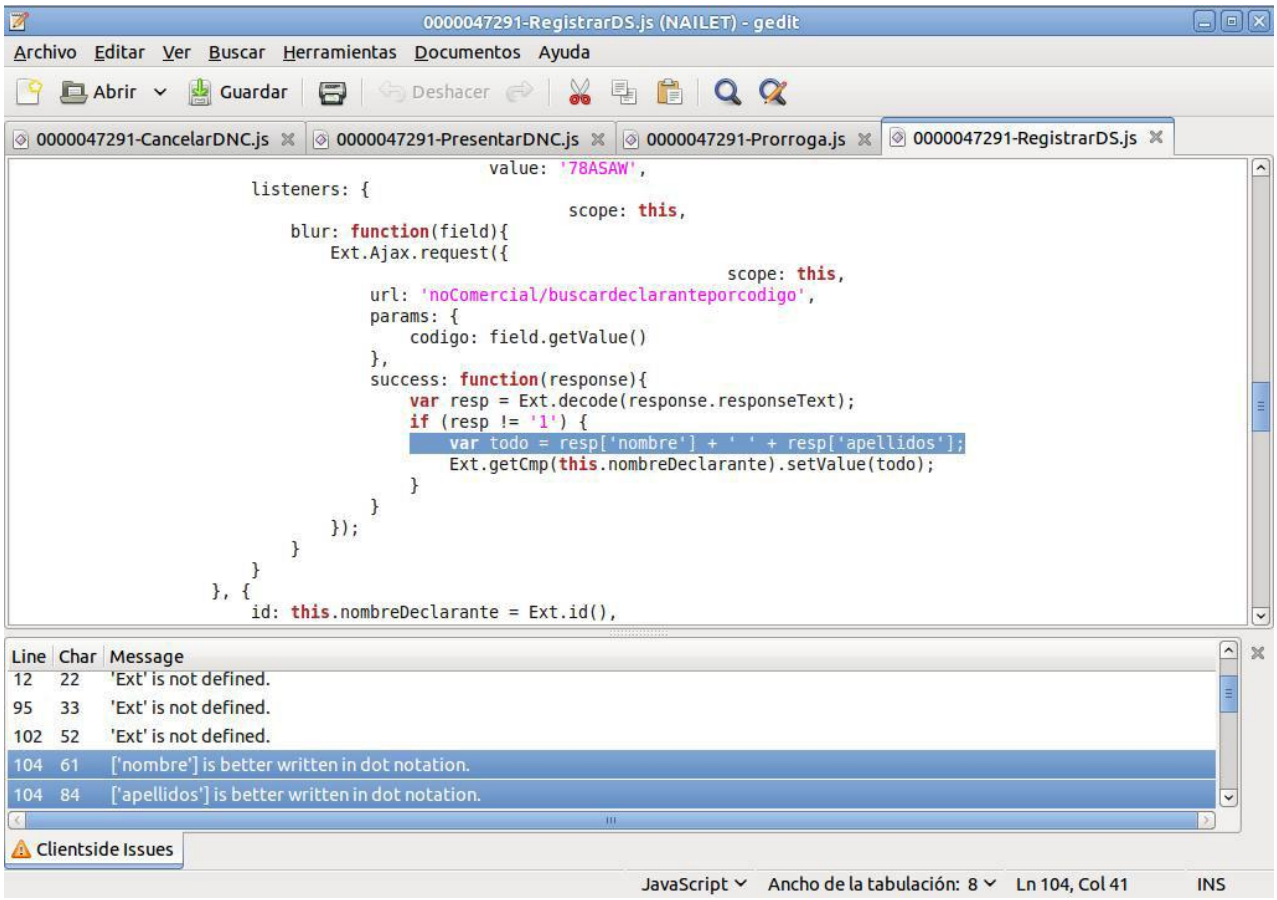


Fig. 9 Primera NC detectada por JSHint en la clase RegistrarDS.

- En la clase Prorroga se detectaron tres No Conformidades, a continuación se muestra una de ellas (las demás se muestran en el Anexo 14 y el Anexo 15):

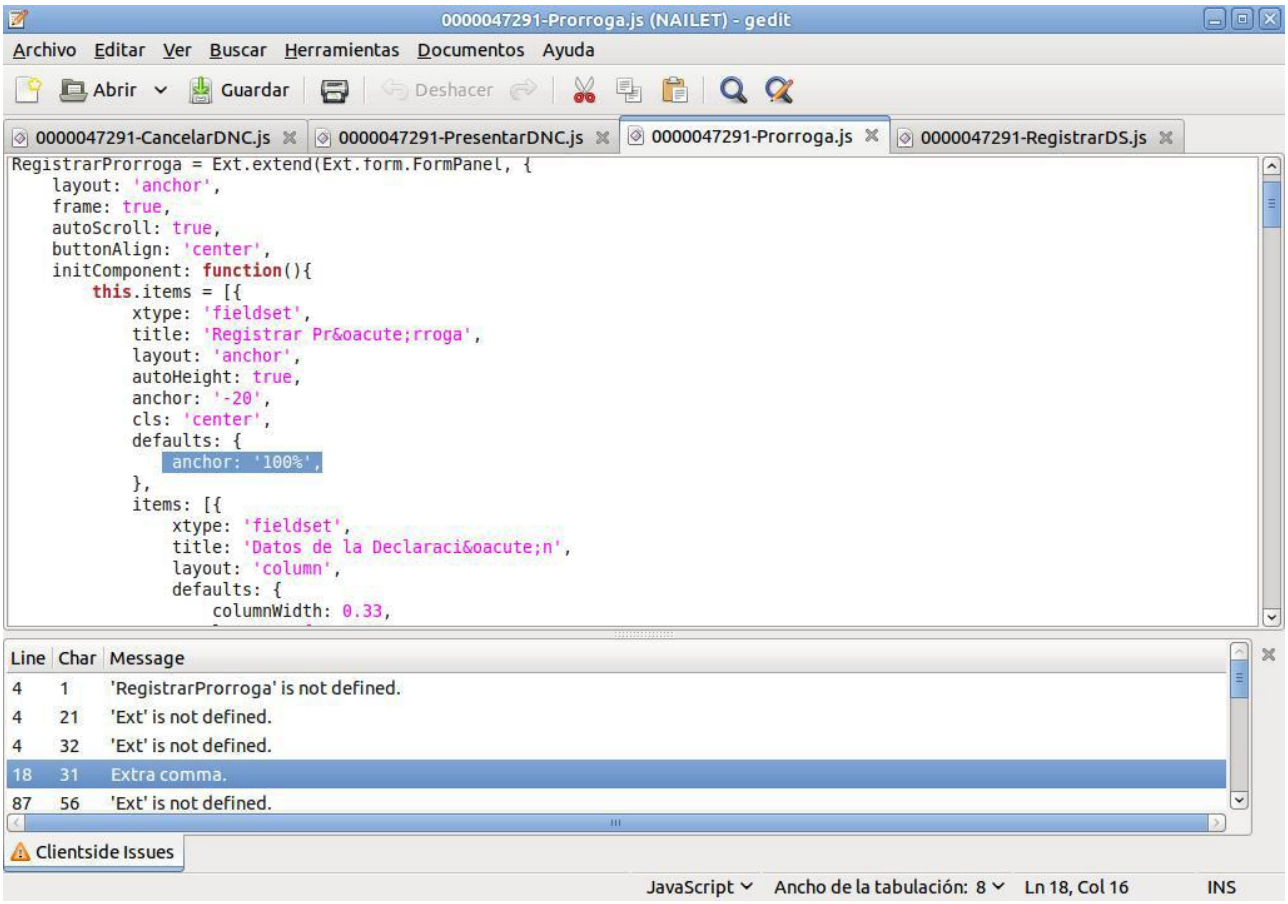


Fig. 10 Primera NC detectada por JSHint en la clase Prorroga.

- **Registro de No Conformidades**

Durante las pruebas de Caja Blanca Estática realizadas con JSHint en el subsistema DNC, en la primera iteración se registraron las siguientes No Conformidades, véase en la siguiente tabla:

Tabla 12: No Conformidades encontradas con JSHint – Primera iteración.

										Estado		
Fecha	**Probador**	**Elemento**	**No.**	**NC**	**Aspecto correspondiente**	**Tipo**	**S**	**NS**	**Etapa de detección**	**RA**	**PD**	**NP**
28/04/ 2012	Nailet Hernández	Código de la clase CancelarDNC.	1	Coma adicional.	Se encuentra en la línea 52.	Error de sintaxis	x		Prueba		x	
28/04/ 2012	Nailet Hernández	Código de la clase Prorroga.	2	Coma adicional.	Se encuentra en la línea 18.	Error de sintaxis	x		Prueba		x	
28/04/ 2012	Nailet Hernández	Código de la clase Prorroga.	3	Miembro 'minValue' duplicado.	Se encuentra en la línea 95.	Error estructural		x	Prueba		x	
28/04/ 2012	Nailet Hernández	Código de la clase Prorroga.	4	Falta un punto y coma.	Se encuentra en la línea 199.	Error de sintaxis	x		Prueba		x	
28/04/ 2012	Nailet Hernández	Código de la clase RegistrarDS.	5	['nombre'] está mejor escrito en notación de puntos.	Se encuentra en la línea 104.	Error de sintaxis		x	Prueba		x	
28/04/ 2012	Nailet Hernández	Código de la clase RegistrarDS.	6	['apellidos'] está mejor escrito en notación de puntos.	Se encuentra en la línea 104.	Error de sintaxis		x	Prueba		x	
28/04/	Nailet	Código de	7	Espacios	Se encuentra en	Error		x	Prueba		x	

2012	Hernández	la clase Registrar DS.		mixtos y tabulaciones.	la línea 112.	de estructural						

- **Registro de incidencias**

Durante la aplicación de las pruebas se registran las incidencias ocurridas en el proceso. (Ver Anexo 16)

3.10 Evaluación de las pruebas de Caja Blanca Estática en el subsistema DNC

Para evaluar los resultados de la ejecución de las pruebas de Caja Blanca Estática en el subsistema DNC, desarrollado en Symfony y ExtJS, se efectuó la aplicación de métricas para obtener el grado de mantenibilidad del código en el mismo y, finalmente, se recogieron los resultados obtenidos en el Informe final de evaluación.

- **Aplicación de métricas de mantenibilidad**

Para evaluar la mantenibilidad del subsistema DNC, se aplicaron las siguientes métricas:

- **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.

Tabla 13: Aplicación de la métrica Registrabilidad de cambios

Clase PresentarDNC	**Clase CancelarDNC**	**Clase Prorroga**
Datos: A = 0 B = 0 Cálculo: X = 0 Interpretación: 0 <= 0 <= 1 - Control de cambios deficiente. - Alta estabilidad.	Datos: A = 2 B = 6 Cálculo: X = 2/6 = 0.33 Interpretación: 0 <= 0.33 <= 1 - Cambios poco registrables. - Alta estabilidad.	Datos: A = 3 B = 6 Cálculo: X = 3/6 = 0.5 Interpretación: 0 <= 0.5 <= 1 - Cambios registrables. - Control de cambios eficiente. - Estabilidad media.
Clase RegistrarDS	**Clase Actions**	**Clase sfAuxiliarDNCTC**
Datos: A = 3 B = 6 Cálculo: X = 3/6 = 0.5 Interpretación: 0 <= 0.5 <= 1 - Cambios registrables. - Control de cambios eficiente. - Estabilidad media.	Datos: A = 3 B = 8 Cálculo: X = 3/8 = 0.38 Interpretación: 0 <= 0.38 <= 1 - Cambios poco registrables. - Alta estabilidad.	Datos: A = 0 B = 0 Cálculo: X = 0 Interpretación: 0 <= 0 <= 1 - Control de cambios deficiente. - Alta estabilidad.
Resultado		
La Registrabilidad de cambios en el subsistema en general es baja, por lo tanto el subsistema presenta baja cambiabilidad y alta estabilidad.		

- **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: Σ (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.

Datos:

T (PresentarDNC) = 0 seg

T (CancelarDNC) = 5 seg

T (Prorroga) = 5 seg

T (RegistrarDS) = 5 seg

T (Actions) = 1200 seg

T (sfAuxiliarDNCTC) = 0 seg

N = 11 fallos

Cálculo:

Σ (5+5+5+1200)/11 = 1215/11 = 110.45 seg/fallo

Interpretación:

Si 0 <= 110.45 <= 300, la conducta del mantenedor, el usuario o el sistema de software es Correcta.

Si 110.45 > 300, la conducta del mantenedor, el usuario o el sistema de software es Incorrecta.

Resultado:

La conducta del mantenedor o el sistema de software, es Correcta, debido a que la métrica Pruebas sin esfuerzo arrojó como resultado 110.45 seg/fallo, lo que equivale a tardar aproximadamente 2 minutos en encontrar un fallo. Esto indica que el sistema tiene una alta Facilidad de prueba.

- **Informe final de evaluación**

Se realizó la reunión de inicio de las pruebas, definiendo el Plan de pruebas a seguir para la realización de las mismas. Se realizaron dos iteraciones de prueba de Caja Blanca Estática en el subsistema DNC donde se aplicaron pruebas manuales mediante Listas de chequeo y pruebas automáticas mediante la herramienta JSHint, obteniendo un total de once errores durante la primera iteración la primera iteración y cero errores en la segunda iteración. Por último se calcularon las métricas para evaluar la mantenibilidad según la cambiabilidad, la estabilidad y la facilidad de pruebas del subsistema probado, determinándose que el subsistema presenta baja cambiabilidad, alta estabilidad y alta facilidad de pruebas.

3.11 Conclusiones del capítulo.

En este capítulo se llevó a cabo la aplicación del procedimiento, definido en el capítulo anterior, en el subsistema Despacho no Comercial del Sistema GINA. Se realizaron una serie de actividades donde se tuvieron en cuenta los recursos disponibles, el tiempo de duración y la cantidad de personas involucradas en el proceso de pruebas. Se realizó la configuración del entorno de pruebas quedando distribuido el hardware y el software necesario para la realización de las pruebas. Para la ejecución de las pruebas emplearon Listas de chequeo en función de evaluar el código de forma manual y también se utilizó la herramienta JSHint en función de evaluar el código de forma automática, pudiendo obtener las No Conformidades existentes y demostrando que ambas herramientas fueron factibles para la realización de pruebas de Caja Blanca Estática. Se aplicaron métricas de mantenibilidad, a través de las cuales se pudo conocer el estado de la cambiabilidad, la estabilidad y la facilidad de pruebas del subsistema DNC. Debido a los resultados alcanzados durante la aplicación en el subsistema DNC de GINA, del procedimiento propuesto, se concluye que la aplicación del mismo fue llevada a cabo satisfactoriamente.

CONCLUSIONES GENERALES

En el presente trabajo se ha definido y descrito un procedimiento de pruebas de Caja Blanca Estática para proyectos desarrollados bajo las tecnologías Symfony y ExtJS en el CEIGE. Las precondiciones establecidas en dicho procedimiento deben ser cumplidas por los desarrolladores de software, mientras que el proceso de pruebas de caja Blanca Estática, propuesto igualmente en el procedimiento mencionado, para los proyectos desarrollados en Symfony y ExtJS, puede ser aplicado por el personal del laboratorio de Calidad del centro en aras de lograr una mejora en la realización de las pruebas y aumentar la calidad de los productos.

En el desarrollo de este trabajo se cumplieron todos los objetivos planteados:

- El estudio realizado sobre las pruebas de Caja Blanca Estática a nivel nacional e internacional permitió identificar las técnicas y herramientas existentes para la realización de este tipo de pruebas.
- El estudio realizado sobre la calidad de software y sus características, proporcionó aumentar los conocimientos acerca de la mantenibilidad e identificar las métricas existentes para medir esta característica mediante el proceso de pruebas de Caja Blanca Estática.
- A través de la descripción de las actividades y los artefactos a desarrollar y las herramientas a utilizar para llevar a cabo el proceso de pruebas de Caja Blanca Estática en los proyectos desarrollados bajo las tecnologías Symfony y ExtJS, se logró mayor completitud y entendimiento de la solución propuesta.
- El procedimiento propuesto se realizó acorde a las necesidades del centro, y busca una mejora de la mantenibilidad en los subsistemas a probar.
- La propuesta fue evaluada en el subsistema Despacho No Comercial del Sistema GINA, obteniéndose buenos resultados.
- El procedimiento definido es manejable, se encuentra bien estructurado y como resultado de su aplicación se obtiene un producto con mayor calidad.

RECOMENDACIONES

Se recomienda que:

- Se apliquen otros tipos de pruebas para que haya mayor efectividad en la etapa de pruebas y mayor calidad en el software.
- Se continúe evaluando la manteniblidad de los proyectos en cuanto a otras subcaracterísticas.
- El resultado de esta investigación sea propuesto como base referencial o material auxiliar en el desarrollo de futuros trabajos.
- Se apliquen pruebas de Caja Blanca Estática en los proyectos del CEIGE que se desarrollen bajos las tecnologías Symfony y ExtJS, siguiendo el procedimiento propuesto en esta investigación, para así contribuir al aumento de la mantenibilidad de los mismos.

BIBLIOGRAFÍA

[1] Kan S.H. 1995: "Metrics and models in software quality engineering", AddisonWesley, Reading, Ma., USA, 1995.

[2] Pressman Roger S.: "Software Engineering: A Practitioner's Approach" (European Adaptation), McGrawHill. 2000. ISBN: 0077096770.

[3] Jimenez Darwin y Aguirre Carlos Eduardo: "Modelo de pruebas de software", Ingeniería en sistemas, Universidad Antonio Mariño, http://www.slideshare.net/dajigar/presentacion-pruebas--presentation, (08/01/2012).

[4] “The Home of Groundbreaking Software Quality” Management Research, http://:www.cigitallabs.com/reso/definitions/software_testing.html, (08/01/2012).

[5] IEEE, IEEE Std1995, “Metrics”, IEEE, 1991.

[6] Pressman Roger S., "Can Internetbased Applications Be Engineered?" in IEEE Software, September/October IEEE Press, 104110, 1998.

[7] Fernández Peña J. M. IPN México. “Pruebas de integración para componentes de software”, Marzo 2002.

[8] LEWIS W. E. "Software Testing and Continuous Quality Improvement" TEAM, AUERBACH PUBLICATIONS, 2005.

[9] JOHNSON R. and J. HOELLER. Expert One-on-One™ J2EE™ Development without EJB™, Wiley Publishing Inc. Indianapolis, Indiana, 2004.

[10] 2Hsoftware. Pruebas y aseguramiento de la calidad, http://www.2hsoftware.com.mx/serviciosit.html, (09/01/2012)

[11] Juristo Natalia, Moreno Ana M. y Vegas Sira. Técnicas de Evaluación de Software. 2005. 131 p.

[12] Fernández Panadero M. Carmen y Villena Román Julio. Pruebas de Programas.

[13] Pruebas de Caja Blanca, http://www.slideshare.net/juancabicho/calidad-de-software-diapositivas/presentation, (12/01/2012)

[14] Rojas J y Barrios E. Métodos de prueba de caja blanca. Grupo ARQUISOFT, 2007, http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node26.html, (09/1/2012)

[15] Pressman Roger S. Ingeniería del software, un enfoque práctico, pages 281-322. quinta Edición. McGrawHill. 2002.

[16] K.C Tai. What to do beyond branch testing. ACM Software Engineering Notes, 14:58-61, 1989.

[17] Beizer Boris. Software Testing Techniques. Van Nostrand Reinhold, second edition, 1990.

[18] Expósito Raúl, ¿Qué es el Análisis Estático del Código?

[19] Dounce Villanueva Enrique. Administración del mantenimiento. 2000

[20] Pressman Roger S. Ingeniería del software, un enfoque práctico. 3ª Edición. McGrawHill. 1993

[21] Knezevic, J., Reliability, Maintainability and Supportability Engineering- A Probabilistic Approach, pág. 292, plus software PROBCHAR, McGraw Hill, Londres (Inglaterra). 1993.

[22] Ruiz, F. y Polo, M., Grupo Alarcos, Dep. 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, (02/02/2012)

[23] Ghezzi Carlo, Jazayeri Mehdi, Mandrioli Dino. Fundamentals of Software Engineering –. Prentice-Hall, Inc. 1991, edición en inglés. ISBN-0-13-820432-2, Capítulo 1 – Software: its nature and qualities.

[24] Instituto Tecnológico de Aguascalientes. Lic. En Informática Sistemas de Información II, 2012, Mantenibilidad del software.html, (03/02/2012)

[25] Knezevic, J., Mantenibilidad, Publicaciones de Ingeniería de Sistemas, c/ Edison, 4, 28006 Madrid, Febrero - 1996

[26] Ingeniería de Software de Gestión III. Ingeniería de Software de Gestión III. Tema 5: Gestión de Proyectos Software Métricas. Dpto. de Lenguajes y Sistemas Informáticos. Escuela técnica superior de ingeniería informática (etsii). Universidad de Sevilla.

[27] Gonzalo Mena Mendoza ISO 9126-3: Métricas Internas de la Calidad del Producto de Software. Estándares de Calidad. Querétaro, marzo de 2006. http://mena.com.mx/gonzalo/maestria/calidad/presenta/iso_9126-3/

[28] Fernández, Giovanny. Estándar codificación DOTNET. España. 2005. 20 p.

[29] Potencier, Fabien y Zaninotto, François, Symfony 1.2, la guía definitiva, 2008.

[30] Rubira Jorge. JSLint y JSHint, analizadores de código javaScript online. 10 de julio de 2011. http://www.jslint-y-jshint-analizadores-de-codigo-javascript-online.htm

[31] Scott Mitchell. Compruebe el JavaScript. MSDN Magazine. Enero 2010.

[32] Benítez Carlos, JSHint, herramienta para medir la calidad del código Javascript, 19/2/2011, http://www.etnassoft.com/2011/02/19/jshint-herramienta-para-medir-la-calidad-del-codigo-javascript/

[33] JavaScript Lint http://www.versioncero.com/noticia/324/javascript-lint

[34] Alma Fernández, JavaScript Lint: Limpia de errores tus scripts. http://www.webmasterlibre.com/2006/08/22/javascript-lint-limpia-de-errores-tus-scripts/

[35] Linstone, Harold A y Turoff, Murray. The Delphi Method. Techniques and Applications. [En línea] 2002. http://is.njit.edu/pubs/delphibook/delphibook.pdf.

[36] Dalkey, Norman C, Brown, Bernice y Cochran, S. The Delphi Method, III: Use of self rating to improve group estimates. Technological Forecasting and Social Change. [En línea] noviembre de 1969. http://www.rand.org/pubs/research_memoranda/2006/RM6115.pdf.

[37] Picado Alvarado, Federico.Análisis de concordancia de atributos. [En línea] diciembre de 2008. http://www.tec.ac.cr/sitios/Vicerrectoria/vie/editorial_tecnologica/Revista_Tecnologia_Marcha/pdf/tecnologia_marcha_21-4/cap%203.pdf.

[38] Moráguez Iglesias, Ing. Arabel. El método Delphi. GestioPolis. [En línea] 2006. http://www.gestiopolis.com/canales6/eco/metodo-delphi-estadistica-de-investigacion-cientifica.htm.

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

Alfonso Febles Anel y Alomá Benítez Denia Francisca. “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.

Galafet Céspedes Dailin y Martínez Amador José J. Proceso de diagnóstico para las organizaciones productivas de la UCI. Universidad de las Ciencias Informáticas. La Habana, 2008. 110 pág.

Gómez Arenas Liliana del S. Metodología para evaluar la calidad de los sistemas de información. ParqueSoft, Cali, Octubre de 2004.

IPP 1000:2008 Elaboración y aprobación de los procedimientos y lineamientos para la actividad productiva.

Isaac Morales Dayanis. “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.

ISO 9126: Software Product Evaluation: Quality Characteristics and Guidelines for their Use. Suiza, 1991.

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

Morales Mejía Johana Andrea y Sierra González Ana María. Plan de negocio para la creación de una empresa de servicio y asesoría de pruebas de software. Universidad tecnológica de Pereira. 2010.

Quintas Sánchez Elizabeth. “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.

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

Ruiz Francisco y Polo Macario. “Mantenimiento del software”. Upm. 2007. 109 pág. Master en ingeniería del software”.

Suárez Pablo y Fontela Carlos. Documentación y pruebas antes del paradigma de objetos. 2003.