Universidad de las Ciencias Informáticas

Facultad 6

Título: “Procedimiento para la evaluación y perfeccionamiento de la calidad de los archivos fuentes en PHP y Java Script”.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Yosvel Vasallo Tabares

Tutor: Ing. Omar Ahmed García Pérez

Ciudad de La Habana, junio 2011

... un nacimiento representa el principio de
todo, es el milagro del presente y la esperanza
del futuro...
Bienvenido a este mundo hija mía.
Prometo estar unido a tí para siempre.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo titulado: “Procedimiento para la evaluación y perfeccionamiento de la calidad de los archivos fuentes en PHP y Java Script” y autorizo a la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio.

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

Yosvel Vasallo Tabares

_____

Firma del autor

Ing. Omar Ahmed García Pérez

_____

Firma del tutor

DATOS DE CONTACTO

Tutor:

- Ing. Omar Ahmed García Pérez

Universidad de las Ciencias Informáticas

Email: oagarcia@uci.cu

Autor:

- Yosvel Vasallo Tabares

Universidad de las Ciencias Informáticas

Email: yvasallo@estudiantes.uci.cu

AGRADECIMIENTOS

A mi mamá y mi papá por darme apoyo incondicional y hacerme creer que yo era el más inteligente de la UCI, por siempre estar ahí cuando más le necesito, por exigir cuando hizo falta y aflojar la mano en el momento más preciso y por trabajar tanto para permitirme estudiar sin preocupaciones.

A mis hermanos por hacerme despejar cuando la cabeza se me quería reventar y me dolían los ojos de tanto leer.

A Gisse, por las horas de sueño dedicadas a los Informes de Evaluación, por aguantar mis candangas con la tesis y por oír mis exposiciones una y otra vez.

A mis abuelos y tíos por las velas derretidas en cada corte de tesis y los vasos de agua puestos a todos los santos del mundo tratando de ayudarme.

A Yordi por darme un empujoncito de vez en cuando, siempre cuando más lo necesité.

A mi tutor Omar, que me ayudó a pesar de estar lleno de trabajo.

A los Kremas:
Dariel (el cabeza), Rodolfo (el Puro), Rey (Candito), Daniel (el Tío), por estar al lado mío en las cosas malas y buenas que me han pasado en esta escuela, por ser mis cuatro nuevos hermanos desde hace cinco años, por aguantar mis pesadeces etc., etc., etc…

AGRADECIMIENTOS

A los Asaltaparques (Yaspa, Monti, Juan José, Leo, Fredy y Jorge), el dream team que luchó por ganar la última copa de futsal corriendo el riesgo de ser aplazados para diciembre solo por despejar un poco y quitarse el estrés de la tesis.

… a todos los demás que han tenido que ver de una forma u otra con el total desarrollo de este trabajo, a los que confiaron en mí y fueron siempre mi apoyo emocional…

… Gracias.

DEDICATORIA

A mi hija, Mia Fernanda Vasallo Breto.

"… nena, apenas te conozco y ya te quiero más que a mí mismo, eres todo en mi vida…"

RESUMEN

El presente trabajo propone un procedimiento para la evaluación y perfeccionamiento de la calidad de los archivos fuentes escritos en lenguaje Java Script y PHP. Durante la confección del mismo primeramente se realizó un estudio de las principales normas de evaluación y codificación, además de una búsqueda de herramientas para automatizar la evaluación. El procedimiento propuesto, adaptado a las principales metodologías utilizadas en la Universidad de Ciencias Informáticas, propone un grupo de actividades con un orden lógico para mejorar la legibilidad del código fuente, eliminar código duplicado y otros problemas que se cometen cuando se programa. Define de manera detallada cada una de las actividades, los roles y artefactos necesarios para su documentación; todo lo anterior en función de brindar una guía práctica y objetiva en su utilización.

Palabras Claves: Calidad, calidad interna, estándar de codificación, normas de evaluación, herramientas de análisis estático, procedimiento, perfeccionamiento de código.

Índice de Contenidos

Índice de Figuras

Introducción

El diseño y desarrollo de productos de buena calidad es un factor clave para lograr la supervivencia y el éxito de la mayoría de las empresas sin importar su tamaño, capital, presencia en el mercado o el tipo de producto que en estas se desarrolle. El objetivo de cada una de ellas se centra principalmente en construir sus productos de una manera rápida, eficiente y con costes de tiempo adecuado, esto es un reto difícil de lograr, dado que en algunas ocasiones las demandas de los clientes son mayores que las capacidades productivas de estas empresas.

En la actualidad, todo se consuma en torno al software e Internet, esto ha provocado que el desarrollo de productos de software se convierta en una de las principales plataformas de conocimiento, crecimiento profesional y corporativo de la actualidad, razón por la cual la construcción de estos sistemas debe ser prácticamente perfecto, y se deben seguir ciertos factores de calidad que así lo aseguran. Son especialmente las empresas productoras de software las que muchas veces producto de la demanda se ven presionadas en cuanto a la rapidez con que deben ser desarrollados cada uno de los productos ya que la producción de software de forma industrial es prácticamente un mito que solo pocas entidades pueden respaldar.

Es en ese momento cuando se comienza a hacer todo con prisa y sin prestarle la debida atención a temas tan importantes como son mantener un buen estilo de diseño, correcto uso de normas de codificación, la aplicación de pruebas unitarias, las pruebas funcionales y las refactorizaciones de código necesarias para obtener un producto final con la debida calidad. Sin esa atención el sistema generalmente se desarrolla con una serie de defectos; surgen los *Code Smell*[1] (olor del código) que no son más que malos olores o problemas como surgimiento de métodos excesivamente largos, los comentarios en el código, el código duplicado, clases muy grandes y complejas, clases con mucha relación entre otras y clases que no son ejecutadas con mucha frecuencia producto de un mal uso de los patrones ***GRASP***[2]; producto de los cuales el sistema se convierte en un código imposible de comprender, estructuralmente complejo, lleno de condiciones encadenadas y métodos con cientos e

[1] *CodeSmell, es un concepto o término desarrollado y popularizado en el mes de febrero del año 2001 por Martin Fowler y Kent Beck, ambos figuras reconocidas cuando de desarrollo de software se habla, para describir algunos problemas que surgían en el código de las aplicaciones.*

[2] ***GRASP**, son patrones generales de software para asignación de responsabilidades.*

incluso miles de líneas de código, afectando la calidad de codificación interna del software, y lamentablemente que el código cumpla su cometido funcional no garantiza que cumpla su cometido no funcional, es decir, no garantiza que sea escalable, mantenible, extensible o incluso legible.

El proceso de desarrollo de software es intensamente intelectual, viéndose fuertemente afectado por la creatividad y juicio de las personas involucradas (1). En este proceso, el cual tiene como propósito la producción eficaz y eficiente de un producto que reúna los requisitos del cliente (2), existe una fase dedicada al mantenimiento, la cual implica cambios para eliminar errores o también para añadir nuevas funcionalidades, adaptarse a los cambios en el medio ambiente o mejorar la estructura del programa.

Garantizar dicha mantenibilidad implica el cumplimiento de un grupo de parámetros. Estos parámetros son, la adecuada cualificación del equipo de Desarrolladores del software, empleo de una documentación con una estructura estandarizadas, sumo cuidado a la hora de realizar el diseño, codificación y prueba, y principalmente garantizar la legibilidad, es decir, que todo esté claramente documentado, espaciado, sin errores, y con una facilidad de uso ágil y de rápido entendimiento. Así se logra una mayor comprensión del proyecto, y las modificaciones pertinentes son más fáciles de realizar.

Dado que un software en sí es complejo, es prácticamente inviable conseguir un 100% de confiabilidad de un programa por pequeño que sea. Además, un software es intangible y por lo general muy abstracto (1), esto hace difícil la propia definición del producto y de sus requisitos, por lo que los cambios ocurren continuamente en la mayoría de los programas tanto dentro de su fase de desarrollo como después de ser desarrollados; sin embargo, esta es una actividad difícil, ya que el esfuerzo necesario para hacer un cambio a menudo está fuera de proporción a la magnitud del cambio. De hecho, un estudio de 487 empresas se encuentra que más de 70% del esfuerzo total en el ciclo de vida del software se destinaron a actividades de mantenimiento y es precisamente la incapacidad de lograr la mantenibilidad la que no permite al software llegar a ser escalable y extensible (3).

Soluciones a este problema se han ido buscando en todas las direcciones perfeccionándose distintos procesos y metodologías de desarrollo de software, se sigue trabajando fuertemente en la capacitación de los recursos humanos tratando de crear una base de buena práctica de programación; constantemente se desarrollan nuevas tecnologías para facilitar el trabajo y automatizar algunos elementos y de esta forma evitar el surgimiento de algunos de estos problemas en las aplicaciones. No obstante, estos son problemas que siguen presentando la mayoría de los equipos de desarrollo de

software a la hora de efectuar su trabajo. Es por ello que una de las áreas en las que se debe seguir mejorando, es en el perfeccionamiento del software en cuanto a calidad de los archivos fuentes se refiere; siendo el uso de herramientas un elemento clave en la calidad, rapidez y eficiencia con que se descubren y suprimen los *Code Smell* que no por ser errores imperceptibles por el usuario ni capaces de descomponer un programa dejan de ser perjudiciales para la salud del producto.

La Universidad de la Ciencias Informáticas (UCI) es una institución universitaria de reciente creación que cuenta con el desarrollo de software entre sus objetivos, proceso que se realiza basado en un modelo que vincula la formación de pregrado de estudiantes y la investigación (4). En este contexto, la universidad se presenta como una institución que presenta un alumnado que sin disponer de experiencia suficiente en el desarrollo del software trabaja en conjunto con profesores y trabajadores para lograr cumplir con los pedidos realizados por instituciones que buscan automatizar tareas para mejorar su trabajo. Para la ejecución de esta tarea la UCI se divide por facultades y un grupo de centros dedicados a la producción de software. Estos centros productivos tienen la misión de desarrollar diversas soluciones informáticas para consumo nacional e internacional, requiriendo de corto o largo tiempo para llevar a cabo dicha tarea; caracterizados por incorporar productos utilizando principalmente lenguajes como PHP y Java Script cuya construcción requiere de un gran volumen de código fuente y diversas personas trabajando sobre el mismo simultáneamente.

Lógicamente a medida que todo sistema evoluciona, comienza a crecer la cantidad de código lo que conduce a inconsistencia lo cual provoca que surjan problemas en este, se repiten fragmentos de código en diferentes partes del software y se violan normas producto de lo cual se ve afectada:

- La comprensión: las secciones repetidas del código que solo se diferencian en algunas líneas o caracteres aumentan la longitud y puede hacerlo difícil de entender rápidamente además del no cumplimiento de las normas de codificación.

- El costo de mantenimiento después de desplegado el software: el mantenimiento se vuelve muy dificultoso ya que el mismo código debe ser arreglado en diferentes partes.

- El tamaño del producto: provoca un incremento en el tiempo de ejecución además de un incremento del tamaño del software producto de la cantidad de líneas de código.

Por lo anteriormente expuesto se plantea como **problema científico de la investigación**: Las deficiencias detectadas en los archivos fuentes escritos en PHP y Java Script de los productos de

software de la UCI están afectando la calidad en lo referente a la comprensión de los archivos fuentes, el coste de mantenimiento de estos y el tamaño del producto.

Se reconoce como **objeto de estudio**: La calidad del software; dentro del mismo se selecciona como **campo de acción**: Calidad de los archivos fuentes de un producto de software, planteándose como **objetivo general**: Elaborar un procedimiento para evaluar y perfeccionar la calidad de los archivos fuentes escritos en lenguaje PHP y Java Script para los productos de software de la UCI, del cual se derivan los siguientes objetivos específicos:

- Realizar la fundamentación teórica de la investigación.
- Confeccionar la total estructura del procedimiento de evaluación y perfeccionamiento.
- Validar el procedimiento a partir de su aplicación en el proceso de desarrollo de productos de software.

Con el propósito de dar cumplimiento a estos objetivos específicos se definen las siguientes **tareas de la investigación**:

- Revisión bibliográfica en busca de investigaciones similares y valoración del aporte que pudieran brindar.
- Estudio de las principales normas y estándares existentes para la evaluación de la calidad del software.
- Determinación de las normas o estándares de evaluación a utilizar en la investigación.
- Identificación y/o definición de principales métricas a utilizar para la medición de un producto de software.
- Análisis de las herramientas que intervienen en el procedimiento a aplicar.
- Definición de las principales fases que contendrá el procedimiento.
- Definición de las actividades a efectuar en cada fase para evaluar y perfeccionar la calidad de los archivos fuentes escritos en lenguaje PHP y Java Script para los productos de software de la UCI.

- Selección de un producto específico para la aplicación del procedimiento.
- Aplicación del procedimiento obtenido.
- Análisis de los resultados obtenidos después de la aplicación del procedimiento

La presente investigación está estructurada en tres capítulos:

- Capítulo 1: Fundamentación Teórica.

En este capítulo se presenta la definición del marco teórico de la investigación, se exponen una serie de conceptos y definiciones necesarias para un mejor entendimiento del trabajo, se efectúa un análisis de algunas normas y métricas existentes utilizadas para llevar a cabo la evaluación de la calidad del software. Además de una breve descripción de las herramientas utilizadas para la evaluación de la calidad de los archivos fuentes.

- Capítulo 2: Propuesta del procedimiento.

En este capítulo se describe de forma detallada el procedimiento obtenido para la evaluación y perfeccionamiento del código fuente dándole cumplimiento de esta forma al objetivo principal del trabajo.

- Capítulo 3: Validación del procedimiento.

En este capítulo el procedimiento desarrollado se aplica en uno de los productos desarrollados en la UCI. Se realiza la evaluación de los resultados después de medido, se exponen los errores encontrados y se realizan las refactorizaciones pertinentes.

Capítulo 1: Fundamentación Teórica

Introducción

En el presente capítulo se realizará la definición del modelo teórico de la investigación. Se tratarán temas de gran importancia para el Evaluador, se estudiarán una serie de conceptos y definiciones necesarias para un mejor entendimiento del trabajo, realizando conjuntamente un análisis de algunas normas y métricas existentes utilizadas para llevar a cabo la evaluación de la calidad del software, así como las diferentes pruebas y medidas que se le efectúan al código, centrándose mayormente en la calidad interna de los productos.

1.1 Calidad

El término calidad tiene múltiples significados y como concepto, ha cambiado y evolucionado con el tiempo. Varias son las definiciones que le han dado personalidades de gran prestigio internacional a la calidad, a los cuales les avala toda una experiencia en esta área. Pero no solo son personalidades las que definen este concepto sino también organizaciones rectoras de estándares, que hacen una valoración sobre cómo se define la calidad como concepto (5).

La calidad se puede definir como la capacidad de lograr objetivos de operación buscados, teniendo como principios básicos la prevención y las mejoras continuas del producto. Esto significa que la calidad es un proyecto interminable, cuyo objetivo es detectar disfunciones tan rápido como sea posible después de que ocurran (6).

La norma ***ISO***[3] 8402-94 define la calidad como: *"El conjunto de características de una entidad que le otorgan la capacidad de satisfacer necesidades explícitas e implícitas."*

La norma ***ISO*** 9000:2000 la define como: *"La capacidad de un conjunto de características intrínsecas para satisfacer requisitos."*

Otras definiciones más completas o formales:

[3] ***ISO,*** *The International Organization for Standardization, es una organización dedicada a la estandarización.*

Kaoru Ishikawa, filósofo japonés de la administración de empresas, plantea que calidad significa desarrollar, diseñar, producir y mantener un producto de calidad que sea el más económico, el más útil y siempre satisfactorio para el consumidor. William Edwards Deming, estadístico estadounidense y profesor universitario, en su libro *"Out of the Crisis"*[4] expone que calidad es la aplicación de los principios y técnicas estadísticas en todas las fases de la producción, dirigida a la fabricación más económica de un producto (servicio) que es útil en grado máximo y que tiene mercado (7).

En la práctica, hay dos tipos de calidad:

- Calidad externa: Corresponde a la satisfacción de los clientes. El logro de la calidad externa requiere proporcionar productos o servicios que satisfagan las expectativas del cliente para establecer lealtad con el cliente y de ese modo mejorar la participación en el mercado. Los beneficiarios de la calidad externa son los clientes y los socios externos de una compañía. Por lo tanto, este tipo de procedimientos requiere escuchar a los clientes y también debe permitir que se consideren las necesidades implícitas que los clientes no expresan (6).

- Calidad interna: Corresponde al mejoramiento de la operación interna de una compañía. El propósito de la calidad interna es implementar los medios para permitir la mejor descripción posible de la organización y, detectar y limitar los funcionamientos incorrectos. Los beneficiarios de la calidad interna son la administración y los empleados de la compañía. La calidad interna pasa generalmente por una etapa participativa en la que se identifican y formalizan los procesos internos (6).

Por consiguiente, el propósito de la calidad es proporcionarle al cliente una oferta apropiada con procesos controlados y al mismo tiempo garantizar que esta mejora no se traduzca en costos adicionales.

1.1.1 Calidad de Software

La calidad de software ha sido usada desde un simple argumento de venta, hasta verdaderos estudios formales para el desarrollo de software. Dicha calidad es muy complicada de definir y de enmarcar en un simple concepto teórico aunque existen algunas características que permiten describirla (5).

[4] *Fuera de la Crisis.*

CAPÍTULO I

En la actualidad todas las instituciones y empresas que buscan el desarrollo en sus producciones necesitan que sus productos posean una calidad con la cual podrán obtener un prestigio a nivel mundial. La calidad de software es la línea que siguen los clientes hoy en día, ya que estos para adquirir cualquier producto para sus beneficios primero tienen en cuenta sus cualidades. Muchos han sido los profesionales que han querido profundizar en este sentido e intentando evaluar la calidad de los productos que se diseñan.

R.S. Pressman (1992) la define como: *"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente".*[5]

ISO/IEC DEC 9126 expone que la calidad es: *"La totalidad de características de un producto de software que confieren su aptitud para satisfacer necesidades explícitas o implícitas".* Esta institución especifica características como funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad que definen la calidad real del software.

En este contexto es posible expresar que la calidad del software se refiere al desarrollo de software basado en estándares con la funcionalidad y rendimiento total que satisfacen los requerimientos del cliente, (5) tomando a estos como base de la medida de calidad y cumpliendo para su desarrollo con algunos requisitos implícitos o expectativas que a menudo no se mencionan, o se mencionan de forma incompleta (por ejemplo el deseo de un buen mantenimiento).

Pressman hace unas definiciones muy acertadas acerca de la calidad de software, sobre cómo debe concebirse, junto con él también lo hace la norma ISO/IEC DEC 9126, ambos coinciden con lo mencionado por Kaoru Ishikawa sobre diseño, construcción con calidad manteniendo un producto económico, pero ***ISO*** habla además en su definición sobre algunos rasgos significativos como funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad que se ajusta a los rasgos de la investigación por lo que se asume esta definición como base teórica de la investigación con respecto a calidad.

1.2 Métricas

[5] *Citado de R. S. Pressman. 1998. Aspectos y métricas de la calidad del Software.*

Aunque los términos medida, medición y métricas se utilizan a menudo indistintamente, existe gran confusión a la hora de referirnos a ellos.

Las métricas son medidas que se le aplica a un producto, estas medidas proporcionan una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto (8).

La ***IEEE***[6] en "*Standard Glossary of Software Engering Terms*"[7] define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado (9).

Atendiendo a los conceptos antes mencionados se concluye que una métrica es una forma de evaluar y una escala, definidas para realizar mediciones de uno o varios atributos.

1.2.1 Métricas de Software

La medición del software es importante ya que permite a los administradores y Desarrolladores a entender mejor el proceso de desarrollo, así como la calidad del software que se produce (10). Estas medidas cuantifican los atributos de un el producto. Ejemplos de atributos que se pueden medir son el tamaño, la complejidad y el importe de la reutilización. Existen métricas para medir el proceso, el proyecto o el producto. Para el caso de medir el producto, existen tres formas de realizar mediciones, se puede medir utilizando métricas internas, externas o en uso, estas dos últimas son aquellas aplicables al software en ejecución (11).

1.2.2 Métricas internas

Las métricas internas pueden aplicarse a un producto de software no ejecutable (como una especificación o código fuente) durante el diseño y la programación. Las métricas internas miden los atributos interiores o indican los atributos externos por medio del análisis de las propiedades estáticas del producto de software intermedio o final. En el desarrollo de un producto de software los productos intermedios deben evaluarse usando las métricas internas que miden las propiedades intrínsecas, incluyendo las que pueden derivarse de un comportamiento simulado. El propósito primario de estas

---

[6] ***IEEE,*** *Siglas en inglés de The Institute of Electrical and Electronics Engineers (Instituto de ingenieros eléctricos y electrónicos).*

[7] *Glosario Estándar de Términos de Ingeniería de Software.*

métricas internas es asegurar el logro de la calidad externa y la calidad durante el uso requerido (12). Las métricas internas constituyen una ventaja para los usuarios, Evaluadores, verificadores, y diseñadores, pues le permiten evaluar la calidad de producto de software y atender los aspectos de la calidad desde las etapas más tempranas. Cada métrica contiene:

- ✓ Nombre.
- ✓ Propósito.
- ✓ Método de aplicación.
- ✓ Medida, fórmula y cómputo de datos.
- ✓ Interpretación del valor medio.
- ✓ Tipo de escala.
- ✓ Tipo de medida.
- ✓ Fuente de medición.
- ✓ Audiencia.

Ejemplo de métrica (12):

Nombre: Registrabilidad de cambios.

Propósito: ¿Se registran adecuadamente los cambios a la especificación y a los módulos con comentario en el código?

Método de aplicación: Registrar la proporción de información sobre cambios a los módulos.

Formula: X=A/B

- A: Número de cambios a funciones o módulos que tienen comentarios confirmados.
- B: Total de módulos o funciones modificadas.

Interpretación: 0 <= X <= 1

- Entre más cercano a 1 más registrable.

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

Tipo de escala: Absoluta.

Tipo de medida:

- X= count/count
- A= count
- B= count

1.2.3 Taxonomías de métricas

- Métricas básicas: Se obtienen directamente de analizar el código o la ejecución del software, obteniendo del propio código fuente todos los valores que en ella se utilizan (13).
- Métricas de agregación: Consisten en la creación de una métrica a partir de un conjunto definido de métricas básicas, generalmente mediante una suma ponderada, en este tipo de métricas a diferencia de las básicas no se analiza el código fuente para obtener valores, sino que se recogen valores ya establecidos por otras métricas y se combinan para medir una característica (13).
- Métricas derivadas: Es una función matemática que utiliza como entrada el valor de otras métricas, estas métricas al igual que las métricas de agregación obtienen sus valores a partir de otras métricas, pero se diferencia en que esta utiliza dichos valores en la función matemática, no combinándola (13).

1.2.4 Métricas para el código fuente

- **LOC (*Lines Of Code*[8]).**

Desde el comienzo los ingenieros de software han estado contando las líneas de códigos que ellos escribían. El conteo de las líneas es utilizado para estimar el mantenimiento necesario y puede ser utilizado para normalizar otras métricas de software (14).

[8] *Line of code, Líneas de Código.*

El número de líneas de código (LOC) es una de las métricas más usadas. Parece sencilla pero no lo es tanto, ya que existen diferentes alternativas:

- contar sólo las líneas de código ejecutable, añadirle las declaraciones de datos.
- contar las líneas de comentarios, añadir las sentencias de control y otras.

De hecho, suele ser frecuente, que al usar diferentes programas de métricas, estas generen diferentes valores de LOC para un mismo fichero.

El uso principal que tiene contar líneas de código, es poder determinar proporciones entre líneas de código y fallos, es decir, generar medidas (y heurísticas) sobre el número de líneas de código y el número de fallos de una aplicación (15).

- **Complejidad ciclomática.**

La métrica de la complejidad ciclomática fue diseñada por McCabe (1976) para indicar la facilidad para testear[9] y entender el código. La complejidad ciclomática se basa en el recuento del número de caminos lógicos individuales contenidos en un programa. Para calcular la complejidad del software, Thomas McCabe utilizó la teoría y flujo de grafos. Para hallar la complejidad ciclomática, el programa se representa como un grafo, y cada instrucción que contiene, un nodo del grafo. Las posibles vías de ejecución a partir de una instrucción (nodo) se representan en el grafo como aristas (16).

Una razón para el amplio uso de la métrica de la complejidad ciclomática es que puede ser fácilmente calculada por el análisis estático y es un método ampliamente utilizado en la industria para el control de calidad (17).

Existen numerosos estudios empíricos que han demostrado una alta correlación entre código con fallos y código con alta complejidad ciclomática.

- **Código duplicado.**

[9] *Someter el software a pruebas.*

Cuando el código se duplica puede llegar a ser más difícil hacer cambios en el software porque el mismo cambio también debe hacerse en todas las copias. Esto toma más tiempo y también es propenso a errores, ya que es fácil olvidarse de hacer el cambio en varios lugares.

Una forma de medir código duplicado es por el método de Baker[10] (18) o también por *matching layout*[11] (19).

Un sistema de software puede tener código fuente en varios lugares que ofrecen la misma funcionalidad. Esto se denomina conceptualmente duplicar código. El duplicado de código no se puede detectar de forma automática y por lo tanto son detectados por las inspecciones realizadas por los seres humanos con el uso de herramientas.

Existen además otras métricas menos comunes para medir la calidad del código fuente como:

Excessive Method Length.[12]

Es otro tipo de medida que se le efectúa al código. Por lo general indican que el método está haciendo demasiado y debe reducir el tamaño de método mediante la creación de métodos auxiliares y la eliminación de cualquier copia / pega de código.

***Too Many Methods*.**[13]

Indica una clase con muchos métodos y muy probablemente a un sospechoso bueno para refactorización, a fin de reducir su complejidad y encontrar una manera de tener objetos menos complejos.

1.3 Norma o estándar

Según ISO, un estándar es un conjunto de acuerdos documentados que contienen especificaciones técnicas u otros criterios precisos para ser usados constantemente, como reglas, lineamientos o

[10] *Brenda S. Baker, personalidad reconocida en el mundo por la creación de algoritmos para solucionar problemas.*

[11] *Matching Layout es una técnica de detección de código duplicado, se entiende como detección de copias.*

[12] *Método demasiado grande.*

[13] *Excesiva cantidad de métodos.*

definiciones de características. Todo esto con la finalidad de asegurar que los materiales, productos, procesos y servicios son óptimos para su propósito (20). Es el documento aprobado por consenso por un organismo reconocido, que proporciona reglas, pautas y/o características para uso común, con el objeto de obtener un óptimo nivel de resultados en un contexto dado (21).

Se define como toda actividad documentada que norma el comportamiento de un grupo de personas. Los estándares brindan los medios para que todos los procesos se efectúen siempre de forma similar. Son una guía para la productividad y la calidad. Es decir, las normas son un modelo, un patrón, ejemplo o criterio a seguir. Cada norma tiene un campo de validez que define la aplicación. Por esta razón, muchos productos están sujetos a muchas normas. Una norma tiene valor de regla y tiene por finalidad definir las características que deben poseer un objeto y los productos que han de tener una compatibilidad para ser usados a nivel internacional. La estandarización o norma se basa en el trabajo conjunto de todas las partes involucradas: productores, profesionales, usuarios y administración pública. Además, recoge los deseos, las propuestas de todas las instituciones relevantes como son los fabricantes, las asociaciones de consumidores, los juristas, los centros de investigación, las entidades de certificación e inspección. Las normas cubren todos los aspectos técnicos relacionados con la información, su producción y su gestión. Son coherentes y consistentes. Han sido desarrolladas por comités técnicos bajo supervisión de un organismo especializado.

1.3.1 Bondades de las normas o estándares

Los estándares permiten reducir considerablemente los costos que surgen en el momento de implementar sistemas que van desde la anexión de un nuevo recurso hasta la analizabilidad de los sistemas legados que existen en cualquier organización. Si todos los desarrolladores programan como quieren, cualquiera utiliza su propio estándar o toma cualquier estándar que use otro proyecto, indudablemente esto significará un posible caos o incoherencia en el código fuente.

1.3.2 Normas o Estándares de Codificación

Para garantizar la calidad de un producto, existen normas que funcionan como reglas a seguir (22), aunque cada empresa puede contar con normas internas específicas.

Adoptar un estándar permite enfocarse más en el código. Aporta consistencia pues las diferentes porciones de código que pueden ser provenientes de diversos lenguajes guardan un patrón que se puede reconocer.

Mejorar la legibilidad del código permite que otras personas puedan colaborar más eficazmente, pues la redacción del código no les resulta incómoda. En consecuencia, el mantenimiento es menos pesado como resultado de los aportes anteriormente mencionados.

El valor a largo plazo que brinda el software a una organización está en proporción directa a la calidad del código fuente. Durante su vida útil, un programa será manejado por muchos pares de manos y ojos. Si un programa es capaz de comunicar claramente su estructura y características, es menos probable que se rompa cuando se modifica en un futuro no muy lejano. Las convenciones de código pueden ayudar a reducir la fragilidad de estos programas (23).

"Buenas normas de codificación son importantes en cualquier proyecto de desarrollo, pero sobre todo cuando varios programadores están trabajando sobre el mismo proyecto. Usar normas de codificación ayuda a asegurar que el código es de alta calidad, tiene menos bugs[14], y es fácil de mantener".[15]

1.3.3 Prácticas generales de codificación

- Indentación[16]

Para todos los lenguajes de código, la indentación se realiza utilizando el caracter espacio. La realización de un *Tab* dentro del código es equivalente al ingreso de cuatro espacios.

Sin importar el lenguaje de programación, toda la indentación debe hacerse a 4 espacios.

- Legibilidad vs. Compresión

Se prefiere la legibilidad al ahorro en el tamaño de los archivos cuando se trata de mantener los archivos existentes. La utilización de espacios en blanco está recomendada junto con la utilización de los caracteres ***ASCII***[17] donde sea apropiado.

---

[14] *Bugs, es un defecto de software (computer bug en inglés), es el resultado de un fallo o deficiencia durante el proceso de creación de programas*

[15] *Citado el día 10 de enero del 2011 de http://phpsenior.blogspot.com/2008/07/estndares-o-muerte-para-php.html*

[16] *Significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores con el fin de mejorar la legibilidad.*

[17] ***ASCII**, acrónimo en inglés de American Standard Code for Information Interchange, cuyo significado es Código Estadounidense Estándar para el Intercambio de Información.*

No hay ninguna necesidad o propósito que lleve al Desarrollador a obstruir el código Java Script. Se deben usar procesos del lado del servidor que compriman automáticamente todos los archivos estáticos del lado del cliente como CSS y Java Script.

- Idioma del código

Todo el código fuente debe estar escrito en el idioma inglés. Esto incluye el nombre de módulos, componentes, clases y variables.

La documentación de los métodos debe estar en el idioma nativo de la persona que lo escribió, esto para mejorar la comprensión de la documentación.

1.3.4 PHP. Estándares de codificación

PHP, por sus siglas *Hypertext Preprocessor*[18], es un lenguaje interpretado de alto nivel, incrustado en páginas HTML y ejecutado en el servidor. Este permite hacer cualquier cosa que se pueda hacer con un script ***CGI***[19] como procesar la información de los formularios, generar páginas con contenido dinámico o mandar o recibir *cookies* entre otras. Una de las características más importantes de PHP es su capacidad de soporte para un gran número de bases de datos.

Para el trabajo en PHP existen varios documentos que pueden ayudar a los Desarrolladores en cuanto a la forma en que deben escribir el código. En el sitio PHP.net, de conformidad con el manual de ***PEAR***[20], hay un gran documento que resume las normas que se recomiendan para todo el código PHP. Estas normas se basan en estándares JavaDocs[21]con solo algunas excepciones. Existen otros documentos que describen estándares de codificación como el escrito por Frederick Kristiansen de DB/Media Lab en Noruega (24). Su versión parece ser una traducción basada en PHP de las normas sobre la base de *Todd C's Hoff Coding Standard* (25). Otro recurso posible para los Desarrolladores

---

[18] *Procesador de hipertexto.*

[19] ***CGI**,Common Gateway Interface*

[20] ***PEAR**, es un framework y sistema de distribución de componentes reutilizables PHP. La misión de PEAR es proporcionar los componentes reutilizables, difundir mejores prácticas para el desarrollo de PHP y educar a los Desarrolladores.*

[21] *Javadoc, utilidad de Sun Microsystems para la generación de documentación de APIs en formato HTML a partir de código fuente Java.*

son las normas escritas para trabajar en el PHP *Zend Framework*. Sin embargo, estos documentos parecen ser algo específicos y son un poco estrictos en lo que es y no es aceptable. Por lo que resultaría mejor una guía suelta condicionada por las propias necesidades del Desarrollador, y guiada por un documento existente que se adecúe mejor a las características del desarrollo.

Siguiendo con la primera propuesta realizada en el manual de ***PEAR***, PHP ofrece un software que automáticamente analiza el código fuente y la producción eficaz de un manual de trabajo basado en los comentarios dentro del código. El software es nombrado phpDocumentor[22]y está disponible de forma gratuita a cualquier Desarrollador que desee utilizarla además de otras herramientas que proporcionan ayuda en lo referente a la producción eficaz de software.

1.3.4.1 Principios generales de codificación en PHP (26)

Para los archivos que solo contienen código PHP, la etiqueta de cierre ("?>"), no está permitida. Esta no es requerida por PHP, y omitiéndola prevenimos la inyección accidental de espacios en blanco en la respuesta.

- Longitud máxima de línea

La longitud del fin de la línea es de 80 caracteres. Se debe tratar de mantener cada línea de código bajo 80 caracteres cuando sea posible y práctico. Sin embargo, líneas más largas son aceptadas en algunas circunstancias. La longitud máxima de cualquier línea de código PHP es de 120 caracteres.

- Terminación de línea

La terminación de línea sigue la convención del archivo de texto Unix[23]. Las líneas deben terminar con un único caracter de salto de línea. Los saltos de línea son representados como ordinales 10, o hexadecimales 0x0A.

Convención de nombres

- Clases

[22] *phpDocumentor, es un programa de generación automática de documentación para el lenguaje PHP.*

[23] *Unix, es un sistema operativo portable, multitarea y multiusuario.*

Los nombres de las clases sólo pueden contener caracteres alfanuméricos. Se adoptarán los espacios de nombres de PHP cuando estén disponibles y sea factible su uso en el desarrollo de aplicaciones.

- Nombres de archivos

Para todos los archivos, sólo caracteres alfanuméricos, guión bajo, y el caracter (“-”) son permitidos. Espacios están estrictamente prohibidos.

- Funciones y métodos

Los nombres de las funciones pueden contener sólo caracteres alfanuméricos. Guión bajo no está permitido. Los nombres de función deben ser tan detallados como sea posible para describir completamente su propósito y comportamiento.

Los métodos que permitan acceder a variables estáticas o de instancia siempre deben llevar el prefijo *set* o *get*. En la implementación de patrones de diseño, tales como los patrones *singleton* o *factory*, el nombre del método puede contener el nombre del patrón mientras sea para describir con mayor profundidad su comportamiento.

Para métodos en los objetos que son declarados con el modificador *private* o *protected*, el primer caracter del nombre del método debe ser un guión bajo (_). Esta es la única aplicación aceptable del guión bajo en el nombre de un método. Los métodos declarados como *public* nunca deben llevar guión bajo.

- Variables

Los nombres de las variables pueden contener sólo caracteres alfanuméricos. Guión bajo no está permitido.

Para instanciar variables que son declaradas con el modificador *private* o *protected*, el primer caracter del nombre de la variable debe ser un único guión bajo. Esta es la única aplicación aceptable del guión bajo en el nombre de una variable. Variables declaradas como *public* nunca deben comenzar con guión bajo.

Como con los nombres de las funciones los nombres de las variables deben comenzar con letra en minúscula.

Los nombres de variables deben contener un nivel de detalle para describir los datos que se almacenarán en esta.

Nombres de variables concisos como *$i* o *$n* se permiten pero para contextos de pequeños bucles. Si un bucle contiene más de veinte líneas de código, la variable índice debe contener un nombre más descriptivo.

- Constantes

Las constantes pueden contener caracteres alfanuméricos y guión bajo. Todas las letras usadas en el nombre de una constante deben ser en mayúsculas, mientras que todas las palabras en el nombre de una constante deben estar separadas por un guión bajo, ejemplo: *MODELS_MAKING_THIS.*

1.3.5 Java Script. Estándares de codificación

Java Script es un lenguaje de programación creado principalmente para hacer páginas web dinámicas. Este se basa en las tecnologías del lado del cliente, ya que es el navegador el que soporta la carga de procesamiento. Dentro de sus características se encuentra brindar interactividad en las páginas con el cliente. Es un lenguaje con muchas posibilidades dentro de la programación Web que permite la programación de pequeños scripts, pero también de programas grandes, orientados a objetos, con funciones y estructuras de datos complejas. Java Script le permite controlar al Desarrollador todo lo que ocurre en la página, cuando la está visualizando el cliente. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje script de programación del lado del cliente más utilizado. El mismo, al igual que otros lenguajes presenta un grupo de reglas o recomendaciones a seguir para mejorar la legibilidad, se trata de un conjunto de convenciones de codificación y las normas para su uso en la programación utilizando este lenguaje, inspiradas todas en el *Sun Document Code Conventions for the Java Programming Language*[24]. Estas normas han sido muy modificadas, por supuesto, ya que Java Script no es Java (27).

1.3.5.1 Principios generales de codificación en Java Script (28)

- La mayoría del código debe estar alojado en archivos Java Script externos. Estos deben ser incluidos al final del *tag BODY* para obtener el mejor rendimiento posible de la aplicación.

[24] *Documento de SUN sobre convenciones de codificación para la programación en lenguaje Java.*

- Todas las variables de tipo *Boolean* deben empezar con *is*.
- El nombre de las variables y las funciones deben ser asignados de manera lógica: Por ejemplo *popUpWindowForAd* en vez de *utilizarmyWindow*.
- Los bloques largos de código deben estar separados por comentarios para indicar las secciones del archivo.
- Las constantes y la configuración de las variables deben realizarse al inicio del archivo.
- Se debe poner empeño durante el proceso de desarrollo en la creación de funciones las cuales puedan ser generalizadas, tomar parámetros, y retornar valores.
- Comentar el código, esta actividad ayuda a la resolución de problemas presentados en la codificación o uso de las funciones Java Script.
- Minimizar el uso de variables globales, en el caso de especificar alguna, esta variable debe estar claramente identificada (Ver Figura # 1).

```
1 window.globalVar = { ... }
```

Figura # 1. Correcto uso de variables globales

- Debe incluirse un espacio en blanco luego de cada coma y dos puntos (y un punto y coma donde sea aplicable). No deben agregarse espacios inmediatamente entre los paréntesis y la sentencia que encierra.
- Adicionalmente, los corchetes deben aparecer siempre en la misma línea del argumento que les precede como se muestra en la Figura # 2:

Correcto

```
for (var i = 0, j = arr.length; i < j; i++) {
    // Do something.
}
```

Incorrecto

```
for ( var i = 0, j = arr.length; i < j; i++ )
{
// Do something.
}
```

Figura # 2. Correcto e incorrecto uso de corchetes

- Todas las variables en Java Script deben ser escritas completamente en minúsculas.

1.4 Principales normas de evaluación del código

Existen variadas instituciones dedicadas y capaces de establecer un grupo de parámetros a seguir como normas o estándares a la hora de realizar la evaluación de la calidad un producto de software. ISO/IEC presenta tres normas de cómo evaluar el software, estas normas son:

- ISO/IEC-9126.
- ISO/IEC-14598.
- ISO/IEC-25000

1.4.1 ISO/IEC-9126

ISO/IEC-9126 para la evaluación del software plantea un modelo de calidad en el que se recogen las investigaciones de multitud de modelos de calidad propuestos por los investigadores durante los últimos 30 años para la caracterización de la calidad del producto software. La misma establece para la evaluación del software tres vistas:

- Interna.

- Externa.
- En uso.

La interna es la encargada de establecer los lineamientos medibles a partir de las características intrínsecas del propio producto software (como el código fuente); la externa encargada de los lineamientos medibles a partir del comportamiento del producto software (como durante una prueba) y la vista en uso es la encargada de establecer dichos parámetros durante la utilización efectiva por parte del usuario (en un entorno de pre o producción). En la Figura # 3 se muestra dicha norma con sus correspondientes vistas.

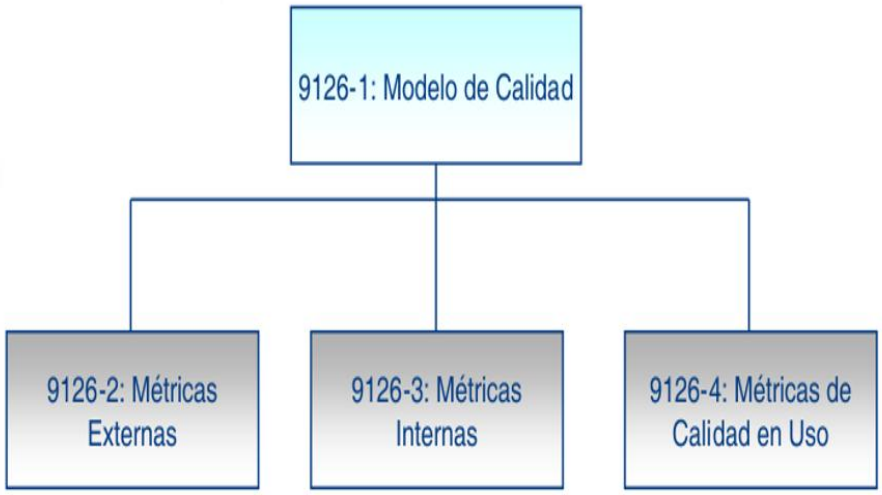

Figura # 3. Norma Internacional ISO/IEC 9126:2001

Específicamente, la vista interna se ocupa de las propiedades del software como el tamaño, la complejidad o la conformidad con las normas o estándares de codificación preestablecidos por el equipo de desarrollo. La externa se ocupa de analizar el comportamiento del software en producción y estudia atributos como el rendimiento de un software en una máquina determinada, el uso de memoria de un programa o el tiempo de funcionamiento entre fallos; mientras que la vista en uso mide la productividad y efectividad del usuario final al utilizar el software. Se observa que cada una de estas vistas está estrechamente relacionada ya que los valores de la vista interna afectan a los de la vista externa y los de la vista externa a los de la vista en uso (Ver Figura # 4**¡Error! No se encuentra el origen de la referencia.**).

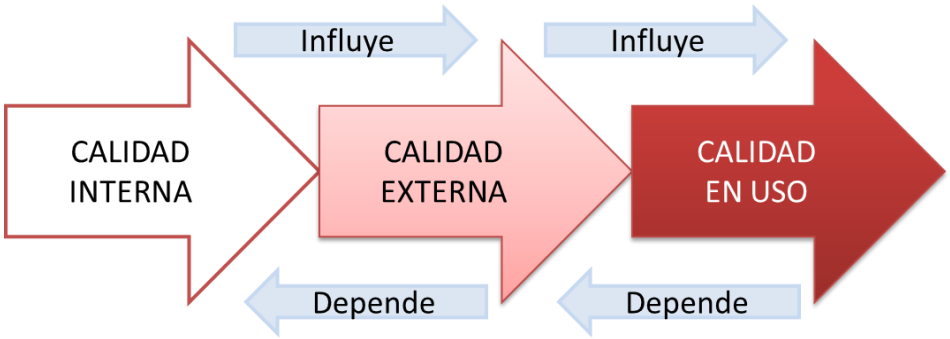

Figura # 4. Relación entre las vistas establecidas por ISO

Así por ejemplo un software con una alta complejidad, probado sobre una máquina con bajas prestaciones tendrá un rendimiento bajo que provocará que el usuario final tenga un rendimiento inferior al esperado independientemente de sus factores humanos.

Cada una de estas vistas está compuesta por características que se dividen por subcaracterísticas y estas a su vez están formadas por atributos como se muestra en la Figura # 5.

Figura # 5. Categorías ISO/IEC 9126:2001. Vista interna y externa

Entre las categorías anteriores se toma como referente con el fin de valorar la calidad interna del software dos de las seis categorías que sobresalen en el modelo ISO/IEC 9126, en este sentido, se estima que la base de los parámetros de evaluación del software debe estar integrada por la mantenibilidad y la eficiencia. Las mismas están compuestas por ocho atributos. A continuación se detallan las subcaracterísticas correspondientes a la mantenibilidad y a la eficiencia.

Mantenibilidad:

- Analizabilidad: Es la capacidad del producto software para serle diagnosticadas deficiencias o causas de los fallos en el software, o para identificar las partes que han de ser modificadas.
- Capacidad de cambio: Capacidad del producto de software que permite que una determinada modificación sea implementada.
- Estabilidad: Capacidad de evitar efectos inesperados tras realizar modificaciones en el software.
- Capacidad de pruebas: Capacidad para validar los cambios en el software.
- Adherencia a las normas: Capacidad del producto software para adherirse a normas o convenciones relacionadas con el mantenimiento.

Eficiencia:

- Comportamiento en el tiempo: Capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas.
- Utilización de recursos: Capacidad del producto software para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.
- Cumplimiento de la eficiencia: Capacidad del producto software para adherirse a normas o convenciones relacionadas con la eficiencia.

Estas subcaracterísticas proveen terminología consistente para la calidad de productos de software. Ellas también proveen un marco de trabajo para especificar los requerimientos de la calidad para productos de software, y para hacer análisis y evaluaciones entre capacidades de productos de software, compuestas por atributos que toman sus valores de un grupo de medidas efectuadas sobre el software ya sea ejecutable o no ejecutable. Estas mediciones están dadas por un grupo de métricas que pueden ser básicas, de agregación o derivadas (Ver epígrafe Taxonomías de métricas).

Pero si bien el modelo indica que estas subcaracterísticas a su vez se subdividen en atributos, no se especifica en la norma cuales son esos atributos, ya que se entiende que estos son entidades dependientes del producto software y variarán según varíe la naturaleza del software analizado: lenguaje, paradigma de programación o complejidad tecnológica.

1.4.2 ISO/IEC-14598

ISO/IEC-14598 ofrece una visión general, explica la relación entre su serie y el modelo de calidad de la ISO/IEC 9126, define los términos técnicos utilizados, contiene requisitos generales para la especificación y evaluación de la calidad del software, y clarifica los conceptos generales. Además, provee un marco de trabajo para evaluar la calidad de todos los tipos de productos de software y establece requisitos para métodos de medición y evaluación de los productos de software. La ISO/IEC 14598 está prevista para que se use conjuntamente con la ISO/IEC 9126-1 (29).

Es actualmente usada como base metodológica para la evaluación del producto de software. Es una serie de normas que dan los métodos para la medición y la evaluación de la calidad del producto de software a un nivel internacional.

Los servicios relacionados con la evaluación de software de productos son generalmente adaptados a las necesidades de los usuarios finales individuales o proveedores, en función de por qué se pidió la evaluación. Los servicios de evaluación de software incluyen:

- Definición de perfiles de calidad de referencia de software.
- Evaluación de acuerdo con los modelos de calidad predefinidos.
- Certificación de la calidad del software de acuerdo a los modelos de calidad y normas.
- Las comparaciones entre productos.
- La reingeniería del software.
- Servicio de Monitoreo de calidad del producto.

1.4.3 Familia de normas ISO/IEC-25000

En lo que se refiere a calidad del producto la norma ISO/IEC-25000 proporciona una guía para el uso de las nuevas series de estándares internacionales, llamados Requisitos y Evaluación de Calidad de

Productos de Software (SQuaRE). Constituyen una serie de normas basadas en la **ISO/IEC-9126** y en la **ISO/IEC-14598**, y su objetivo principal es guiar el desarrollo de los productos de software con la especificación y evaluación de requisitos de calidad. Establece criterios para la especificación de requisitos de calidad de productos software y su evaluación (11).

Al igual que la norma **ISO/IEC-9126**, este estándar define tres vistas diferenciadas en el estudio de la calidad de un producto las cuales son vista interna, externa y en uso donde la vista interna se ocupa de las propiedades del software como el tamaño o la complejidad que este puede tener.

La primera vista puede utilizarse desde las primeras fases del desarrollo, permitiendo detectar deficiencias en el software en edades muy tempranas del ciclo de vida del software.

La segunda, sin embargo, necesita que el producto software esté completo y se utilizará por tanto cuando pase a la parte de producción del producto, siendo muy dependiente de la máquina donde se ejecute.

Por último, la tercera vista que también estudia el producto software finalizado será dependiente del usuario y estará condicionada a los factores personales del mismo.

La serie ISO 25000 no establece los niveles de calidad deseables para cada proyecto, si bien se recomienda que los requisitos de calidad deban ser proporcionales a las necesidades de la aplicación y lo crítico que sea el correcto funcionamiento del sistema implementado.

ISO/IEC 25000 establece que la calidad del producto software está compuesta de características de calidad, las cuales a su vez se componen de subcaracterísticas. Así mismo, establece que las medidas de calidad de software indican las características y subcaracterísticas de calidad del producto de software.

El valor de estas medidas de calidad de software se obtiene por la aplicación de una función de medida a los elementos de medida de calidad.

Aunque las normas ISO/IEC 9126 y 25000 establecen cuales son las características de la calidad de un producto software y sus subcaracterísticas, no indica qué medidas de calidad indican una subcaracterísticas.

Fueron seleccionados de cada norma algunos conceptos importantes; se extrajo en el caso de la norma ISO 9126 un grupo de características que están presentes en dos de sus parámetros de la vista

interna, es el caso de la mantenibilidad y el rendimiento, que serán utilizados como base para la evaluación del software. De la norma ISO 14598 solo se seguirán algunos servicios que la misma ofrece para realizar la evaluación, es el caso de la comparación y reingeniería de productos, estos conceptos se utilizarán como una guía para crear un grupo de actividades propias de esta investigación que servirán para realizar la correcta evaluación y corrección del software. La norma 25000 no será utilizada directamente producto de que aunque se conoce que engloba las dos mencionadas anteriormente, de la misma no se tiene suficiente conocimiento como para ser empleada en esta investigación.

1.5 Herramientas de evaluación de la calidad de código

El desarrollo de software es algo muy complejo y la medida de su calidad real no es automatizable ni hay un convenio universal de en qué consiste, con lo cual hay que saber leer e interpretar el resultado que producen las herramientas encargadas de determinar el nivel de calidad que presenta cada software. Si estas herramientas están avisando de que nuestro código tiene problemas, es muy probable que tengan razón y debemos poner medidas correctoras.

1.5.1 Clasificación de herramientas de evaluación de la calidad del código

Las herramientas de evaluación de la calidad del código se dividen en dos categorías, según la forma en que realizan la evaluación del producto:

- Herramientas de análisis dinámico: Aquellas herramientas que utilizan el análisis del software ejecutando el código fuente de dicho software. Estas herramientas suelen requerir el uso de librerías especiales e incluso pueden necesitar recompilar el código del programa.

- Herramientas de análisis estático: Aquellas herramientas que llevan a cabo el análisis sin necesidad de ejecutar el software bajo estudio. Este tipo de análisis puede ser realizado sobre el código fuente o sobre el *byte code*[25].

1.5.1.1 Herramientas de análisis estático

[25] *Byte code, es un código intermedio más abstracto que el código máquina.*

Las herramientas de análisis estático permiten realizar una evaluación desde las primeras etapas, garantizando la calidad del software antes de su ejecución. A continuación se describen las herramientas que pertenecen a dicha categoría, ya que estas por su característica de no ejecución del código fuente son las indicadas para lograr los fines de esta investigación.

El análisis estático del código se podría definir como un conjunto de técnicas de análisis donde el programa bajo estudio no es ejecutado (al contrario que con el análisis dinámico), sino que es analizado. El término análisis estático se aplica normalmente cuando es realizado mediante herramientas automáticas y también es conocido como *Code Review*[26]. Por otro lado, cuando es realizado manualmente por personas, se conoce con el término de *Program Comprehension*[27].

La mayoría de las herramientas de análisis estático de código utilizan el concepto de regla como unidad atómica con la que se realiza dicho análisis. Una regla define un patrón que identificará el conjunto de caracteres o de líneas de caracteres que debe ser cumplido por el código que está analizando. Además de detectar defectos, las herramientas de análisis estático del código pueden utilizarse para obtener cálculos sobre la complejidad del código fuente, como por ejemplo la complejidad ciclomática de los métodos.

1.6 Herramientas que intervienen en el procedimiento

Las herramientas de análisis estático de código basan sus resultados y defectos detectados en la aplicación de un grupo de reglas como son complejidad ciclomática, tamaño de los métodos y tamaño de clases, o simplemente el correcto uso de los estándares de codificación. Todos estos parámetros de evaluación han sido diseñados por personas y en cada herramienta han sido implementados de manera distinta; por esta razón cada una de las herramientas de análisis estático de código puede devolver valores ligeramente distintos en una evaluación realizada sobre el mismo fragmento de código, por lo que la utilización de todas ellas proporciona un criterio más amplio a la hora de establecer una evaluación del nivel de calidad que presenta el código fuente, serán utilizadas todas las herramientas que se describen en este epígrafe.

[26] *Code Review, revisión del código.*

[27] *Program Comprehension, comprensión del programa.*

La selección de estas herramientas se realizó orientada principalmente a su uso libre basado en la licencia que estas presentan. No se tuvo en cuenta la flexibilidad que puedan tener estas con respecto al sistema operativo sobre el cual se ejecute, aunque sí se trató de que en su mayoría fueran ejecutadas sobre sistema operativo Linux.

1.6.1 PHP Code Sniffer

PHP Code Sniffer es un programa realmente práctico y fundamental para estandarizar la codificación del software, escrito en PHP, Java Script y CSS. Se trata de una herramienta muy conocida de desarrollo fundamental que asegura que su código se mantiene limpio y consistente. PHP Code Sniffer valida la sintaxis del código contra un estándar definido, este puede ser el de Zend, PEAR o alguno propio que definamos, lo cual es muy importante para evitar algunos errores comunes en la semántica hecha por los Desarrolladores ayudando así a evaluar y mejorar la legibilidad del código escrito. La misma podrá ser utilizada para medir el nivel de cumplimiento del estándar utilizado en el lugar que se aplique, en caso de que sea un estándar creado por el propio equipo de desarrollo esta herramienta brinda la posibilidad de crear un fichero PHP, en el cual se agreguen los parámetros que se desean evaluar seleccionándolos desde una base que la misma presenta.

1.6.2 PHP Depend

PHP Depend es un pequeño programa que realiza análisis de código estático en una determinada fuente. Cuando se habla de análisis de código estático se refiere a que PHP Depend primero toma el código fuente y lo convierte en una estructura interna de datos fácilmente procesables. Esta estructura de datos normalmente se llama un ***AST***[28] (árbol de sintaxis abstracta), que representa a las distintas declaraciones y los elementos utilizados en la base de la fuente analizada. Luego toma la generada AST y mide los valores de varias de las métricas de software. Cada uno de estos valores representa un aspecto de la calidad del software analizado, estos valores son mostrados al Evaluador que realiza el análisis mediante gráficas, debe tenerse en cuenta que estas métricas son indicativos de las posibles áreas problemáticas o sea que son sólo indicios de problemas. PHP Depend muestra los niveles de calidad de nuestro diseño en términos de extensibilidad, reusabilidad y mantenimiento mediante gráficas. Esta herramienta puede proporcionar mucha información útil acerca de un proyecto

[28] ***AST***, *Abstract Syntax Tree.*

de PHP especificado y ayuda a identificar que partes de una aplicación es en la que una refactorización debe ser aplicada.

1.6.3 PHPMD

PHPMD - PHP *Mess Detector*[29]. Se trata de un *spin-off*[30]del proyecto de PHP Depend[31] y pretende ser un equivalente para PHP del ***PMD***[32], herramienta bien conocida de Java. PHPMD es utilizada para el flujo de las métricas medida en bruto por PHP Depend. Esta herramienta detecta los problemas medidos por PHP Depend.

PHPMD necesita una determinada fuente con base de código PHP y nos brinda la funcionalidad de buscar varios problemas potenciales dentro de esa fuente. Estos problemas pueden ser:

Posibles errores de escritura, código no óptimo, expresiones considerablemente complicadas, parámetros no utilizados, métodos y propiedades.

Esta herramienta es muy configurable respecto a los problemas que se deseen buscar y entre las herramientas utilizadas para la evaluación es la que mayor cantidad de reglas presenta.

Esta herramienta será utilizada como apoyo a PHP Code Sniffer en cuanto a problemas como complejidad ciclomática, código no óptimo y nombre de variables, además podrá ser muy útil para detectar problemas de tamaño y complejidad de los métodos y clases, basada en las métricas de PHP Depend.

1.6.4 PHPCPD

PHPCPD es un detector de Copia / Pega (**CPD**[33]) para el código PHP. El objetivo de PHPCPD no es sustituir a instrumentos más sofisticados como PHPMD, sino más bien ofrecer una alternativa a ellos cuando sólo tiene que obtener una visión general rápida de código duplicado en un proyecto. PHPCPD

[29] *Mess Detector, detector de desorden.*

[30] *Spin-off: proyecto nacido como extensión de otro anterior.*

[31] *PHP Depend es un pequeño programa que realiza análisis de código estático en una determinada fuente.*

[32] ***PMD**, Program Mess Detector, programa detector de problemas en el código en Java.*

[33] ***CPD**, siglas en inglés de Copy / Paste Detector.*

muestra la URL del duplicado y además ofrece un valor del porciento de duplicaciones detectados respecto a las líneas de código existentes en el código evaluado. Esta forma de mostrar los resultados es muy conveniente ya que ese valor podrá ser utilizado para saber el nivel de calidad con respecto a este parámetro y además la cantidad de líneas que se podrán ahorrar si se efectúa una reutilización de este código.

1.6.5 JSLint

JSLint es una herramienta de análisis de código estático utilizada en el desarrollo de software para comprobar si el código fuente Java Script cumple con las normas de codificación. Es un *parser*[34] Java Script que verifica que el código es sintácticamente correcto, informa de las variables no declaradas o no utilizadas y otros errores comunes que son introducidos en el código fuente por los Desarrolladores. Escrita por Douglas Crockford[35] para validar el código Java Script, muestra advertencias, errores y mensajes de como reemplazar una línea de código por otro más robusto, es el caso de los operadores de comparación (en inglés: *comparison operators*) también llamados operadores condicionales, el caso concreto de **es igual** y de **no es igual**. Comprueba el código fuente Java Script sin tener que ejecutar el script o la apertura de la página web. Presenta una versión *online*[36] y otra ejecutable desde la propia PC sin necesidad de enviar tu código a la web. Además brinda la opción de realizar la evaluación a través de la consola o con una interfaz que muestra detalladamente cada uno de los problemas detectados. Esta herramienta brinda la opción de habilitar o deshabilitar opciones de revisión según se desee. Será utilizada con el fin de acercar la forma de escribir el código a las convenciones principales de codificación en este lenguaje.

1.6.6 Cloc

Cloc es una herramienta portable de fácil uso. Su principal función es contar las líneas de código de cada uno de los archivos a los que se le aplique. Esta herramienta es capaz de distinguir entre las líneas de comentario, líneas en blanco y las líneas de código. Puede generar sus resultados en

---

[34] *Parser, analizador.*

[35] *Douglas Crockford es un programador informático y empresario estadounidense, mejor conocido por su participación continúa en el desarrollo del lenguaje Java Script.*

[36] *Online, en línea.*

diferentes formatos de archivo como texto plano XML o YAML. Funciona en diferentes sistemas operativos como Linux y Mac OS. Es una herramienta muy configurable con respecto a la forma de mostrar los resultados. Funciona por consola y es aplicable para cualquier lenguaje de programación que exista. Esta herramienta se utilizará para recoger la cantidad de líneas que existen en cada uno de los lenguajes para posteriormente conocer cuantas líneas se están evaluando de cada lenguaje y según la cantidad de errores que se encuentre se puede obtener un por ciento con respecto a estos valores. Además, podrá mostrar si existió alguna reducción en cuanto a cantidad total de líneas de código de la aplicación, esto significaría una disminución en el tamaño de la aplicación.

Conclusiones Parciales

En el presente capítulo se analizaron diferentes conceptos fundamentales para un mejor entendimiento del trabajo, se definió el concepto de calidad de software por el cual se rige la investigación seleccionándose el establecido por ISO 9126 como el más apropiado para ser utilizado. De la misma manera se abordaron temas como métricas de software y métricas internas de software, esclareciendo su definición y mostrando ejemplos de dichas métricas. Además, se presentó un epígrafe dedicado a las normas o estándares donde se expone su definición conceptual, importancia y además una serie de estándares de codificación generales para todo lenguaje, identificándose una serie de reglas necesarias para una mejor legibilidad del código en general. Esto también se presentó más específico para el lenguaje PHP y Java Script, sirviendo estas reglas como una base para un posible rediseño de los estándares de codificación utilizados en los centros de donde provienen los futuros productos a evaluar y perfeccionar. Se mostraron además, las principales normas de evaluación de la calidad del software, centrándose principalmente en las dedicadas a la evaluación de la calidad interna del mismo. Se extrajeron de estas los parámetros que mejor describen la calidad de los archivos fuentes y pasos importantes que presentan para realizar la evaluación. Se describieron los tipos de herramientas que existen para evaluar el software y se realizó un estudio donde se seleccionó la herramienta PHPCPD para la búsqueda de código duplicado en PHP, JSLint para detectar errores de sintaxis en el código Java Script, Cloc para contar y eliminar líneas en blanco y comentarios en Java Script y PHP, PHPMD para detectar problemas de tamaño y complejidad, PHP Depend para evaluar la calidad en cuanto a tamaño y complejidad, y PHP Code Sniffer para comprobar el cumplimiento de estándares de codificación.

Capítulo 2: Propuesta del Procedimiento

Introducción

En el presente capítulo se define el procedimiento de evaluación y perfeccionamiento de los archivos de código fuente escritos en PHP y Java Script. En el mismo se define claramente cada uno de sus componentes fundamentales, como son los artefactos que en él se generan, se exponen cada uno de los pasos y actividades que se realizan para lograr la mejora de la calidad de codificación en los productos a los que se le aplique.

2.1. Procedimiento para la evaluación y perfeccionamiento de la calidad de los archivos fuentes

Un procedimiento es una serie común de instrucciones, operaciones y pasos definidos, que permiten realizar un trabajo de forma correcta, suelen realizarse siempre de la misma manera. (30)

El procedimiento ofrece pasos para llevar a cabo la evaluación y perfeccionamiento de la calidad de los archivos fuentes elaborados en lenguaje PHP y Java Script en los productos de software. Dicho procedimiento cuenta con elementos que organizan y especifican las actividades que se realizarán, definiendo el alcance, los objetivos de cada tareas, describiendo detalladamente cada una de ellas para hacer más fácil el trabajo y especificando los roles que intervienen y los artefactos generados durante todas estas tareas presentes en el procedimiento.

2.2. Objetivo

Este procedimiento tiene como objetivo la evaluación y perfeccionamiento de la calidad del código fuente escrito en PHP y Java Script; se define de forma detallada los pasos a seguir para ejecutar esta tarea utilizando herramientas destinadas al desarrollo y herramientas automatizadas para la evaluación.

2.3. Alcance

El procedimiento va dirigido a los productos de software de la UCI cuyos archivos fuentes se hallen en lenguaje PHP y Java Script, el mismo se centra en la calidad de los archivos fuentes buscando mejoras en cuanto a cumplimiento de estándares de codificación, mantenibilidad y en algunos casos rendimiento, o sea disminución en los tiempos de espera del cliente para descargar los scripts del servidor. En el procedimiento se describen los roles que intervendrán, las actividades a realizar y

artefactos que se generarán, describiendo minuciosamente el formato de cada uno de los artefactos necesarios para documentar la información generada y los cambios realizados durante su aplicación. Así es aplicable en proyectos de cualquier tamaño ya que este es independiente de esta característica y se aplica en las principales metodologías de desarrollo de software que se utilizan en la UCI (RUP, Scrum, open UP, eXtreme Programing).

2.4. Presentación del procedimiento por fases

El procedimiento cuenta con dos fases, la primera fase llamada fase de Estructuración, engloba la actividad general de planificación y tiene como objetivo fundamental planificar y definir el ambiente para llevar a cabo la evaluación y la posterior corrección. Esta fase es en la que se sustenta la fase de desarrollo, en ella se siguen una serie de políticas que posibilitarán la recogida de información necesaria para llevar a cabo la configuración del entorno de trabajo. El Planificador recoge mediante encuentros con otros trabajadores datos relevantes con el fin de realizar el Plan de Evaluación por el que se rigen las tareas a realizar.

La fase de Desarrollo es la fase que engloba las actividades generales de evaluación y corrección, es la encargada de realizar el análisis de todos los archivos de código fuente, con el fin de detectar e informar la mayor cantidad de errores basada en la información proporcionada por la fase de estructuración. Esta es la fase medidora y correctora que garantiza la calidad del código. Representan un papel importante en esta fase las herramientas de análisis de código estático y las herramientas de desarrollo. Surge de esta fase un código mejorado como resultado final del procedimiento.

2.4.1. Premisas para su aplicación

Para que el procedimiento se aplique con la calidad requerida, se deben tener en cuenta las siguientes premisas:

- Personal del proyecto comprometido con el procedimiento: Este procedimiento necesita de un alto compromiso del grupo de implicados en su utilización para que este pueda cumplir con su cometido.
- Personal con conocimientos para aplicar el procedimiento: El personal que aplicará el procedimiento debe estar estrechamente familiarizado con cada uno de los pasos a seguir y de las herramientas que se utilizan durante el desarrollo del mismo para lograr la mayor productividad.

2.4.2. Inserción de las fases dentro del proceso de desarrollo

El procedimiento está dividido en dos fases fundamentales, cada una de ellas con un objetivo, en el caso de la fase de Estructuración la cual tiene como objetivo fundamental la total elaboración de un Plan de Evaluación donde queden plasmados los plazos de evaluación y los hitos que van a ser evaluados por iteración, quedaría insertada entre las fases que anteceden a la fase de desarrollo o construcción de cada metodología. (Ver Figura # 6)

La fase de Desarrollo es la que basada en todos los datos recolectados anteriormente procede a realizar las tareas necesarias para lograr el objetivo final del procedimiento. Esta fase necesita para su ejecución el código fuente generado por el equipo de desarrollo hasta el momento en que se decide aplicar el procedimiento, es por esto que esta fase queda insertada dentro del proceso de desarrollo en la fase de construcción o desarrollo. (Ver Figura # 6)

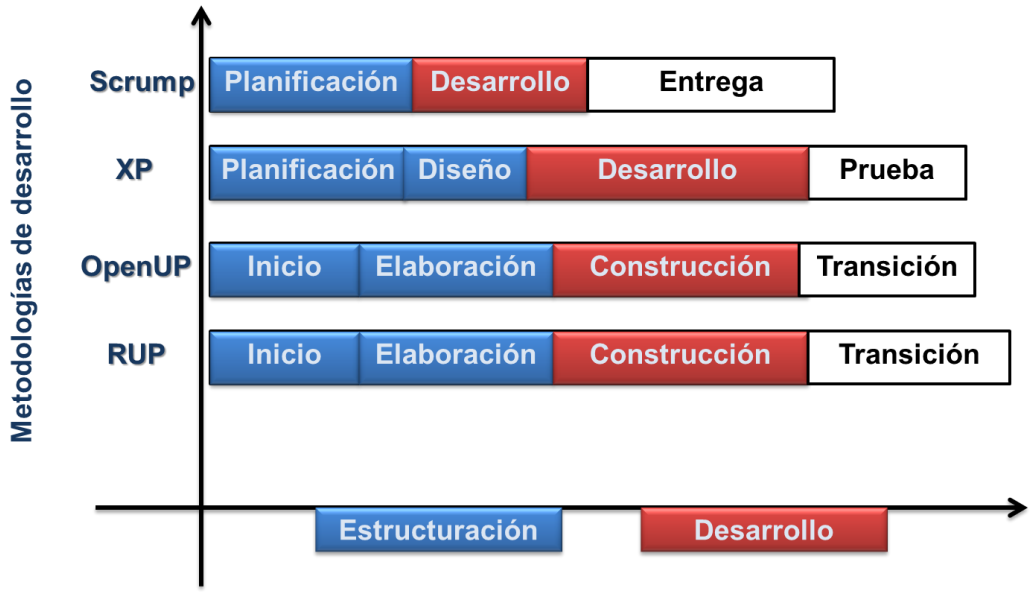

Figura # 6. Inserción de fases dentro de procesos de desarrollo

2.4.3. Secuencia de Actividades Generales

Dentro de cada fase del procedimiento se ejecutan actividades que engloban individualmente un conjunto de pasos a realizar para lograr el objetivo de cada una de estas fases. En la fase de Estructuración se realiza la planificación total del procedimiento y en la fase de Desarrollo se realiza

una evaluación inicial y de ser necesario se corrigen los defectos para realizar después una reevaluación. El propósito de esta reevaluación es comparar los resultados obtenidos después de culminada la actividad de Corrección con los obtenidos en la evaluación previa realizada para asegurar una efectiva corrección de defectos, definiéndose de esta manera un proceso recurrente como se muestra en la Figura # 7.

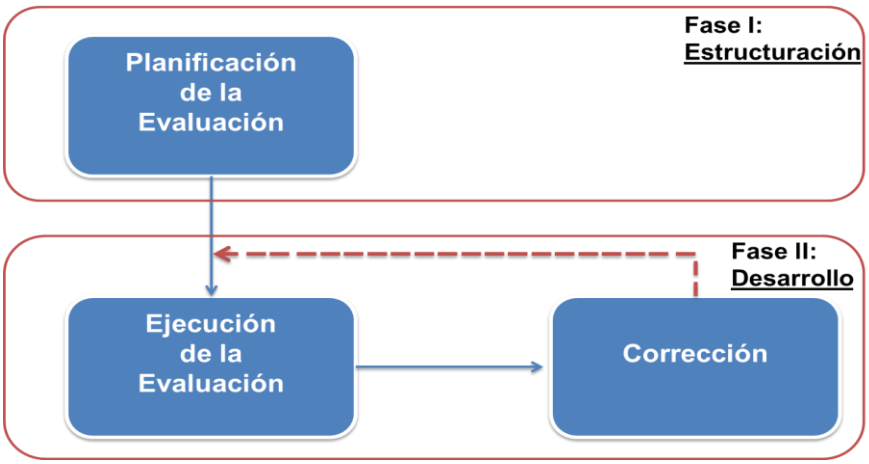

Figura # 7. Secuencia de Actividades Generales

2.5. Definición y descripción de tareas por actividades

2.5.1. Planificación de la Evaluación

La actividad de Planificación de la Evaluación enmarcada dentro de la fase de Estructuración se realizará posterior a la discusión y definición del cronograma del proyecto, es la encargada de recoger la información necesaria para consumar principalmente el Plan de Evaluación (Ver Expediente de proyecto.). Esta tarea se sustenta sobre artefactos que varían su nombre según la metodología utilizada, por esta razón el Planificador, el Arquitecto y el Evaluador deben tener bien claro cuales son los datos reales que debe buscar, para esto se debe efectuar un encuentro previo al comienzo del procedimiento donde quedarán delegadas cada una de las tareas por roles, a partir de este momento cada uno de los trabajadores que intervienen en esta actividad tendrá un máximo de dos días para obtener la información necesaria. La actividad comienza con la recopilación de datos como el Estándar de Codificación a utilizar, el registro de los Desarrolladores que forman parte del equipo de desarrollo, y la especificación de requisitos funcionales establecidos hasta ese momento en el cronograma de proyecto. En esta actividad, la recogida de datos sobre los Desarrolladores se produce con el fin de seleccionar dos de ellos que formarán parte del equipo de corrección. Además, se presenta especial

atención a la configuración del entorno de trabajo definiendo las herramientas de evaluación y tomando las herramientas de desarrollo ya definidas con las que se va a trabajar durante la fase de Desarrollo.

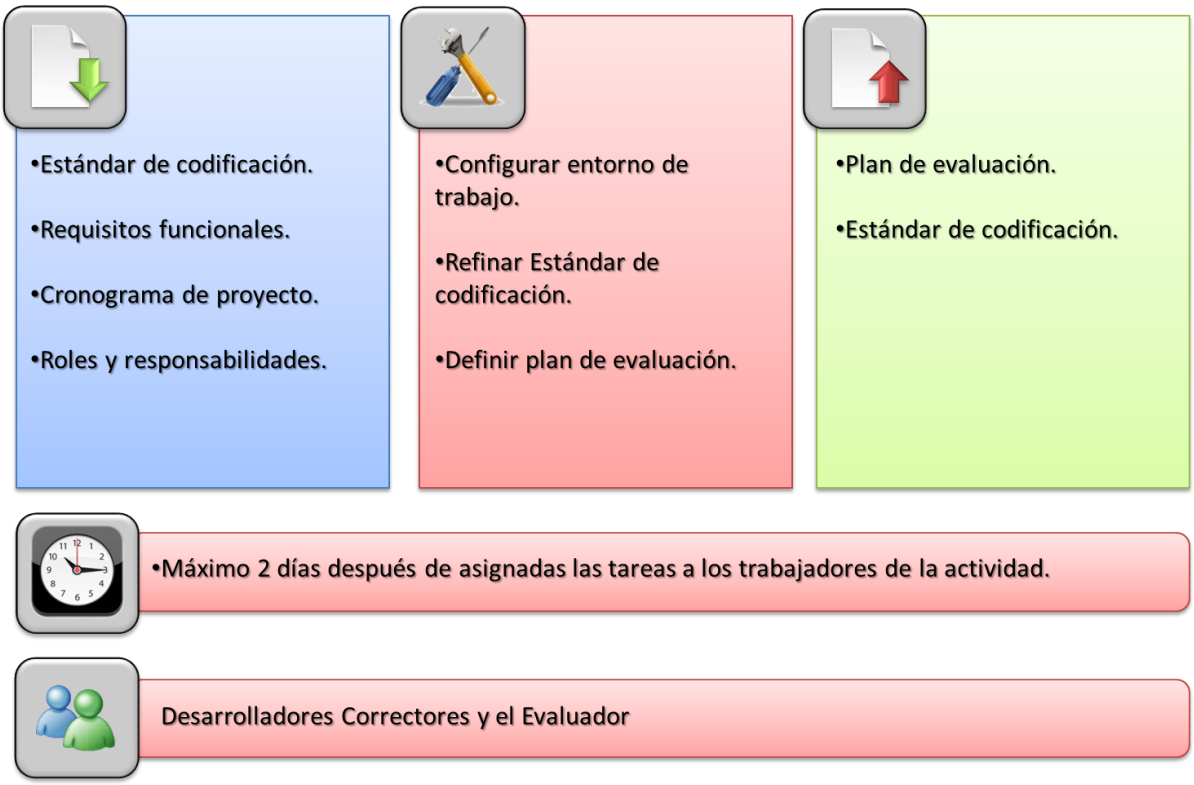

Figura # 8. Vista de la Actividad de Planificación. Definición de tareas.

Paso 1. Refinar Estándar de codificación

Durante la ejecución de este paso se define un Estándar de codificación y se reúnen los Desarrolladores y el Arquitecto en busca de mejoras de legibilidad, robustez y comprensión. En este paso el Arquitecto es el que decide los cambios que se le realizarán al estándar de codificación.

Paso 2. Configuración del entorno de trabajo

En este paso se especifican las herramientas que intervendrán en la evaluación y se toman además las herramientas de corrección de los defectos que no son más que las propias herramientas de desarrollo utilizadas hasta el momento, para esto el Evaluador debe conocer cuales son estas herramientas de desarrollo que están presentes en la implementación del producto, esta información está plasmada en el documento de Ambiente de Desarrollo y debe quedar especificada en el Plan de Evaluación (Ver Expediente de Proyecto) para que en el momento en que se necesite saber esta

información no sea necesario solicitar el documento de Ambiente de Desarrollo. Además, se selecciona el personal que estará implicado en las tareas de corrección de defectos.

Para seleccionar a los Desarrolladores-Correctores, se efectúa una selección por competencia basados en un grupo de factores a los cuales se les otorga una calificación entre 1 y 5, donde 1 representa malo, 2 regular, 3 bueno, 4 muy bueno y 5 excelente. Luego de aplicada la calificación se suman los valores y el resultado se divide entre la cantidad de factores evaluados, para lograr así seleccionar los dos mejores del total de calificaciones; en caso de ser evaluados más de dos Desarrolladores con la misma puntuación se realiza una selección al azar.

Factores y descripción:

Desempeño	
Responsabilidad	Grado de compromiso que asume para el cumplimiento de las metas. Grado de tranquilidad que le genera a su superior. (Por ejemplo entregando el código probado)
Exactitud y calidad de trabajo	Coherencia entre el trabajo solicitado y el efectivamente realizado. Grado de perfeccionismo que demuestra en el trabajo. El trabajo realizado cumple con lo requerido y además es de buena calidad. (El código funciona correctamente y está optimizado.)
Orden y claridad del trabajo	Sus desarrollos pueden ser abordados con facilidad por otras personas. (Los nombres de las variables son claros, el código es ordenado y legible, de ser necesario modificar su código es posible hacerlo)
Capacidad de realización	Autonomía, pragmatismo. Posibilidad de llegar a la última instancia de una tarea superando los obstáculos. Capacidad de interactuar con otros en búsqueda de alcanzar las metas.
Comprensión de situaciones	Capacidad de entender conceptos y situaciones rápidamente. Capacidad de modelar elementos complejos, tanto técnicos, funcionales o conceptuales.
Sentido Común	Capacidad para ubicarse en las situaciones de manera coherente. Capacidad de elegir alternativas convenientes con visión estratégica a futuro y siendo realista.

Cumplimiento de los procedimientos existentes	Grado de cumplimiento de las normas, procedimientos y políticas existentes.
Grado de Conocimiento Técnico	Conocimiento de las distintas herramientas necesarias para desarrollar su labor.
Factor Humano – Actitud	
Actitud hacia el Centro de trabajo	Capacidad de defender los intereses del centro y adherirse a sus lineamientos. Disponibilidad para extender el horario de trabajo ante una necesidad puntual.
Disposición	Se muestra predispuesto hacia la tarea. Manifiesta una actitud positiva frente a los diferentes requerimientos. Entusiasmo y Motivación.
Puntualidad	Puntualidad en horario laboral y reuniones.
Habilidades	
Creatividad	Ofrece alternativas innovadoras para solucionar problemas. Capacidad de vincular distintos conocimientos para una nueva aplicación de los mismos.
Adaptabilidad (temas, grupos, funciones)	Capacidad para desempeñarse con facilidad en situaciones que no le son naturales. Adaptabilidad a situaciones adversas.
Respuesta bajo presión	Capacidad de mantener la calma y transmitirla a sus compañeros. Capacidad de tomar decisiones correctas bajo presión. Capacidad de sacar provecho de situaciones adversas. Capacidad de realización en estos casos.
Idioma Inglés	Posee facilidad para comprender textos escritos en idioma inglés, sus conocimientos son: ninguno, escasos, elementales medios o superiores.

Paso 3. Definir Plan de Evaluaciones

Para la ejecución de este paso es necesario que el Planificador conozca los requisitos funcionales del producto y el momento exacto en que culminará su elaboración en cada una de las iteraciones, estos datos se encuentran en el cronograma del proyecto. Con estos datos el Planificador establece un cronograma de evaluación y corrección que estará dentro del Plan de Evaluación (Ver Expediente de Proyecto) el cual contiene los requisitos que serán evaluados al final de cada iteración de la implementación, especificando el momento exacto en que se le efectuará a cada uno la evaluación y corrección. Para realizar este documento se debe tener el estándar de codificación propuesto por el Arquitecto y el personal encargado de la corrección seleccionado por el Evaluador, para esto sesionarán el Arquitecto, el Evaluador y el Planificador y habrá un intercambio de información para que de esta forma quede confeccionado el Plan de Evaluación (Ver Expediente de Proyecto). En este documento además quedan registradas las personas que estarán implicadas dentro del procedimiento y las herramientas que se utilizarán tanto para la evaluación como para la corrección.

Para una realización ordenada y efectiva de la evaluación que se debe especificar los archivos que se afectarán con el desarrollo de cada requisito. Esta información quedará plasmada en la sección Archivos Afectados por Requisitos del Plan de Evaluación, la sección se actualizará en conjunto con el Plan de Pruebas que se realiza en el inicio de la fase de desarrollo o construcción y el encargado de realizar la actualización es el Analista.

Secuencia de tareas

Figura # 9. Secuencia normal de tareas. Actividad de Planificación.

Roles que intervienen en esta actividad

- Planificador: Redacta el Plan de Evaluación, documenta el cronograma de realización de las actividades de evaluación y corrección.
- Arquitecto: Encargado de realizar la definición o el refinamiento del Estándar de Codificación, es el principal responsable de los cambios que se le podrán realizar basado en evidencias dadas por los Desarrolladores.
- Evaluador: Realiza la selección de los Desarrolladores-Correctores que intervendrán en las labores de corrección y define las herramientas que intervendrán en el procedimiento.

2.5.2. Ejecución de la Evaluación

La actividad de ejecución de la evaluación está enmarcada en la fase de Desarrollo, es una tarea iterativa e incremental ya que se ejecuta una vez por cada iteración de construcción y se va agregando código haciendo que la cantidad de código que se evalúa aumente. Dispone para cada vez que se ejecute de dos a seis horas, en dependencia de la cantidad de código que se evalúe y de la cantidad de errores que se detecten. Comienza en el momento del montaje o instalación de las herramientas establecidas necesarias para llevar a cabo la evaluación, además se dejan instaladas las herramientas que se utilizarán para realizar las correcciones. Posteriormente se dispone a configurar cada una de las herramientas de evaluación, esta configuración no es más que definir los parámetros por los que se va a guiar dicha herramienta para realizar la evaluación del código, para esto el Evaluador debe tener conocimiento de estas herramientas. Los trabajadores de esta fase guiados por el plan previsto en la fase de Estructuración comienzan a realizar las evaluaciones identificando defectos en los archivos de código fuente.

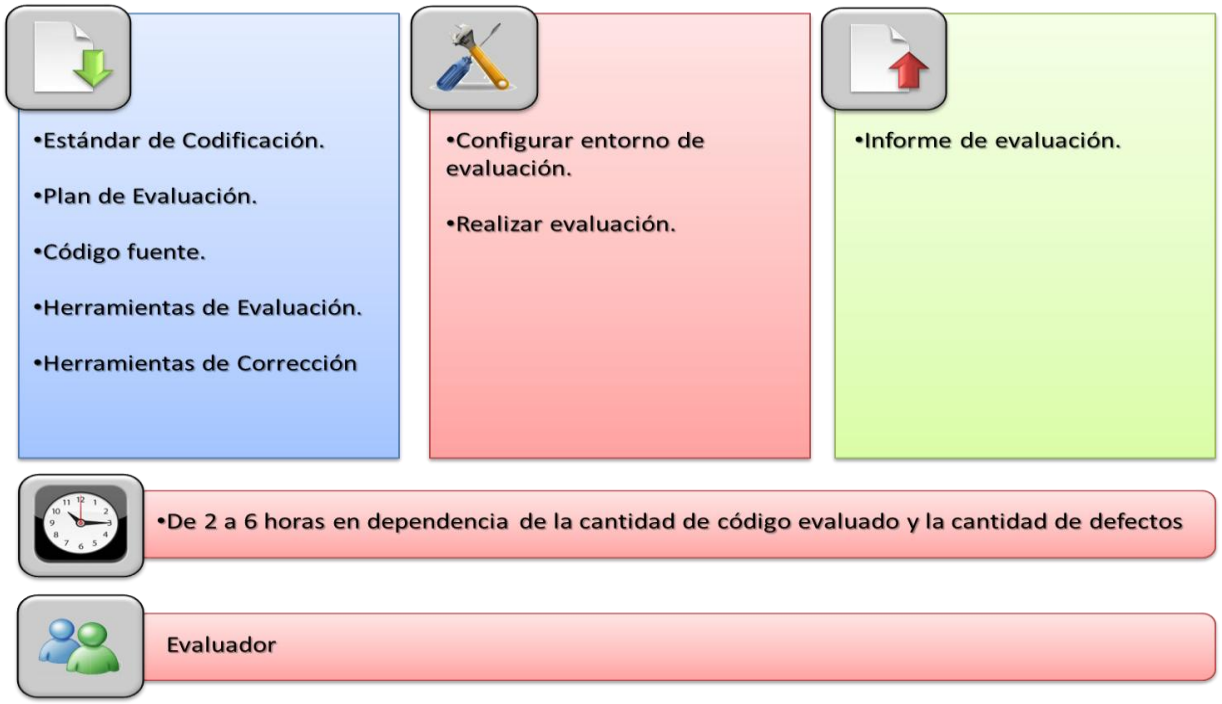

Figura # 10. Vista de la Actividad de Evaluación. Definición de tareas.

Paso 1. Configurar entorno de evaluación

Este paso tiene como entrada las herramientas de evaluación y corrección, el Evaluador crea su propio entorno de trabajo montando cada una de las herramientas para realizar la evaluación, también debe dejar instaladas las herramientas de corrección en cada una de las PC de los Desarrolladores-Correctores y además de colocar en el escritorio de la PC la herramienta Cloc que podrá ser en algunas ocasiones utilizada por estos trabajadores. Las herramientas de evaluación y corrección deben haber quedado especificadas en el Plan de Evaluación (Ver Expediente de Proyecto) en la fase anterior.

El Evaluador deberá configurar las herramientas de evaluación que buscan los problemas de incumplimiento de estándar de codificación guiado por el estándar de codificación que está presente en el Plan de Evaluación. Las herramientas que se encargan de medir este factor son JSLint para Java Script y PHP Code Sniffer para PHP.

Configuración de herramientas:

JSLint

Para configurar JSLint el Evaluador en caso de utilizar la versión con interfaz por consola debe leer el archivo README.TXT que está presente dentro de la carpeta de la propia herramienta donde se especifica la manera de activar y desactivar sus funciones. La activación de las funciones depende del estándar y están todas activadas por defectos.

PHP Code Sniffer

La configuración de PHP Code Sniffer está dada por el estándar de codificación. Esta herramienta presenta configuraciones preestablecidas de los estándares más utilizados aunque en caso de presentar un estándar con características específicas se deberá leer la guía de realización de esta tarea. (Ver Expediente de Proyecto)

PHPMD

En el caso de PHPMD también se le pueden activar y desactivar todas sus funciones pero se recomienda que se utilicen todas activadas con excepción de las reglas de diseño que no se tomarán en cuenta. Para crear un grupo de reglas se debe seguir la guía de creación. (Ver Expediente de Proyecto)

Las demás herramientas de evaluación no necesitan de ninguna configuración, aunque si se les puede definir la forma de mostrar los resultados.

Cloc

Cloc mostrará de manera automática una tabla que tendrá la información necesaria para documentar la sección de descripción general del código a evaluar en el Informe de Evaluación.

PHP Depend

A la herramienta PHP Depend se le definirá como forma de mostrar los resultados la gráfica pyramid.svg de la cual se extraerá el resultado de la métrica en cuanto a tamaño y complejidad para PHP.

PHPCPD

Esta herramienta muestra el porciento de código duplicado y el lugar exacto donde se encuentra el duplicado en el software evaluado, esta información puede ser utilizada por el Evaluador para documentar el Informe de Evaluación.

Paso 2. Realizar evaluación

Esta tarea tiene como entradas el Código Fuente y el Plan de Evaluación (Ver Expediente de Proyecto) por el cual se guía el Evaluador para realizar la evaluación. En esta tarea es donde se toma el código fuente de las funcionalidades definidas y se le realiza la evaluación, surgiendo posteriormente un Informe de Evaluación (Ver Expediente de Proyecto) con los defectos detectados.

Estrategia a seguir:

Primeramente se debe aplicar la herramienta Cloc con el fin de conocer la cantidad de líneas ya sean de código, comentario o en blanco; esto se realizará para todas las funcionalidades que se evaluarán en conjunto, los valores extraídos se registran en el Informe de Evaluación en la sección de Descripción General de Código a Evaluar y serán de gran importancia para llegar a conocer el nivel de calidad que presenta el código basado en la métrica (extraída del sitio calisoft.uci.cu):

$$\frac{\sum_{i=1}^{N} Total.fallas.detectadas(i)}{Total.KLOC inspeccionadas}$$

Figura # 11 Métrica de evaluación

Los errores deben ser clasificados según la herramienta que lo detectó, en el caso de los errores detectados por PHP Code Sniffer y JSLint serán calificados en errores de tipo estándar de codificación y se registrarán con sus siglas (EC), en el caso de PHPCPD se clasificarán en errores de tipo código duplicado y se representarán como (CD). PHPMD por ser la herramienta que mayor cantidad de taxonomías de errores detecta, el registro de los mismos será general para todos siendo representados con las siglas (TC) que representarán los errores de tamaño y complejidad con excepción de las detecciones de código duplicado que seguirán representándose con el acrónimo (CD). El Evaluador debe cerciorarse de que los errores queden documentados correctamente, ordenados por línea de código y por archivo ya que este documento es por el cual se guiarán los Desarrolladores-Correctores para ejecutar las labores de corrección.

Una vez aplicadas las herramientas, se debe realizar una inspección de cada uno de los errores en busca de errores iguales detectados por diferentes herramientas. Los duplicados deben ser suprimidos de inmediato para no alterar el resultado de la métrica. En el caso de que los duplicados de errores de código PHP contengan un error detectado por la herramienta PHPMD se eliminará siempre el error detectado por PHPMD.

Para la utilización de la métrica se deben seleccionar la clasificación de la falla que se vaya a medir y el lenguaje al que se le aplica, a ese tipo de falla se le cuenta la cantidad de ocurrencias en el Informe de Evaluación y es ese número el que se divide entre la cantidad de líneas de código analizadas según el lenguaje; la cantidad de líneas de código analizadas permanece constante para cada tipo de falla por lenguaje. El resultado de la métrica es el grado de ocurrencia de dicho error.

El resultado de la métrica se evalúa como:

Bajo: resultado <= 0.15 (La evaluación puede o no realizarse para esta clasificación)

Medio: 0.15 >= resultado <= 0.50

Alto: 0.50 >= resultado <= 1

Observaciones:

Los resultados de la métrica nos muestran un nivel de cómo se encuentra nuestro código, es necesario que en lo referente a los errores de código duplicado sea cual sea el resultado de la métrica se analice la posibilidad de reutilizar la porción de código duplicado que se detectó.

PHP Depend se utilizará para la evaluación de los problemas de Tamaño y Complejidad, por lo que los resultados de la métrica aunque se extraen igual para cada taxonomía de errores no son los utilizados para cualificar el código. Esta herramienta genera una gráfica en forma de pirámide que define en baja, media o alta la calidad de varios parámetros, estos parámetros se encuentran divididos en tres listas que son:

- Tamaño y complejidad.
- Acoplamiento.
- Herencia.

Para medir el código solo se tendrán en cuenta los parámetros pertenecientes a la lista Tamaño y complejidad que contiene los indicadores:

- NOP: Número de paquetes o de carpetas.
- NOC: Número de clases.
- NOM: Número de métodos por clases.
- LOC: Número de líneas ejecutables.
- CYCLO: Complejidad ciclomática de métodos.

Estos indicadores están representados en los 5 escalones más bajos del lado izquierdo de la pirámide como se muestra en la Figura # 12.

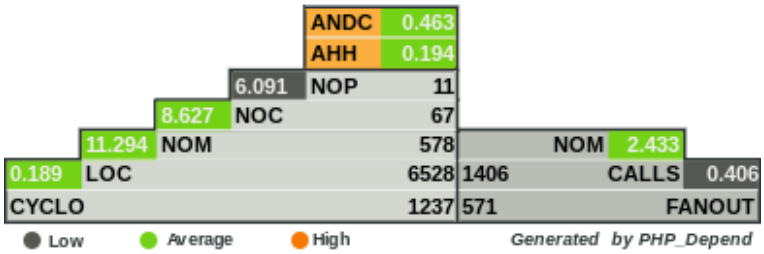

Figura # 12. Gráfica de pirámide de PHP Depend

Para obtener una cualificación respecto al tamaño y complejidad se realizará un promedio entre los resultados que muestra PHP Depend (alto, medio, bajo).

Secuencia de tareas

Figura # 13. Secuencia normal de tareas. Actividad de Evaluación

Roles que intervienen en esta actividad

- Evaluador: Es el revisor técnico encargado tanto de evaluar el código fuente de entrada, como de crear su entorno de trabajo, instalando y configurando cada una de las herramientas que utiliza.

2.5.3. Corrección

La actividad de corrección se encuentra dentro de la fase de desarrollo, comienza una vez culminada la actividad de Evaluación debido a que la misma depende del Informe de Evaluación (Ver Expediente

de Proyecto) como artefacto de salida. En el Informe de Evaluación debió haber quedado plasmado cada uno de los defectos detectados y es al comienzo de esta actividad donde se identifican los elementos a los cuales se les pretende realizar la refactorización basado en el resultado de la métrica. Para la realización de los cambios más complejos se realiza una Solicitud de Cambio. Esta actividad tiene como artefacto de entrada además el Código Fuente del producto. La actividad tiene como artefacto de salida un informe donde quedarán plasmados los defectos de mayor complejidad que fueron corregidos y la corrección que se le efectuó llamado Informe de Resultados. (Ver Expediente de Proyecto)

Figura # 14. Vista de la Actividad de Corrección. Definición de tareas

Paso 1. Identificar defectos a corregir

Este paso es realizado por los Desarrolladores iniciales y por el Evaluador, tiene como entrada el Informe de Evaluación (Ver Expediente de Proyecto) y como salida un grupo de posibles fuentes a corregir. El Informe de Evaluación será utilizado para poder conciliar los defectos detectados que se van a corregir, ya que existe la posibilidad de que algunos de los defectos detectados por las herramientas no puedan o deban ser corregidos producto del impacto que puedan tener en el diseño, o simplemente porque por decisión de los propios Desarrolladores los errores no representan un problema para la legibilidad ni la funcionalidad del código.

Paso 2. Realizar Solicitud de Cambio

Esta tarea genera una planilla de Solicitud de Cambio (Ver Expediente de Proyecto) la cual es documentada con los defectos más complejos que se planean corregir o con los errores donde la decisión de corregir es dudosa. Solo se documentarán los errores de código duplicado y de tamaño y complejidad ya que los errores de estándar de codificación se consideran de baja complejidad y el Desarrollador-Corrector debe tener autonomía para solucionarlos sin necesidad de solicitar permisos; aunque cabe la posibilidad de que para dar solución a alguno de estos errores sea necesaria una autorización. La Solicitud de Cambio original que está presente en el proceso de desarrollo fue modificada y adaptada para su uso especial en este procedimiento de manera que facilite el trabajo; fueron suprimidas algunas de las secciones que este exhibía debido a la cantidad de errores que pudieran ser detectados y la complejidad que estos presentan. Esta solicitud es evaluada por un comité de evaluación; el comité está formado por un grupo de trabajo formado por el Arquitecto, los Desarrolladores iniciales de las funcionalidades y el Analista, que deciden si se aprueba o no cada cambio, basado en las implicaciones del cambio y el impacto que puedan tener en el diseño, para esto el Arquitecto, los Desarrolladores y el Analista sesionan con el fin de analizar cada uno de estos defectos, el tiempo de esta tarea está dado por la cantidad de errores que aparezcan en la solicitud.

Paso 3. Realizar corrección

Este paso tiene como entrada el Código Fuente, se ejecuta para todos los errores de estándar de codificación y para cada error aprobado en la Solicitud de Cambio, los Desarrolladores-Correctores comienzan a corregir cada uno de los defectos del final de los archivos hacia adelante para no modificar las líneas de código donde se encuentran los siguientes errores, utilizando para ello las herramientas de desarrollo especificadas en el Plan de Evaluación (Ver Expediente de Proyecto) que fueron instaladas por el Evaluador en la actividad anterior. A medida que se va terminando la corrección de los descritos en la Solicitud de Cambio se documenta según la acción que se realizó sobre él en el Informe de Resultados de Corrección (Ver Expediente de Proyecto) que será entregado al Evaluador. El resultado es archivado en el Historial de Evaluaciones que se crea en el momento de finalizada la primera corrección efectuada al producto.

Observaciones:

Después de realizados cada uno de estos pasos se prosigue a realizar la reevaluación que no es más que la aplicación del Paso 2. **Realizar evaluación** de la actividad **Ejecución de la Evaluación**, para comparar los resultados obtenidos previamente con los obtenidos en esta nueva evaluación. Es en ese

momento en el que se compara cada uno de los resultados para asegurar que fue fructífera la aplicación del procedimiento.

En caso que se requiera disminuir el tamaño de los archivos fuentes se tendrá que eliminar todo el comentario y las líneas en blanco del producto, se utilizará para esto la herramienta Cloc la cual brinda esa funcionalidad; se debe tener en cuenta que al eliminar algunos comentarios en el código se limita la legibilidad del mismo y podría complicar las labores de mantenimiento futuro del producto. La decisión de realizar esta acción deberá ser tomada en función de las necesidades reales que existan, el encargado de tomar la decisión es el Arquitecto.

Secuencia de tareas

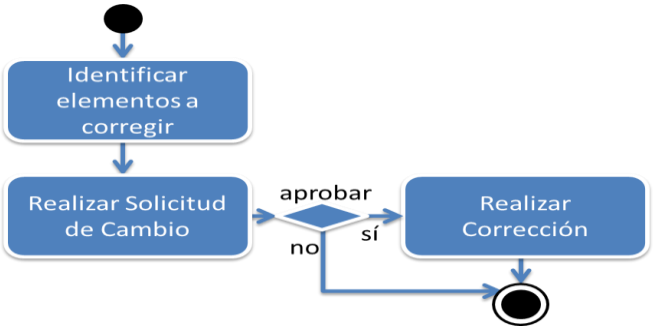

Figura # 15. Secuencia normal de tareas. Actividad de Corrección

Roles que intervienen en esta actividad

- Desarrolladores-Correctores: Encargados de realizar junto al Evaluador la identificación de defectos, la Solicitud de Cambio y posteriormente la corrección de los errores.
- Evaluador: Revisor técnico que en esta fase solo cumple la labor de realizar la identificación de las fuentes a corregir junto a los Desarrolladores-Correctores.

2.6. Descripción de Roles

- Planificador: Gestiona la planificación, la documenta, planifica en el momento en que se van a aplicar las evaluaciones y las correcciones, las fechas de entrega de los resultados alcanzados, así como los recursos utilizados en la aplicación del procedimiento. Vela por que se acoja el cronograma, debe ser una persona con capacidad para delegar tareas, de coordinación, liderazgo y dominar las técnicas de trabajo en equipo. Se propone que sea la misma persona que realizó las tareas de planificación en los inicios del proceso de desarrollo.

- Desarrolladores-Correctores: Es el trabajador que efectúa la actividad de corrección de los errores. Los mismos forman parte del conjunto de Desarrolladores que actúan en el proceso de desarrollo del producto de software, para su selección se emplea el método de competencia mostrado anteriormente. Se subordina al Evaluador, al cual tiene que entregarle la información generada luego de aplicado el procedimiento para que este actualice el Historial de Evaluaciones.
- Evaluador: Es el revisor técnico encargado de realizar directamente la evaluación, transita por las dos fases del procedimiento cumpliendo tareas como la selección del personal implicado en las labores de corrección y está presente en la identificación de las fuentes a corregir.

2.7. Descripción de Artefactos

Nombre del artefacto	Descripción	Responsable
Historial de Evaluaciones	Es un acumulado histórico de información que agrupa los informes de evaluación, contiene todos los datos y artefactos recopilados en aplicaciones previas de este procedimiento.	Evaluador
Estándar de codificación	Artefacto en el cual aparecen las normas y reglas de codificación establecidas por el equipo de trabajo. Esta información se halla en el documento de arquitectura en el caso de RUP, en el caso de XP el artefacto que la contiene es el Documento de Concepción del Sistema.	Arquitecto apoyado por los Desarrolladores del sistema
Plan de Evaluación	Artefacto que definen los momentos en los que se van a realizar las evaluaciones y que requisitos o funcionalidades se ven implicadas dentro de cada una de ellas, este documento recoge además las herramientas que serán utilizadas para evaluar y corregir, el estándar de codificación, lugar donde se realizará, recursos materiales y los implicados en cada una de las tareas, cada uno definido por un rol de acuerdo con la función que realiza.	Planificador

Nombre del artefacto	Descripción	Responsable
Informe de Evaluación	Es el artefacto que recoge el resultado de la evaluación realizada por cada archivo; contiene las funcionalidades que se evaluaron, la línea exacta donde se encontró el error, la descripción del error, las taxonomías de estos y la herramienta que lo detectó. Presenta una sección que recoge información importante acerca del código evaluado como el total de líneas evaluadas, en blanco, de comentario y ejecutables. En este artefacto se registran los valores de la métrica utilizando toda la información que en él se refleja.	Evaluador
Informe de Resultados de Corrección	Artefacto que contiene los errores más complejos que se corrigieron por cada archivo. Este documento es generado después de realizada la corrección de defectos con el fin de documentar todos los cambios que se hayan efectuado.	Evaluador y Desarrolladores-Correctores
Solicitud de Cambio	Artefacto modificado a partir de la Solicitud de Cambio que está presente en el proceso de desarrollo, presenta los campos Número del error, Archivo correspondiente, Descripción del error, Pedido de cambio y la respuesta dada por el comité sobre el cambio.	

Conclusiones Parciales

En el capítulo se presentó un procedimiento para realizar la evaluación y perfeccionamiento de los archivos de código fuente escritos en PHP y Java Script. Se especificaron los pasos a seguir para lograr evaluar y mejorar la calidad del código. Describiendo detalladamente cada tarea que se ejecuta con sus respectivos responsables y con el uso de artefactos diseñados específicamente para ser utilizados en este procedimiento.

Capítulo 3: Aplicación y Validación del Procedimiento

Introducción

En el presente capítulo se muestra una descripción de la aplicación del procedimiento mostrado en el capítulo dos de este trabajo a un producto de software. En el mismo se describe como se realizaron los pasos y tareas necesarias, haciendo mayor énfasis en las actividades más significativas para su correcta aplicación. Se muestra como resultado y evidencia de la aplicación del procedimiento la información resultante, que debidamente quedó documentada en cada uno de los artefactos, además, una medida de las mejoras que trajo consigo la aplicación del procedimiento.

3.1. Selección del producto

Una vez confeccionado totalmente el procedimiento se hace necesario para su validación realizar la selección de un producto específico, ya que sería inviable realizar dicho procedimiento a más de uno, debido a que la cantidad de código con que se trabaja en cada software es grande y tomaría demasiado tiempo y esfuerzo. La UCI presenta actualmente una gran cantidad de software que cumplen con la característica fundamental que necesita un producto para que le pueda ser aplicado el procedimiento, que no es más que la presencia en este de código mayormente escrito en lenguaje Java Script y PHP. Este grupo de software existentes con dichas características se limita o acota debido a que se hace necesario para poder realizar la validación del procedimiento que el producto de software seleccionado al menos haya comenzado su fase de construcción o desarrollo, habiendo terminado como mínimo una iteración en la cual se haya comenzado a trabajar en la implementación de alguno de los principales módulos que están presentes en el documento visión, esto se debe a que se necesita para lograr observar el verdadero resultado del procedimiento al menos un 40% del total de código para realizarle la evaluación y corrección.

Basado en estos criterios de selección se realiza una selección intencional o no probabilísticas, también llamadas muestras dirigidas (31) debido a que es este el tipo de muestreo aplicado cuando se tiene poco tiempo para acometer la investigación. Fue seleccionado para la aplicación del procedimiento el producto Generador Dinámico de Reportes (GDR) de DATEC, ya que además de cumplir con todas las características antes mencionadas es el producto insignia de DATEC, centro donde se desarrolla la investigación.

Este producto se encuentra actualmente culminado en su versión 1.7. El GDR es una herramienta multiplataforma desarrollada con tecnologías web, utilizando RUP como metodología de desarrollo, Ext JS versión 3.0 para el modelado de interfaces y Symfony 1.1.7 como marco de trabajo. La utilización de Ext JS y Symfony implica que este es un producto que contiene una gran cantidad de código Java Script y PHP. El mismo permite la creación y edición de informes/reportes extrayendo datos de una amplia gama de gestores de bases de datos (32). El GDR permite cargar en una vista estándar los modelos de bases de datos con los que se desea trabajar a través de una interfaz amigable, creada con el objetivo de brindar la mayor cantidad de opciones posibles para el diseño de los reportes.

El procedimiento descrito en el capítulo anterior está confeccionado para ser desarrollado de manera iterativa e incremental, evaluando las porciones de código que se escriben en cada iteración; el principal objetivo de esto es no acumular tanta cantidad de código no evaluado debido a que se hace imposible documentar y reparar tal cantidad de líneas en tan poco tiempo. GDR presenta en la actualidad 30317 líneas de código Java Script y 5067 líneas de código PHP, este código se encuentra totalmente no evaluado por lo cual se tomarán en el caso de Java Script un total de 16588 líneas que representa un 54.7 % del total de líneas y para PHP se tomarán 5067 que representan el 100 % del total. Las líneas de código PHP conforman todos los módulos del GDR incluyendo el componente que provee el API de integración con sistemas externos y en el caso de las líneas de código Java Script conforman los módulos *model_designer*[37]*, query_builder*[38]*, report_manager*[39]y *report_viewer*[40].

3.2. Breve descripción del producto seleccionado

El GDR es una herramienta web que permite a sus clientes consultar los gestores de bases de datos de sus organizaciones y poder generar reportes con la información que estos manejan. Está compuesto por cinco módulos que permitirán conectarse a una base de datos para confeccionar, mover, copiar, eliminar o renombrar los reportes, además exportarlos como HTML, PDF, EXCEL y CSV.

---

[37] *Diseñador de modelos*

[38] *Diseñador de consultas*

[39] *Administrador de reportes*

[40] *Visor de reportes*

Para la aplicación del procedimiento el equipo de trabajo del GDR recibió una pequeña capacitación del uso de cada una de las herramientas de evaluación que se utilizan en el procedimiento.

3.3. Iniciando la estructuración del Plan de Evaluación

Como acción inicial y para comenzar a dar cumplimiento a cada una de las actividades descritas en el procedimiento se dio inicio a la fase de Estructuración. Para ello sesionaron bajo la supervisión del jefe de proyecto, el revisor técnico que desempeña su rol de Evaluador dentro del procedimiento, el Planificador y el Arquitecto, desglosando y definiendo las tareas de cada uno de ellos para confeccionar el Plan de Evaluación (Ver Expediente de Proyecto); quedando según el procedimiento, el Evaluador como el encargado de la selección de dos Desarrolladores, el Arquitecto responsable de reunirse con los Desarrolladores iniciales del producto para definir el estándar de codificación y el Planificador para la realización del cronograma de evaluación y corrección. Se tuvo en cuenta para la realización del Plan de Evaluación (Ver Expediente de Proyecto) que esta evaluación tenía un carácter extraordinario, producto de que en ella se evaluaba la mayor parte del código fuente de la aplicación y este no había recibido ninguna evaluación y corrección previa, por lo que podrían ser detectados una gran cantidad de errores. Esta reunión tuvo una duración de 15 minutos y en ella inicialmente quedaron definidos los nombres del Planificador, del Evaluador y del Arquitecto con su respectivo rol. Cada uno de estos trabajadores tuvo un espacio de 24 horas para recolectar la información que se les solicitó.

3.4. Confección del Plan de Evaluación

Para la confección del Plan de Evaluación (Ver Expediente de Proyecto) se reunieron el Arquitecto, el Evaluador y el Planificador durante treinta minutos para discutir la información que había sido recolectada por cada uno de ellos. Para la codificación de los archivos escritos en lenguaje PHP se adoptó como estándar de codificación un estándar previamente establecido basado en las pautas de codificación PHP de *Zend Framework*, el mismo presenta como principales características la utilización de *camelCaps*[41] para nombrar las variables y para nombrar funciones, además de no permitir la

[41] *Estándar para la nomenclatura similar a las jorobas de un camello que prohíbe el uso de mayúsculas iniciales Ej.: varEjemplo.*

etiqueta de cierre ("?>") para archivos que solo contengan código PHP, manteniendo una indentación[42] de cuatro espacios sin permitir tabulaciones para la misma.

Se detectó entre la información recolectada como principal dificultad para la calidad del código fuente la no existencia en el Centro DATEC de un estándar de codificación para la escritura de código fuente Java Script. Por esta razón fue necesario tomar una decisión inmediata que definiera el estilo de codificación que se debía utilizar para lograr homogeneizar todo el código Java Script con la aplicación de las herramientas. Se tomó la decisión de que la escritura de código Java Script se acercara lo mayormente posible a la propia estructura de Zend Framework para PHP sin olvidar algunas cuestiones importantes para la escritura de código Java Script. Se seleccionaron como características a seguir para la codificación en Java Script:

- Indentación a 4 espacios sin utilizar tabulaciones.
- Romper líneas muy largas después de coma y antes de operadores.
- No permitir espacios entre paréntesis, antes de comas, o punto y comas.
- Espacios a cada lado de signos (===, *!==, II, &&, ¿*) y después de coma.
- Uso obligatorio de (===) y (!==).
- Formato de las sentencias *switch* de la siguiente forma.

```
switch (true) {
    case 1:
        break;
}
```

- Tamaño máximo de línea máximo 120 caracteres.
- Formato de las sentencias de control *if/else/else if*, de la siguiente forma.

[42] *Significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores con el fin de mejorar la legibilidad.*

```
if (true) {
    a = 2;
} else {

}
```

- Formato de las sentencias de control *for*, de la siguiente forma.

```
for (int i = 0; i < a; i ++) {
    b = 2;
}
```

Por ser esta una aplicación extraordinaria del procedimiento producto de la evaluación de la mayoría del código fuente en los dos lenguajes, se decidió realizar el cronograma de evaluación no guiado por requisitos sino por cada uno de los módulos existentes en el producto, aplicándole las herramientas a todos estos.

El Plan de Evaluación quedó totalmente confeccionado tras esta reunión. (Ver Expediente de Proyecto)

3.5. Aplicación de la evaluación

Comenzó a dársele cumplimiento al Plan de Evaluación (Ver Expediente de Proyecto) con el desarrollo de la evaluación o aplicación de las herramientas correspondientes a cada archivo que contenía cada uno de los módulos especificados.

Inicialmente, para recolectar la información necesaria y lograr evaluar el código se aplicó la herramienta Cloc donde se obtuvo el total de líneas de código, de comentario y en blanco de lenguaje PHP y Java Script, y con esta información se documentó la primera sección del Informe de Evaluación (Ver Expediente de Proyecto).

Inmediatamente se comenzaron a aplicar cada una de las demás herramientas en el siguiente orden:

PHPCS, PHPMD, PHPCPD, PHP Depend y JSLint.

Los errores detectados fueron documentado en el Informe de Evaluación, surgiendo como principales errores, la indentación incorrecta, utilización de tabulaciones, variables y métodos no utilizados y de gran tamaño, líneas que sobrepasaban el nivel aceptable de 80 caracteres y el máximo de 120, variables que no siguen el estándar de nomenclatura *camelCaps*, aparición de etiquetas de cierre (?>) en todos los archivos PHP. En el caso de Java Script además se detectó el uso incorrecto de algunas

funciones que pudieran traer afectaciones funcionales como el *parseInt()*[43] sin pasarle la base 10 como parámetro adicional, *switch* con *break* que nunca llegan a ejecutarse y sin *default*. Para Java Script se detectaron 6165 errores de 13111 líneas de código evaluadas que representa un 47% de líneas con errores como se muestra en la Figura # 16. Para PHP se detectaron 2046 que representan un 54% de líneas con errores (Ver Figura # 16) de los cuales 1898 fueron violaciones en el cumplimiento del estándar de codificación que representan un 50% del total de líneas y un 90% del total de errores para el lenguaje PHP, lo cual demuestra que a pesar de la definición previa a la implementación de un estándar de codificación se omiten al menos en algún momento varias de las pautas establecidas por este. El 10% restante de errores pertenecían a fallas de tamaño y complejidad, y código duplicado, con un 7% y un 3% respectivamente respecto al total de errores para PHP como se muestra en la Figura # 17, detectándose un 1,70% de código duplicado y un 3,87% de problemas de tamaño y complejidad respecto al total de líneas ejecutables analizadas de 3766. Toda la información extraída fue documentada en el Informe de Evaluación para que pudiera ser evaluado en bajo, medio o alto según la métrica que se muestra en la métrica del capítulo 2. Para el total cumplimiento de la tarea de evaluación fueron empleadas 5 sesiones de 8 horas, de las cuales se tomó una hora de cada una para aplicar las herramientas sobre el código fuente, ya que la mayoría del tiempo se empleó documentando los errores detectados por cada una de las herramientas.

[43] *Función para convertir una cadena de caracteres a enteros.*

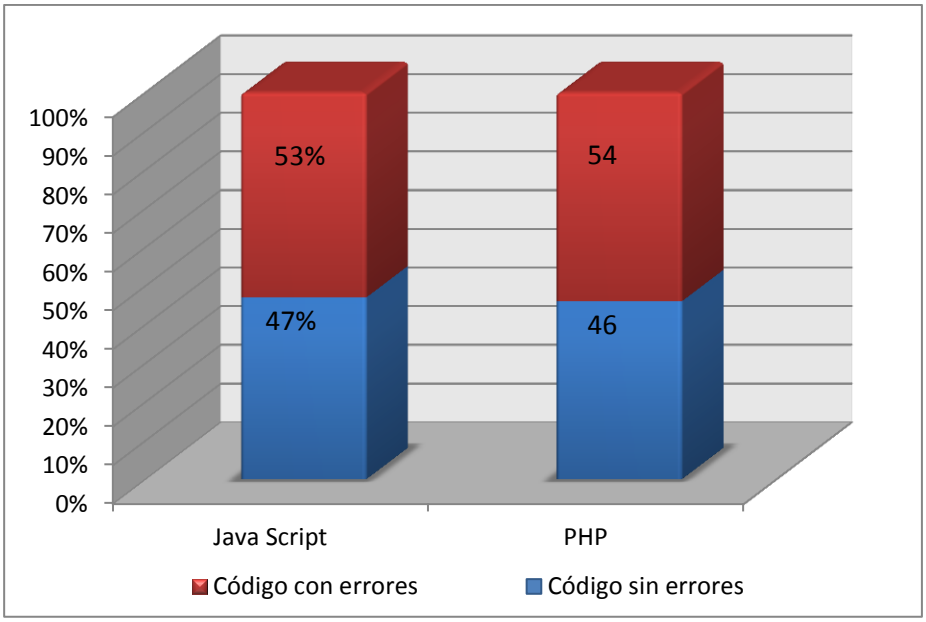

Figura # 16 Porcentaje de código con errores antes de corregir

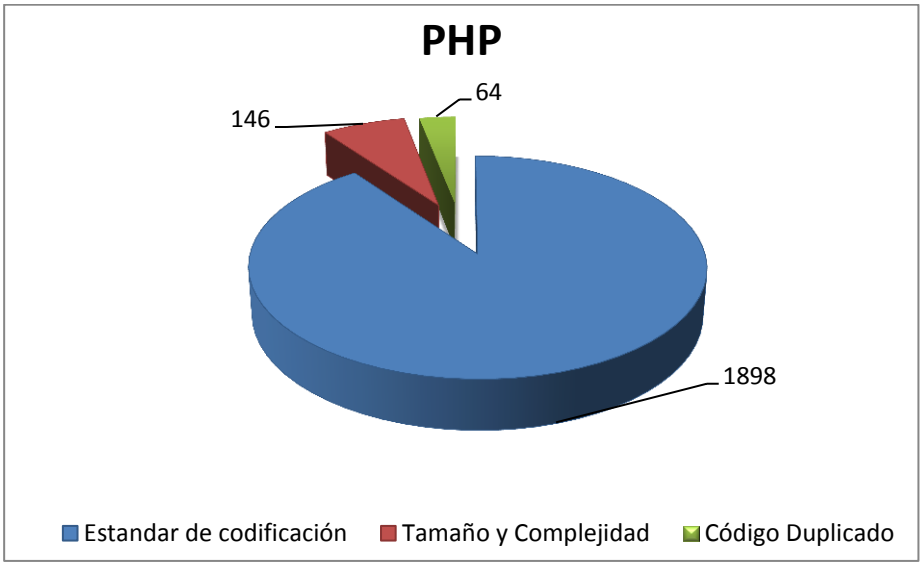

Figura # 17 Porcentaje por tipo de error para PHP antes de realizar corrección

3.6. Corrección de errores detectados

Para comenzar la actividad de corrección se revisaron cada uno de los errores que estaban descritos en el Informe de Evaluación, seleccionando a cuales de ellos se les aplicaría la corrección. Para esto, se reunieron los Desarrolladores iniciales y el Evaluador, detectando un total de 42 errores de gran

complejidad los cuales fueron documentados en la Solicitud de Cambio para su aprobación o no aprobación.

Fueron aceptados 26 errores del total solicitado. Las principales causas de la no aprobación de estos fueron:

- Única solución para el problema.
- Cambios con efectos colaterales que afectan otras funcionalidades.
- Métodos a utilizar en implementaciones futuras.
- Problemas con cierta complejidad para modificar de cara a una próxima liberación.

Por parte de los Desarrolladores-Correctores los cuales tienen la autoridad de corregir gran parte de los problemas de estándar de codificación se decidió:

- Disminuir la cantidad de líneas de código del software permitiendo la ocurrencia de todas las líneas de más de 80 caracteres y de algunas líneas de más de 120 caracteres (específicamente las que contenían mensajes de texto producto del cual dicha línea sobrepasaba la cantidad de caracteres).
- No reparar algunos de los problemas que detectan las herramientas producto que estos no constituían una afectación real a la calidad de la codificación.

Fue utilizada para la corrección de errores el IDE NetBeans la cual utilizaron los Desarrolladores-Correctores para realizar su tarea.

La corrección contó para su total cumplimiento de cuatro sesiones de trabajo de seis horas cada una donde fueron corregidos un promedio de 8200 errores de los cuales los de mayor complejidad fueron documentados en el Informe de Resultados de Corrección.

3.7. Reevaluación del código

Después de haberse completado la actividad de corrección y verificado el eficaz funcionamiento de cada una de las funciones que fueron afectadas con cada corrección, se realizó la reevaluación del código. Se aplicaron las herramientas de evaluación en el mismo orden, de las cuales se obtuvieron

datos relevantes que permitieron calificar de efectiva la corrección anteriormente realizada (Ver Expediente de Proyecto).

Fueron detectados 404 errores de los cuales el 67% pertenecían a Java Script y el resto a PHP. Entre estos errores se detectaron una minoría de errores nuevos insertados en la corrección, la gran mayoría de los errores detectados en esta reevaluación fueron los que por motivos específicos discutidos no se les aplicó la corrección (Ver Figura # 18).

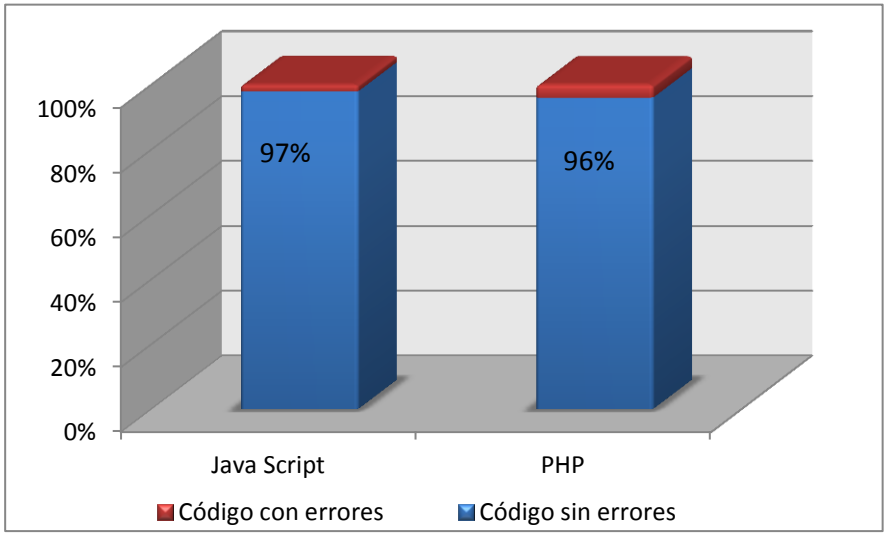

Figura # 18 Porcentaje de código con errores después de corregir

La actividad de reevaluación fue debidamente documentada para lograr hacer una comparación entre el producto antes y después de aplicada la corrección.

Fueron eliminadas en el caso de Java Script un total de 1140 líneas que representan un 8,7% del total inicial evaluadas para este lenguaje, de las cuales 562 eran líneas ejecutables y el resto eran líneas sobrantes que existían entre líneas de código, estas se eliminaron sin llegar a obstruir o dificultar la legibilidad del código, además se adquirió un alto nivel de uniformidad con la reparación de las líneas ajustándose todas al estilo definido.

Para los archivos escritos en lenguaje PHP la corrección condujo también a disminuir la cantidad de líneas de 5067 iniciales a 4526 posterior a esta. A diferencia de Java Script, en los archivos PHP la cantidad de espacios en blanco aumentó de 544 existentes al inicio a 580, sucediendo de la misma manera con las líneas de comentario que aunque no fue un aumento considerable se debe tener en

cuenta ya que se emplearon algunas líneas extras con la división de líneas extremadamente largas. Esta información presupone un código mucho más legible, mejorado, y por consiguiente más mantenible, lo que ratifica la correcta aplicación de dicha labor de corrección para este lenguaje.

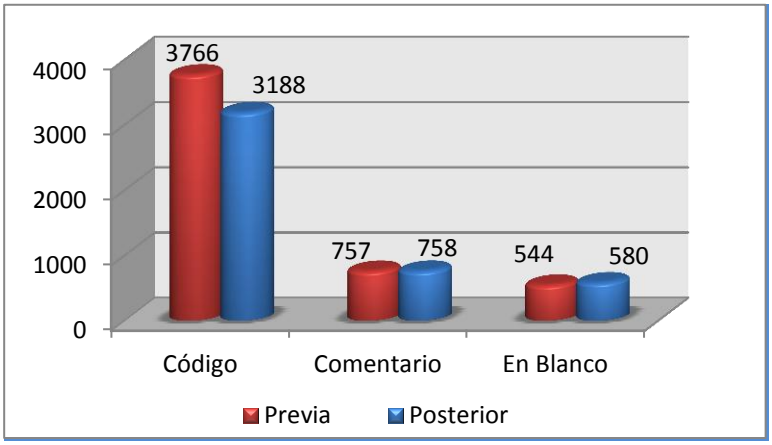

Figura # 19 Comparación previa y posterior a la corrección del código PHP

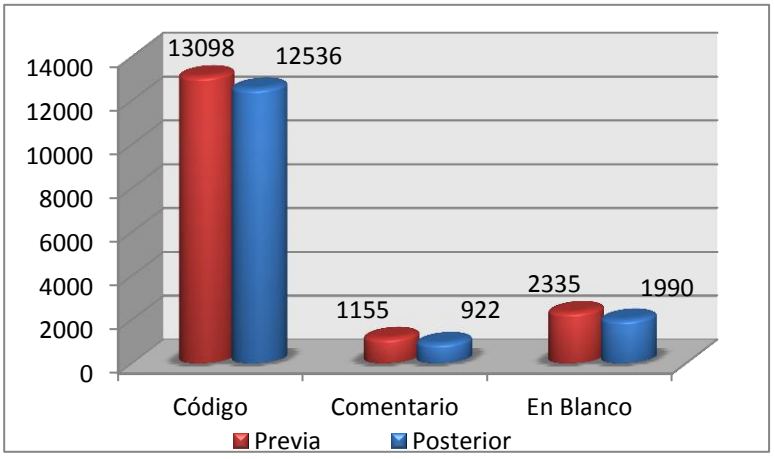

Figura # 20 Comparación previa y posterior a la corrección del código Java Script

3.8. Cloc para disminuir la cantidad de líneas sin llegar a ofuscar

La decisión de aplicar o no la herramienta Cloc con el fin de eliminar todas las líneas de comentarios y en blanco fue tomada por el Arquitecto, el mismo decidió que se le aplicaría dicha herramienta al código. Para el caso que fuera necesario realizar cualquier reparación emergente y que fuera necesario analizar nuevamente el código fuente de la aplicación se dejaría una copia de mantenimiento que tuviera el código debidamente espaciado y con todos los comentarios (como se encuentra después de aplicada la corrección).

La aplicación de esta herramienta tuvo grandes ganancias en cuanto a tamaño de los archivos ya que se eliminaron un total de 4250 líneas lo cual representa una disminución del 22% del total de líneas obtenidas después de aplicada la actividad de corrección (Ver Figura # 21).

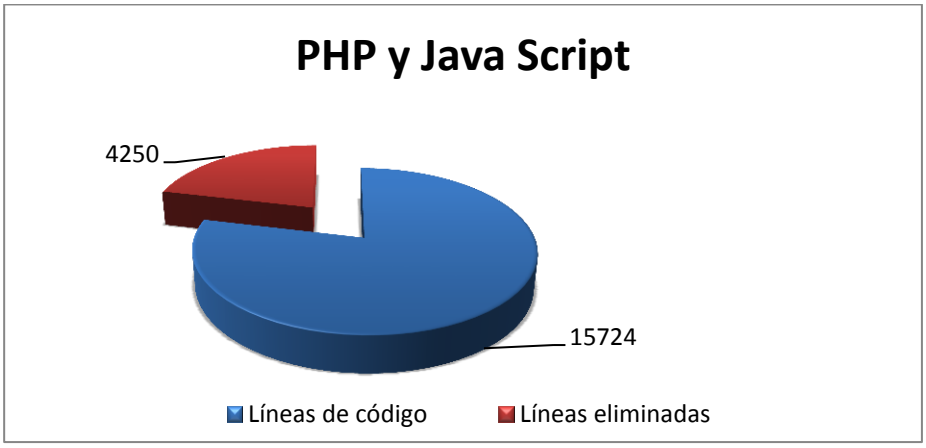

Figura # 21 Total de líneas eliminadas después de aplicada la herramienta Cloc

3.9. Ganancias generales después de aplicado el procedimiento

Inicialmente la porción de producto analizado contaba con un total de 21655 líneas de código entre Java Script y PHP; entre este total existían líneas de código, de comentario y otras en blanco. Luego de aplicado totalmente el procedimiento, esta cantidad descendió a 15724, lo que representa un 27% de reducción de tamaño del producto respecto a esas líneas analizadas (Ver Figura # 22). Al mismo tiempo estos valores disminuyen además el tiempo que será utilizado en un futuro para realizarle mantenimiento a este software. Acompañado de este importe de líneas disminuidas, también se encuentra el grado de igualdad con que está escrito el software para los dos lenguajes después de aplicado el procedimiento, esto también es un factor determinante en la rapidez con que será reparado o actualizado el software.

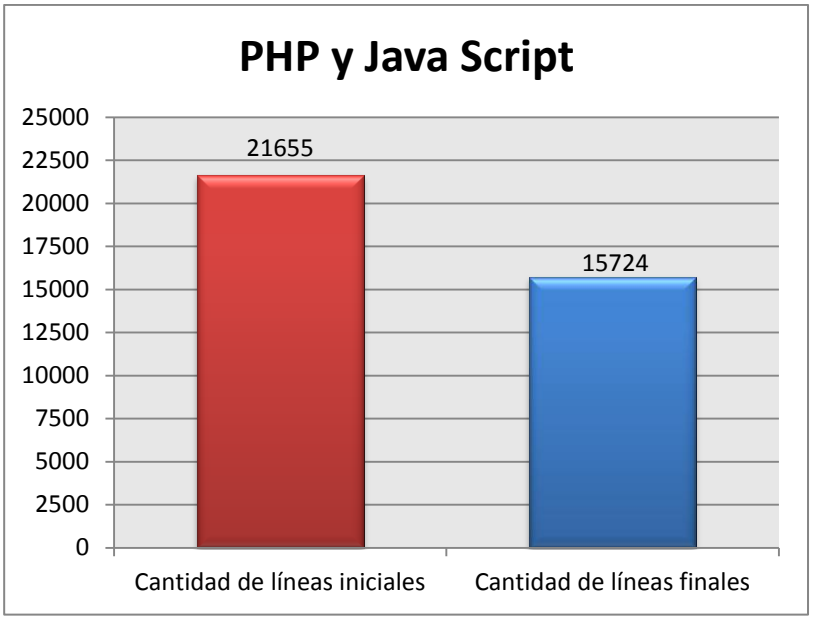

Figura # 22 Reducción obtenidas en líneas de código

Conclusiones Parciales

En este capítulo se describió la aplicación del procedimiento de evaluación y perfeccionamiento de la calidad del código PHP y Java Script, seleccionando para esta tarea al producto del Centro DATEC Generador Dinámico de Reportes. Se tomó una porción de código fuente a la cual se le aplicaron las herramientas correspondientes para lograr realizar la evaluación de la calidad con que estaba escrito dicho código, detectándose un grupo de errores, siendo los más significativos los problemas en incumplimiento de normas de codificación. Se realizó la corrección de los defectos detectados y se comprobaron los niveles de calidad alcanzados después de realizada la corrección, mostrando con datos reales los problemas que existían en el código y las ganancias adquiridas con la realización de la corrección y del procedimiento en general mediante gráficas.

Conclusiones

Al aplicar el procedimiento y luego de los resultados agenciados, se le da total cumplimiento al objetivo por el cual se realizó la investigación:

- ✓ Se realizó una exploración profunda sobre la posible existencia de investigaciones similares, analizando una gran cantidad de bibliografía relacionada con la evaluación de software, así como las normas de evaluación existentes, lo cual permitió conocer los elementos necesarios para realizar la total creación del procedimiento y tener una base que pudiera servir para guiar a los equipos de desarrollo en cuanto a los estándares de codificación que deben seguir.
- ✓ Se propuso un procedimiento que engloba roles, responsabilidades, actividades y artefactos para llevar a cabo la evaluación y perfeccionamiento del código fuente.
- ✓ Se aplicó el procedimiento desarrollado en uno de los productos desarrollados en la UCI, con el objetivo de poder comprobar si el mismo soluciona los problemas existentes, obteniéndose como resultado una disminución de las líneas de código y un equilibrio en el estilo de programación que en este se utiliza.

Recomendaciones

Se recomienda:

- ✓ Continuar investigando acerca de nuevas herramientas que puedan ser insertadas dentro del procedimiento; principalmente para aplicarse al lenguaje Java Script en cuanto a estándares de codificación específicos.
- ✓ Estudiar con mayor profundidad la norma ISO 25000 para enriquecer el procedimiento en cuanto a pasos y actividades a realizar.

Referencias Bibliográficas

1. **Sommerville, I.** *Ingeniería de Software.* s.l. : Pearson Educación, 2002.

2. **Jacaboson, I, Booch, G y Rumbaugh, J.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.

3. **Pressman, R.S.** *Ingeniería del Software: Un enfoque práctico.* s.l. : McGraw-Hill, 1993.

4. Sitio oficial de La Universidad de Ciencias Informáticas. [En línea] [Citado el: 3 de Noviembre de 2010.] http://uci.cu.

5. Definición de Calidad de Software. [En línea] [Citado el: 10 de noviembre de 2010.] http://www.mitecnologico.com.

6. Kioskea. [En línea] [Citado el: 21 de Diciembre de 2010.] http://es.kioskea.net/contents/qualite/qualite-introduction.php3.

7. **Deming, Edwards.** *Out of the Crisis.* Cambridge : Cambridge University Press. 1986.

8. **Pressman, Rogert.** *Aspectos y métricas de la calidad del Software.* 1998.

9. **Varios.** *Standard for Software Quality Management System.* 1992.

10. **Domínguez Vera, Edgar Danilo y Martínez Flores, José Luis.** Programa Doctoral en Ingeniería de Sistemas. *La enseñanza de métricas de software.* 1999. Vol. ll, 5.

11. ISO25000. [En línea] [Citado el: 29 de Enero de 2011.] http://iso25000.com/.

12. **Querétaro.** Norma 9126. *Métricas Internas de calidad del producto de software.*

13. Calidad de Software. [En línea] euriko. [Citado el: 13 de Noviembre de 2010.] http://www.euriko.com.mx/euriko-ia/39-productos/76-calidad-de-software.html.

14. **Rosenberg.** *Some Misconceptions About Lines of Code.* s.l. : Fourth International Software Metrics Symposium, 1997.

15. **Park, R. E.** *Software Size Measurement: A Framework for Counting Source.*

16. *A Complexity Measure.* **McCabe, TJ.** Proceedings of the 2nd international conference on Software engineering.

17. **Fenton, M. Neil.** *Software Metrics: successes, failures and new directions.* s.l. : Elsevier, 1999.

18. **Baker, B. S.** *On Finding Duplication and Near-Duplication in Large Software Systems.* s.l. : Proceedings of the Second Working Conference on Reverse Engineering, 1995.

19. **Mayrand, J, Leblanc, C y Merlo, E M.** *Experiment on the Automatic Detection of Function Clones in Software System Using Metrics.* s.l. : International Conference on Software Maintenance, 1996.

20. **Álvarez, J. L. M.** *Aplicación de un Sistema Experto para el desarrollo de Sistema Evaluador del modelo Capability Maturity Model (CMM) niveles dos y tres.* México : Universidad de las Américas, 2004.

21. **9126, ISO/IEC "ISO/IEC.** *Information technology - Software product evaluation -Quality characteristics and guidelines for their use, JTC 1 Organization.* s.l. : ISO, 1991.

22. Metabetageek. [En línea] [Citado el: 15 de Enero de 2011.] http://metabetageek.com/2010/01/01/php-best-practices-resources-for-coding-standards/.

23. **Motta, Eduardo De La.** eduardodelamotta. [En línea] [Citado el: 14 de Enero de 2011.] http://eduardodelamotta.com/2008/por-que-usar-estandares-en-la-codificacion.html.

24. Dagbladet. [En línea] [Citado el: 21 de Enero de 2011.] http://www.dagbladet.no/development/phpcodingstandard/.

25. Possibility. [En línea] [Citado el: 28 de Enero de 2011.] http://www.possibility.com/Cpp/CppCodingStandard.html.

26. **Cristalab.org.** Reglas de codificación y lineamientos de código PHP. [En línea] [Citado el: 9 de Enero de 2011.] http://www.cristalab.com/tutoriales.

27. **Crockford.** javascript. [En línea] [Citado el: 26 de Enero de 2011.] http://javascript.crockford.com/code.html.

28. **Core, Axia.** Axia Core. [En línea] [Citado el: 15 de Enero de 2011.] http://axiacore.com/estandares/.

29. Página Oficial de ISO. [En línea] [Citado el: 11 de Noviembre de 2010.] http://www.iso.org.

30. ¿Que es un procedimiento? [En línea] [Citado el: 20 de febrero de 2011.] http://www.ucm.es/info/Psyap/taller/procedimientos/sld002.htm. ISO-8859-1.

31. **Sampieri, M. en C. Roberto Hernández.** *Metodología de la Investigación.* México : Mc Graw Hill, 1998. 970-10-1899-0.

32. **Informaticas, Universidad de las Ciencias.** *Modelo de sistema.* La Habana : s.n., 2010.

33. Wikilearning. [En línea] [Citado el: 21 de Diciembre de 2010.] http://www.wikilearning.com/curso_gratis/teorias_de_calidad-aportaciones_de_kaoru_ishikawa/11500-10.

Bibliografía

¿Que es un procedimiento? [En línea] [Citado el: 20 de febrero de 2011.] http://www.ucm.es/info/Psyap/taller/procedimientos/sld002.htm. ISO-8859-1.

9126, ISO/IEC "ISO/IEC. 1991. *Information technology - Software product evaluation -Quality characteristics and guidelines for their use, JTC 1 Organization.* s.l. : ISO, 1991.

A Complexity Measure. **McCabe, TJ.** Proceedings of the 2nd international conference on Software engineering.

Álvarez, J. L. M. 2004. *Aplicación de un Sistema Experto para el desarrollo de Sistema Evaluador del modelo Capability Maturity Model (CMM) niveles dos y tres.* México : Universidad de las Américas, 2004.

Baker, B. S. 1995. *On Finding Duplication and Near-Duplication in Large Software Systems.* s.l. : Proceedings of the Second Working Conference on Reverse Engineering, 1995.

Calidad de Software. [En línea] euriko. [Citado el: 13 de Noviembre de 2010.] http://www.euriko.com.mx/euriko-ia/39-productos/76-calidad-de-software.html.

Core, Axia. Axia Core. [En línea] [Citado el: 15 de Enero de 2011.] http://axiacore.com/estandares/.

Cristalab.org. Reglas de codificación y lineamientos de código PHP. [En línea] [Citado el: 9 de Enero de 2011.] http://www.cristalab.com/tutoriales.

Crockford. javascript. [En línea] [Citado el: 26 de Enero de 2011.] http://javascript.crockford.com/code.html.

Dagbladet. [En línea] [Citado el: 21 de Enero de 2011.] http://www.dagbladet.no/development/phpcodingstandard/.

Definición de Calidad de Software. [En línea] [Citado el: 10 de noviembre de 2010.] http://www.mitecnologico.com.

Deming, Edwards. *Out of the Crisis.* Cambridge : Cambridge University Press. 1986.

Domínguez Vera, Edgar Danilo y Martínez Flores, José Luis. 1999. Programa Doctoral en Ingeniería de Sistemas. *La enseñanza de métricas de software.* 1999. Vol. ll, 5.

Fenton, M. Neil. 1999. *Software Metrics: successes, failures and new directions.* s.l. : Elsevier, 1999.

Informaticas, Universidad de las Ciencias. 2010. *Modelo de sistema.* La Habana : s.n., 2010.

ISO25000. [En línea] [Citado el: 29 de Enero de 2011.] http://iso25000.com/.

BIBLIOGRAFÍA

Jacaboson, I, Booch, G y Rumbaugh, J. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.

Kioskea. [En línea] [Citado el: 21 de Diciembre de 2010.] http://es.kioskea.net/contents/qualite/qualite-introduction.php3.

Mayrand, J, Leblanc, C y Merlo, E M. 1996. *Experiment on the Automatic Detection of Function Clones in Software System Using Metrics.* s.l. : International Conference on Software Maintenance, 1996.

Metabetageek. [En línea] [Citado el: 15 de Enero de 2011.] http://metabetageek.com/2010/01/01/php-best-practices-resources-for-coding-standards/.

Motta, Eduardo De La. eduardodelamotta. [En línea] [Citado el: 14 de Enero de 2011.] http://eduardodelamotta.com/2008/por-que-usar-estandares-en-la-codificacion.html.

Página Oficial de ISO. [En línea] [Citado el: 11 de Noviembre de 2010.] http://www.iso.org.

Park, R. E. *Software Size Measurement: A Framework for Counting Source.*

Possibility. [En línea] [Citado el: 28 de Enero de 2011.] http://www.possibility.com/Cpp/CppCodingStandard.html.

Pressman, R.S. 1993. *Ingeniería del Software: Un enfoque práctico.* s.l. : McGraw-Hill, 1993.

Pressman, Rogert. 1998. *Aspectos y métricas de la calidad del Software.* 1998.

Querétaro. Norma 9126. *Métricas Internas de calidad del producto de software.*

Rosenberg. 1997. *Some Misconceptions About Lines of Code.* s.l. : Fourth International Software Metrics Symposium, 1997.

Sampieri, M. en C. Roberto Hernández. 1998. *Metodología de la Investigación.* México : Mc Graw Hill, 1998. 970-10-1899-0.

Sitio oficial de La Universidad de Ciencias Informáticas. [En línea] [Citado el: 3 de Noviembre de 2010.] http://uci.cu.

Sommerville, I. 2002. *Ingeniería de Software.* s.l. : Pearson Educación, 2002.

Varios. 1992. *Standard for Software Quality Management System.* 1992.

Wikilearninghttp://www.wikilearning.com/curso_gratis/teorias_de_calidad-aportaciones_de_kaoru_ishikawa/11500-10