Automatización de pruebas unitarias de códigos PHP

Unit Test Automation of PHP source code

Alejandro Villa Betancur, Jorge E. Giraldo Plaza
Ingeniería Informático, Politécnico C. Jaime Isaza Cadavid
jegiraldo@elpoli.edu.co
alejo227@yahoo.com

***Resumen*— Las pruebas unitarias de software permiten evitar errores a gran escala que con el paso del tiempo requieren mayor inversión para su corrección. Sin embargo a veces los frameworks de apoyo disponibles para realizar estas pruebas no son suficientes por sí solos y necesitan ser automatizados. En este artículo se presenta el desarrollo de una estrategia de diseño para automatización de pruebas unitarias de códigos PHP utilizando el framework PHPUnit y su posterior desarrollo empleando la metodología WEBML.**

***Palabras clave*— Pruebas Unitarias, PHP, PHPUnit, Ingeniería Web, WebML**

***Abstract*— Software unit tests allow preventing large-scale errors that over the years require major money and time invest for their correction. Nevertheless sometimes the available frameworks to perform these tests are not enough by itself and they need to be automated. This article presents the development of a design strategy for automation of unit tests PHP codes developed as academic degree work at the Colombian Polytechnic Jaime Isaza Cadavid.**

***Key Word* — Unit Test, PHP, PHPUnit, Web Engineering, WebML**

I. INTRODUCCIÓN

Las pruebas unitarias de software permiten evaluar por separado el correcto funcionamiento de los códigos que lo componen. En la actualidad, existen diferentes soluciones que permiten desarrollar las pruebas unitarias, pero carecen de facilidad en su manejo, lo que conlleva a que el programador desista de su uso. Por lo anterior es necesario contar con un componente intermedio que permita la comunicación con dichas librerías o frameworks de una manera sencilla, rápida y eficaz. De esta manera se podrá llevar a cabo la ejecución de dichas pruebas para evaluar posteriormente la funcionalidad del código escrito y dar paso a realizar un análisis de sus resultados.

La siguiente estrategia de diseño permite llevar a cabo la automatización de pruebas unitarias para archivos de código escritos en el lenguaje PHP a través de un módulo web llamado PHP Testing Studio que es desarrollado a su vez en dicho lenguaje.

Fecha de Recepción: 26 de Agosto de 2011
Fecha de Aceptación: 28 de Abril de 2012

Existen dos enfoques principales para el diseño de pruebas unitarias. Un enfoque de caja blanca orientado a la estructura del código y otro enfoque de caja negra orientado al correcto funcionamiento de éste a partir del análisis de entradas y salidas que posee y verificando que el resultado es el esperado [1]. El primer enfoque resulta ser demasiado subjetivo ya que depende directamente del ambiente y equipo de desarrollo que posee su propia forma de generar estructuras para el código. Por esta razón, este enfoque no será automatizado en el proceso de pruebas realizado por PHP Testing Studio. Sin embargo, se propone adoptar un estándar de codificación que pretende generalizar una estructura específica de códigos PHP a probar. Dicho estándar de codificación ya ha sido implementado en Visual Systems de Colombia S.A., empresa desarrolladora de software para documentos inteligentes, por un grupo de tres analistas del área de desarrollo e investigación encargados del diseño de aplicaciones Web en dicho lenguaje y ha mejorado significativamente los procesos de codificación a partir del momento de su implementación.

El documento se estructura de la siguiente manera: En la sección 2 se presenta un estándar de codificación definido; en la sección siguiente se presenta una estrategia de diseño de las pruebas, posteriormente en la sección 4 y 5 se expone el modelado de la aplicación basada en WEBML; finalmente vienen las conclusiones y bibliografía empleada.

II. ESTÁNDAR DE CODIFICACIÓN DISEÑADO

Un estándar de codificación es un conjunto de reglas que se utilizan para escribir archivos de código fuente con el objetivo de lograr estructuras de código mucho más comprensibles e identificables para otros programadores diferentes al autor.

Para realizar la correcta automatización de pruebas unitarias para códigos PHP se determina que es necesario proponer un estándar de codificación que permita generalizar la estructura de los códigos a ser probados ya que ésta depende directamente de las reglas de desarrollo de cada ambiente y desarrollador. Como modelo a seguir para la generación de códigos PHP se propone como autoría un estándar de codificación del cual sus principales

reglas se muestran en la Tabla 1 para llevar a cabo el correcto funcionamiento del módulo PHP Testing Studio.

REGLA	DESCRIPCIÓN

Comentarios de funciones	Es recomendado que cada función tenga comentarios que expliquen su funcionalidad.
	Ejemplo:
	`<?php`
	`// Esta función imprime hola mundo`
	`function Saludar(){`
	`return("Hola mundo!");`
	`}`
	`?>`

Declaración e indicación de tipos de parámetros de funciones	PHP no es un lenguaje fuertemente tipado, es decir, determinada variable puede cambiar su tipo de dato en medio de la ejecución del programa, lo que puede alterar significativamente el comportamiento de las funciones y generar resultados inesperados.
	Para evitar esto, se propone la posibilidad de indicar el tipo de los parámetros de cada función utilizando un comentario separado por un espacio inmediatamente después de la declaración de cada parámetro con el tipo deseado (*boolean, integer, float, string, array, object*) entre los asteriscos del comentario, de la siguiente manera:
	Ejemplo:
	`function Sumar($a /*integer*/, $b /*string*/){`
	`echo $b;`
	`$a = $a*$a;`
	`return($a);`
	`}`

Buenas prácticas de programación	Al escribir un archivo de código PHP deben tenerse en cuenta las siguientes buenas prácticas de codificación: • El código debe apuntar a comportarse de acuerdo a las especificaciones. • El código debe utilizar un mínimo de recursos de tiempo y memoria. • El código debe ser fácil de leer y comprender. • El código debe ser fácil de depurar. • El código debe ser fácil de mantener. • La interfaz de usuario debe ser independiente de las funciones lógicas.

Tabla 1. Estándar de codificación PHP

III. ESTRATEGIA DE DISEÑO DE PRUEBAS

La estrategia se fomenta en un procedimiento dividido en pasos bajo los cuales el desarrollador o encargado de pruebas podrá seleccionar sus archivos de código PHP o módulos de clase, archivos que contienen funciones que ejecutan tareas específicas entregando resultados a partir de una serie de posibles entradas de datos.

La prueba unitaria busca responder si las funciones que componen el código cumplen o no con el comportamiento esperado. A través de esta estrategia de automatización el proceso será automatizado y se apoyará en el framework de pruebas unitarias PHPUnit.

El proceso además llevará a cabo el diseño de los casos de prueba de la función y la ejecución de éstos con el fin de obtener resultados que serán organizados y presentados en forma de reportes al encargado de las pruebas para apoyarlo a tomar decisiones basadas en estos reportes.

La estrategia de diseño se compone de los pasos que se detallan a continuación.

PASO 1: Seleccionar Módulos de Clase: Un módulo de clase es la unidad de código más pequeña a la cual puede realizarse una prueba unitaria [5]. Se seleccionarán de uno en uno los archivos de este tipo para llevar a cabo la ejecución de las pruebas mediante el módulo PHP Testing Studio (Módulo Desarrollado, se explicará su modelado adelante).

Estos archivos seleccionados son cargados en memoria y dan paso a la generación de la creación de una nueva carpeta en el disco duro, en la ruta seleccionada. De este modo por ejemplo, para el archivo Calculadora.php la carpeta a generar será testCalculadora. La Figura 1 permite entender mejor los procesos realizados en este paso.

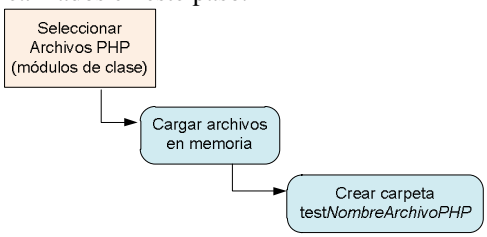

Figura 1. Selección de módulos.

PASO 2: Validar estándar de codificación: Después de realizar la selección de los módulos de clase PHP comienza un ciclo iterativo por cada uno de los archivos de código PHP seleccionados. El archivo pasa por una validación del estándar de codificación propuesto. Enla Figura 2 se expone el paso 2.

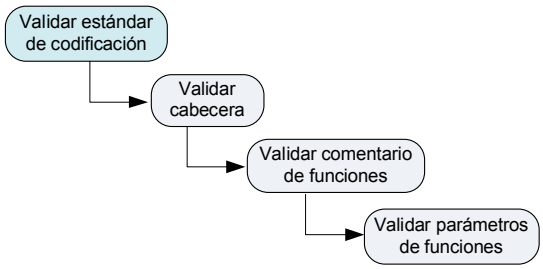

Figura 2. Selección de módulos.

PASO 3: Identificar tipos de parámetros: Después de validar el estándar de codificación del archivo seleccionado, se detectan las funciones o métodos que lo componen y por cada parámetro de cada función o método se identifica el tipo de parámetro. Los tipos de parámetros seleccionados serán almacenados en memoria para su posterior manejo. Su flujo se presenta en la Figura 4.

PASO 3: Identificar *Asserts*: Luego de identificar los tipos de parámetros de las funciones se identifican los posibles asserts a partir de estos. Un assert es la salida o resultado esperado de cierta entrada en la función o método a probar. En PHPUnit son una colección de métodos estáticos para verificar los valores actuales con los valores esperados [6]. Ver Figura 5.
A partir del tipo de parámetros identificados se generan los posibles asserts. Para esto, se escriben comentarios especiales antes de la declaración de la función o método con la etiqueta de inicio de comentario con doble asterisco (/**) y la estructura según el tipo de assert como se muestra en la Figura 3y 4.

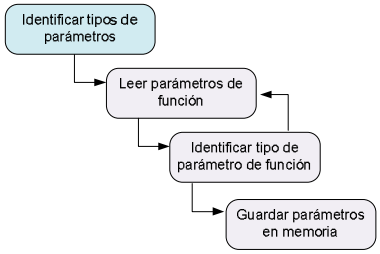

Figura 3. Identificar Tipos de Parámetros

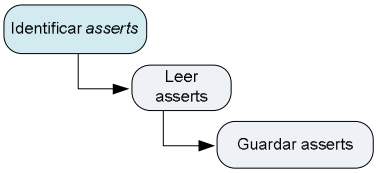

Figura 4. Identificar Asserts

PASO 5: Generar casos de Prueba: Se genera o diseña una clase con los casos de prueba del archivo PHP a partir del uso de la librería PHPUnit y su componente PHPUnit Framework TestCase. Con este componente es posible diseñar y ejecutar diferentes casos de prueba a partir de la lista de entradas generada en el paso anterior. Dichos casos de prueba pueden ser ejecutados repetidamente cuando sea necesario. En la Figura 5 se presenta las actividades que integran el paso 5.

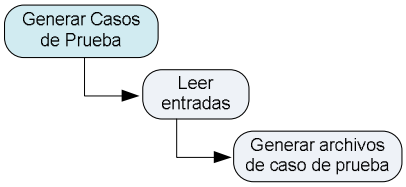

Figura 5. Generar Casos de Prueba

PASO 6: Ejecutar Prueba con PHPUnit: Se ejecuta la prueba mediante el Framework de pruebas PHPUnit. Se valida e inspeccionan los archivos de casos de prueba definidos, verificando que se han contemplado todas las pruebas necesarias, que no existen pruebas duplicadas o implícitas en otras y que el tiempo estimado no sobrepasa la fecha límite de ejecución.

Se invoca el ejecutable phpunit.bat y se envía la ruta en que se encuentra el archivo a probar. PHPUnit generará un log que puede ser capturado y enviado a un archivo temporal para preparar los resultados en el siguiente paso. Este proceso se muestra a continuación en la Figura 6.

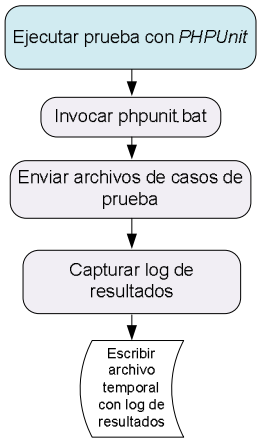

Figura 6. Ejecución de Pruebas.

PASO 7: Preparar Resultados: Por cada una de las pruebas ejecutadas se produce un log de resultados. Dicho log indicará:

Número de pruebas: El número de pruebas que han sido llevadas a cabo mediante la invocación de PHPUnit y los archivos de casos de prueba en PHP.

Número de aciertos: Es el número de resultados esperados a partir de los datos de entrada ingresados después de una ejecución de prueba mediante PHPUnit.

Número de fallos: Es el número de resultados no esperados a partir de los datos de entrada ingresados después de una ejecución de prueba mediante PHPUnit.

IV. DESARROLLO DE MÓDULO WEB EMPLEANDO WEBML

Se realizó un modelado de la estrategia de diseño para la automatización de pruebas unitarias de código PHP, haciendo uso de WEBML. Realizando una descripción detallada de los requerimientos necesarios para desarrollar el módulo web.

Se definen cuatro grupos de usuarios para la aplicación: Administradores, desarrolladores, analistas y encargados de pruebas como se muestra en la Figura 7.

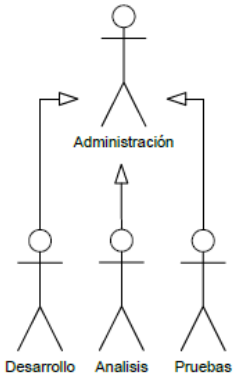

Figura 7. Grupo de usuarios definidos.

En la Figura 8 se presenta el modelo de hipertexto definido para el paso de selección de módulos a probar.

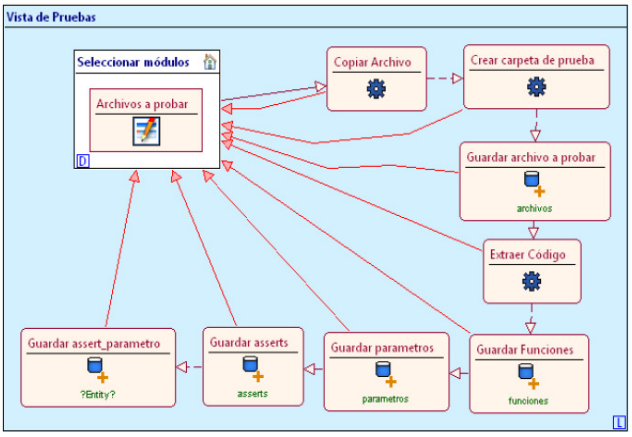

Figura 8. Modelo de Hipertexto.

En la Figura 9 se presenta la vista arquitectónica del módulo desarrollado. Su esquema se basa en la arquitectura cliente/servidor.

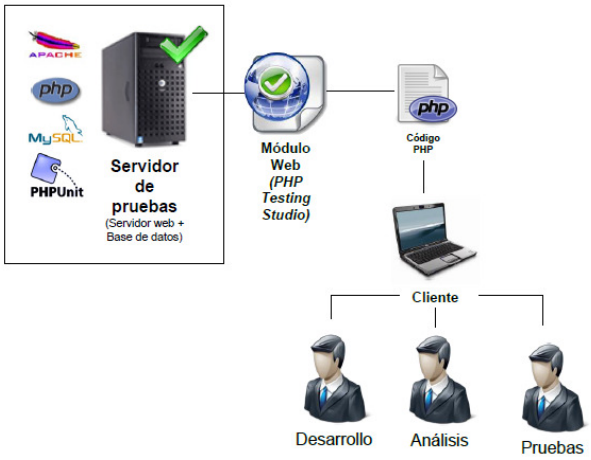

Figura 8. Modelo de Arquitectura.

La implementación de la aplicación tuvo los siguientes aspectos

Hardware:

- Procesador de 1000 MHz (Se recomienda procesador de 2.4GHz doble núcleo).
- Memoria RAM de 512MB (Recomendado 1GB)
- 1 GB de espacio libre en disco
- Pantalla de 800 x 600 pixel (Recomendada una resolución de 1280 x 1024 pixel)

Software:

- Sistema Operativo: Windows XP, Windows 7, Windows Server 2003, Windows Server 2008.
- Xampp 1.7.4 (Apache, MySQL, PHP)
- PHP Unit 5.3

A continuación se presenta en la Figura 9 la pantalla de inicio del módulo PHP Studio. Se puede apreciar los componentes para la selección de los códigos fuente PHP.

Figura 9. Pantallaza Principal.

Finalmente el software presenta un reporte, en el cual se detallan los resultados de cada uno de los códigos y determina si las pruebas fueron satisfactorias.

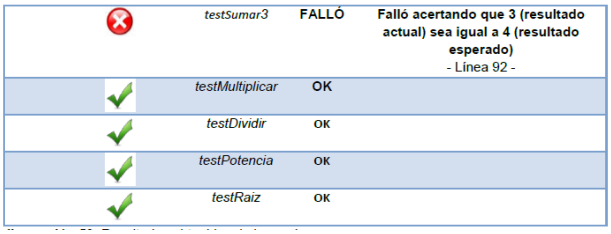

Figura 10. Reporte de resultados.

V. CONCLUSIONES Y RECOMENDACIONES

Las pruebas por sí solas no mejorarán la calidad del software desarrollado, sin embargo, algunos de los posibles problemas, errores y fallos de éste pueden ser detectados a tiempo en medio de su desarrollo. Si este proceso es automático permitirá que el desarrollador o personal de pruebas pueda realizar verificaciones a tiempo que eviten errores mayores y ahorre tiempos y recursos de corrección.

Debe llevarse a cabo una estrategia de automatización de pruebas unitarias que permita llevar al cabo la comunicación entre un módulo web y el framework PHPUnit para llevar a cabo procesos de diseño, ejecución y análisis de resultados automatizados.

A partir de los resultados arrojados por los estudios y experimentos controlados en la empresa Visual Systems de Colombia se decide que aunque las pruebas unitarias tienen un enfoque de caja blanca y caja negra, las primeras resultan demasiado subjetivas para el entorno y equipo de desarrollo por lo cual se puede abordar determinando un estándar de codificación que permita llevar a cabo la generalización de estructuras de códigos PHP a probar.

PHP Testing Studio se desarrollo con los lineamientos de la estrategia de diseño planteada en este artículo para automatizar las pruebas unitarias de códigos PHP y será sometido a procesos de validación para su posterior uso en una desarrolladora de software interesada en implementarla.

REFERENCIAS

[1] PIATTINI M.G., CALVO J.A., CERVERA, J., FERNANDEZ, L. Análisis y diseño de aplicaciones informáticas de gestión, una perspectiva de ingeniería del software. México: Alfaomega Grupo Editor; 2004. 419-469 p.

[2] MESZAROS, Gerard. PHPUnit. xUnit Patterns. Canadá. 2008. p. 1. Disponible en: [http://xunitpatterns.com/PHPUnit.html]

[3] GOMEZ D. No tener pruebas unitarias automatizadas es irracional. [en línea]. DosIdeas. Nov., 2009. [citado 14 de abr. 2011]. Disponible en [http://www.dosideas.com/noticias/desarrollo-de-software/759-no-tener-pruebas-unitarias-automatizadas-es-irracional.pdf]

[4] J.M. Pruebas unitarias con CPPUnit. [en línea]. LWDEJM. Ago., 2005. [citado 13 de abr. 2011]. Disponible en: [http://www.wikilearning.com/tutorial/pruebas_unitarias_con_cppunit/3855-3]

[5] ADRFORMACION. Curso de Visual Basic.NET 2010. [en línea]. Adrformacion. [La Rioja, España], ene. 2011. [citado 11 de abr. 2011]. Disponible en: [http://www.adrformacion.com/cursos/vbnet2010/leccion1/tutorial9.html]

[6] BERGMANN, S. PHPUnit Official Manual. Git Hub Inc, Alemania. 2010.

[7] CERI S. FRATERNALI P., BONGIO A., BRAMBILLA M., COMAI S., MATERA M. Designing Data-Intensive WEB Applications. 3 ed. San Francisco: Morgan Kauffman Publishers; 2003. 332 p.