ESTRATEGIA DE DISEÑO PARA LA AUTOMATIZACIÓN DE PRUEBAS UNITARIAS DE CÓDIGOS PHP UTILIZANDO EL FRAMEWORK PHPUNIT

Alejandro Villa Betancur [1], Jorge E. Giraldo Plaza [2]

[1]Estudiante de ingeniería Informática, Politécnico Colombiano Jaime Isaza Cadavid. Carrera 48 Nº 7 – 151 Medellín, Colombia. alejo227@yahoo.com.
[2]Ingeniero de Sistemas, M.Sc. Profesor Asistente, Facultad de Ingenierías, Politécnico Colombiano Jaime Isaza Cadavid. Carrera 48 Nº 7- 151 Medellín, Colombia. jegiraldo@elpoli.edu.co.

RESUMEN

Las soluciones para pruebas unitarias de software basadas en librerías no son suficientes debido que para su uso es necesario su automatización, lo que genera doble trabajo para el programador. En este documento se presenta el desarrollo de una estrategia de diseño para automatización de pruebas unitarias de códigos PHP utilizando el *framework PHPUnit* como trabajo de grado en el Politécnico Colombiano Jaime Isaza Cadavid.

Palabras clave: Pruebas Unitarias, Automatización de Pruebas, PHP, *PHPUnit*, Estrategias de pruebas.

Recibido: 15 de Abril de 2011. Aceptado: 16 de Diciembre de 2011
Received: April 15th, 2011. Accepted: December 16th, 2011

DESIGN STRATEGY FOR UNIT TESTS AUTOMATION OF PHP CODES USING PHPUNIT FRAMEWORK

ABSTRACT

The solutions for software unit testing libraries are not based on sufficient due to its use are necessary for its automation, which creates double work for the programmer. This paper presents the development of a design strategy for automated unit testing PHP code using PHPUnit framework as graduate work at the Politécnico Colombiano Jaime Isaza Cadavid.

Keywords: *Unit tests, Test automation, PHP, PHPUnit, Test Strategies.*

1. INTRODUCCIÓN

Las pruebas unitarias de software permiten evaluar por separado el correcto funcionamiento de los códigos que componen los módulos del sistema. En la actualidad, existen diferentes *librerías* para automatizar las pruebas unitarias, pero no son fáciles de implementar ni resultan amigables para el desarrollador quien finalmente desiste del proceso para evitar mayores esfuerzos.

Por lo anterior es necesario contar con un componente intermedio entre las librerías y el código a probar. De esta manera la ejecución de la pruebas se agilizaría podría dar paso a realizar un análisis de sus resultados. Sin embargo es necesario realizar las pruebas de una manera estructurada y secuencialmente lógica que permita se realice una prueba de la manera apropiada; a esto se le conoce como Estrategia de Pruebas.

Existen dos enfoques principales para el diseño de pruebas unitarias, el primero es el de caja blanca orientado a la estructura del código y el segundo enfoque conocido como de caja negra, orientado al correcto funcionamiento del software a partir del análisis de entradas y salidas que posee y verificando que el resultado es el esperado [1]. Las pruebas unitarias de caja blanca 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, este enfoque no será automatizado en el proceso [3].

Para la automatización de la estrategia de diseño de pruebas se desarrolló una herramienta llamada *PHP Testing Studio*. Para el correcto funcionamiento de la herramienta se recomienda entonces hacer uso del estándar de codificación propuesto. 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 dando como resultado mejoras en los procesos de codificación a partir del momento de su implementación.

El artículo tiene la siguiente estructura: en la Sección 2 se presenta el estándar de codificación propuesto, de importancia en el correcto funcionamiento de la herramienta, a continuación en la sección 3 se expone la estrategia de diseño de pruebas unitarias definida, explicando detalladamente cada uno de sus pasos, posteriormente en la sección 4 se presenta la aquitectura computacional definida para el desarrollo de *PHP Testing Studio*; finalmente se presentan las conclusiones y trabajo futuro, así como las referencias bibliográficas.

2. ESTÁNDAR DE CODIFICACIÓN

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 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.

3. ESTRATEGIA DE DISEÑO PROPUESTA

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 dichas funciones que componen el código cumplen o no con el comportamiento esperado. A través de esta estrategia de automatización dicho proceso será automatizado y se apoyará en el framework de pruebas unitarias PHPUnit [4,5].

Tabla 1. Estándar de codificación PHP

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.

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. A continuación se explican los distintos pasos en detalle.

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 [4]. 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.

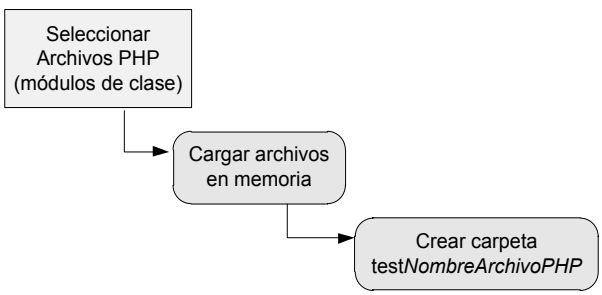

Fig.1. Selección de módulos.

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.

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 3.

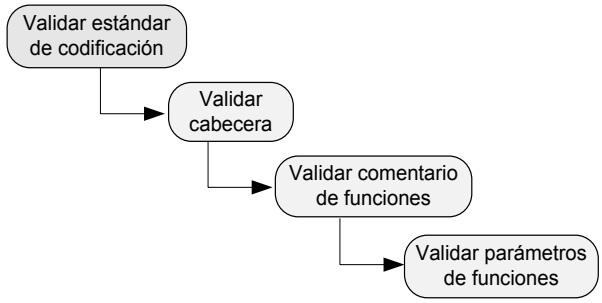

Fig.2. Selección de módulos.

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 [5]. Ver Figura 4.

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.

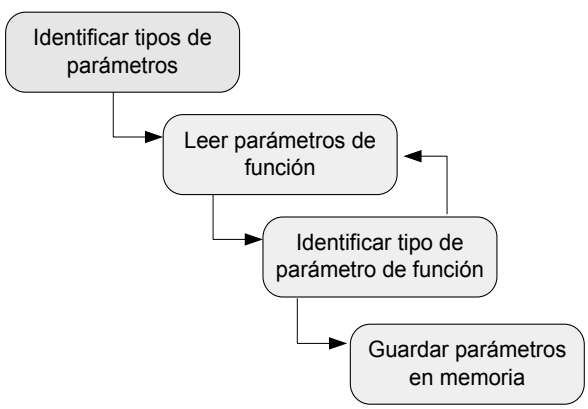

Fig.3. Identificar Tipos de Parámetros

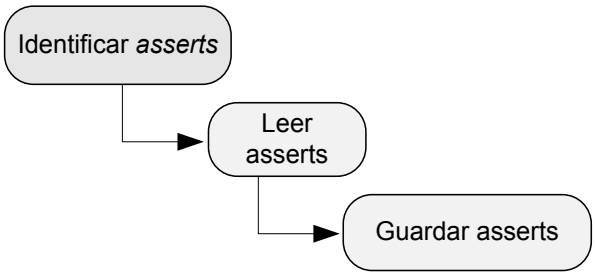

Fig.4. Identificar Asserts

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.

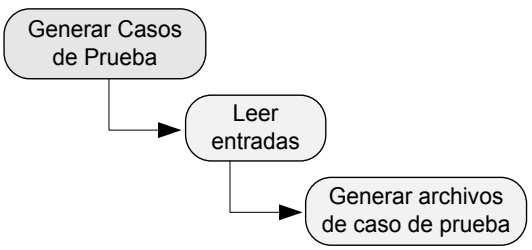

Fig.5. Generar Casos de Prueba

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.

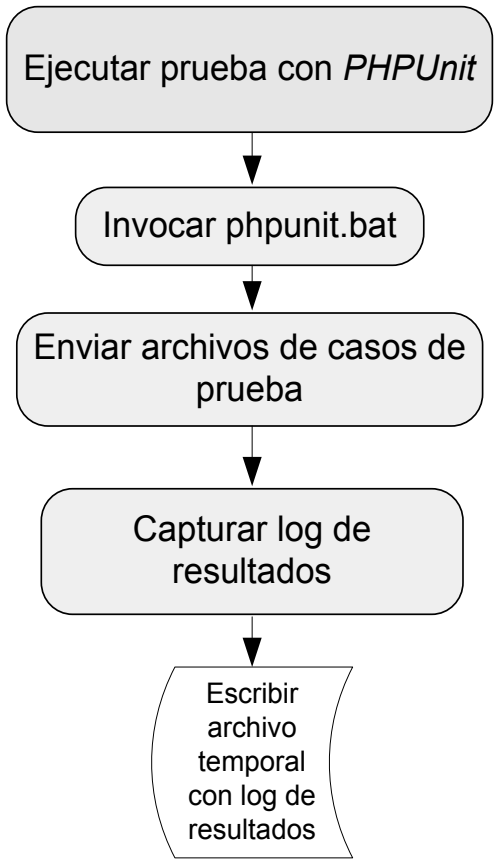

Fig.6. Ejecución de Pruebas.

PASO 8: Preparar Resultados: Por cada una de las pruebas ejecutadas se produce un reporte de los resultados, el cual contiene la siguiente información:

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.

4. ARQUITECTURA DEFINIDA

Para la implementación de la estrategia de diseño de pruebas unitarias se propone el desarrollo de un modelo web llamado PHP Testing Studio, el cual esta soportado por la siguiente infraestructura tecnológica. Ver Figura 7.

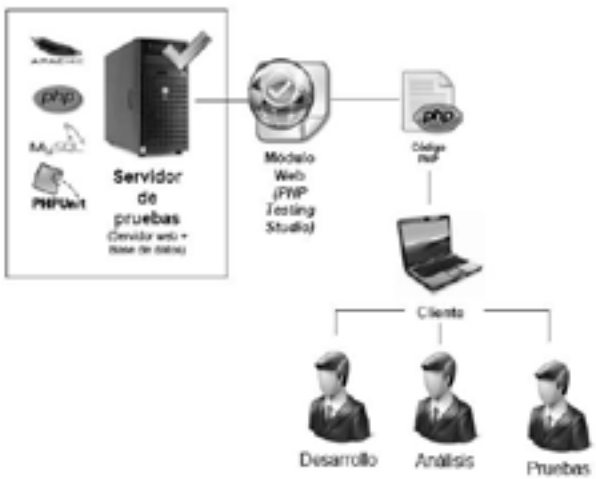

Fig.5. Infraestructura propuesta

La arquitectura tecnológica propuesta consta de una interfaz web (PHP Testing Studio), el cual tiene como cliente principal personal del área de desarroolo, de diseño y análisis o del propio analista de pruebas, sin embargo se espera que el aporte sea para quien programa, es decir, para el área de desarrollo.

Una vez PHP Testing Studio realiza la prueba, se reporta el resultado al usuario y a la vez se registra en un repositorio, para que posteriormente se analicen los resultados y rendimiento de los programadores, así como el diseño de futuros casos de prueba.

5. CONCLUSIONES Y TRABAJO FUTURO

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 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 desarrolló bajo 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 la industria.

6. AGRADECIMIENTOS

Este artículo es resultado del proyecto “Desarrollo de un módulo Web para la automatización de pruebas unitarias de códigos PHP utilizando el *framework PHPUnit*”. Los autores agradecen a la institución y al comité de proyectos del Politécnico Jaime Isaza Cadavid que con sus valiosos aportes contribuyeron a la continua mejora de este proyecto.

7. REFERENCIAS BIBLIOGRÁFICAS

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