Refactoring: avanzando con las soluciones

En la edición anterior de Hackers & Developers Magazine hicimos una introducción a la técnica de *Refactoring* y planteamos soluciones a problemas cotidianos. En esta entrega final, veremos problemas más complejos que también pueden ser solucionados implementando la técnica de *Refactoring*.

Escrito por: **Eugenia Bahit** (GLAMP Hacker & eXtreme Programmer)

Eugenia es **Arquitecta de Software**, docente e instructora de tecnologías **GLAMP** (GNU/Linux, Apache, MySQL, Python y PHP) y **Agile coach** (UTN) especializada en Scrum y **eXtreme Programming**. Miembro de la **Free Software Foundation**, **The Linux Foundation** y **Debian Hackers**. Creadora de python-printr, Europio Engine y colaboradora de Vim.

Webs:
Cursos de programación: www.cursosdeprogramacionadistancia.com
Web personal: www.eugeniabahit.com

Redes sociales:
Twitter / Identi.ca: **@eugeniabahit**

En la edición anterior, comentábamos la diferencia entre problema y error, explicando que la técnica de *Refactoring* había sido pensada para solucionar problemas pero no, para corregir errores.

Los problemas están muy ligados a la legibilidad del código y su reutilización. Por lo tanto, el *Refactoring* tiene por objetivo, hacer el código más legible y reutilizable.

En esta entrega nos enfocaremos en problemas de legibilidad y reutilización, algo más complejos que los que vimos en el capítulo anterior.

PROBLEMA: Métodos extensos

No solo una expresión puede ser extensa. Muchas veces, nos encontraremos con métodos con extensos algoritmos que realizan varias acciones:

```
function suscribir_al_newsletter() {
    if(isset($_POST['email'])) {
        $email = $_POST['email'];
    } else {
        $email = '';
    }

    if(isset($_POST['nombre'])) {
        $nombre = $_POST['nombre'];
    } else {
        $nombre = '';
    }

    if($email == '') {
        $errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($email) < 7) {
        $errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $email = strtolower(strip_tags($email));
    }

    if(strlen($nombre) < 5) {
        $errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $nombre = strtoupper(strip_tags($nombre));
    }

    $id = 0;
    $msg = 'Error desconocido';

    if(isset($errors)) {
        $msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$nombre}", "{$email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) {
        $msg = 'Se ha suscrito al newsletter';
    }

    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $msg, $plantilla);
}
```

Cuando existen métodos tan extensos, probablemente, la solución consista en la combinación de diversas técnicas, que van desde agrupar expresiones en una misma línea hasta evitar la asignación de variables temporales (como vimos la edición pasada) y extraer código llevándolo a diferentes métodos. En estos casos, donde más de una técnica puede ser necesaria, un buen comienzo será detectar la cantidad de acciones y responsabilidades que el método o función presentan actualmente.

Si observamos el código anterior, podremos encontrar las siguientes acciones y responsabilidades:

1. Recibir dos datos desde un formulario asignando dicho valor a dos variables;
2. Validar que los datos recibidos desde el formulario sean correctos, retornando un mensaje de error en caso contrario;

3. Filtrar los datos recibidos desde el formulario;
4. Guardar los datos en la base de datos;
5. Mostrar un mensaje en pantalla.

No hace falta demasiado para ver que una sola función, está realizando el trabajo de 5 funciones. Es entonces, cuando procedemos a extraer el código y crear nuevos métodos. Para ello, primero pasamos los bloques de códigos (agrupados por responsabilidad) a nuevos métodos:

```
function recibir_datos() {
    if(isset($_POST['email'])) {
        $email = $_POST['email'];
    } else {
        $email = '';
    }

    if(isset($_POST['nombre'])) {
        $nombre = $_POST['nombre'];
    } else {
        $nombre = '';
    }
}

function validar_datos() {
    if($email == '') {
        $errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($email) < 7) {
        $errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $email = strtolower(strip_tags($email));
    }

    if(strlen($nombre) < 5) {
        $errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $nombre = strtoupper(strip_tags($nombre));
    }
}

function guardar_datos() {
    $id = 0;
    $msg = 'Error desconocido';

    if(isset($errors)) {
        $msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$nombre}", "{$email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) {
        $msg = 'Se ha suscrito al newsletter';
    }
}

function mostrar_mensaje() {
    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $msg, $plantilla);
}
```

```
function suscribir_al_newsletter() {
    recibir_datos();
    validar_datos();
    guardar_datos();
    mostrar_mensaje();
}
```

Acto seguido, verificamos aquellas variables de uso temporal que hayan quedado definidas pero que sean requeridas por más de una función y las convertimos en propiedades de clase:

```
function recibir_datos() {
    if(isset($_POST['email'])) {
        $this->email = $_POST['email'];
    } else {
        $this->email = '';
    }

    if(isset($_POST['nombre'])) {
        $this->nombre = $_POST['nombre'];
    } else {
        $this->nombre = '';
    }
}

function validar_datos() {
    if($this->email == '') {
        $this->errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($this->email) < 7) {
        $this->errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $this->email = strtolower(strip_tags($this->email));
    }

    if(strlen($this->nombre) < 5) {
        $this->errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $this->nombre = strtoupper(strip_tags($this->nombre));
    }
}

function guardar_datos() {
    $id = 0;
    $this->msg = 'Error desconocido';

    if(isset($this->errors)) {
        $this->msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$this->nombre}", "{$this->email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) {
        $this->msg = 'Se ha suscrito al newsletter';
    }
}

function mostrar_mensaje() {
```

```
    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $this->msg, $plantilla);
}

function suscribir_al_newsletter() {
    recibir_datos();
    validar_datos();
    guardar_datos();
    mostrar_mensaje();
}
```

Finalmente, aquellas instrucciones que puedan definirse en una sola línea, pueden ser *refactorizadas*:

```
function recibir_datos() {
    $this->email = isset($_POST['email']) ? $_POST['email'] : '';
    $this->nombre = isset($_POST['nombre']) ? $_POST['nombre'] : '';
}

function validar_datos() {
    if($this->email == '') {
        $this->errors[] = 'E-mail no puede estar vacío';
    elseif(strlen($this->email) < 7) {
        $this->errors[] = 'El e-mail no puede tener menos de 7 caracteres';
    } else {
        $this->email = strtolower(strip_tags($this->email));
    }

    if(strlen($this->nombre) < 5) {
        $this->errors[] = 'Por favor, indique un nombre y apellido legítimo';
    } else {
        $this->nombre = strtoupper(strip_tags($this->nombre));
    }
}

function guardar_datos() {
    $id = 0;
    $this->msg = 'Error desconocido';

    if(isset($this->errors)) {
        $this->msg = implode('<br/>', $errors);
    } else {
        $sql = 'INSERT INTO newsletter (nombre, email) VALUES (?, ?);
        $data = array('ss', "{$this->nombre}", "{$this->email}");
        $id = DBLayer::ejecutar();
    }

    if($id > 0) $this->msg = 'Se ha suscrito al newsletter';
}

function mostrar_mensaje() {
    $plantilla = file_get_contents('system_msg.html');
    print str_replace('{MSG}', $this->msg, $plantilla);
}

function suscribir_al_newsletter() {
    recibir_datos();
    validar_datos();
    guardar_datos();
    mostrar_mensaje();
```

```
}
```

PROBLEMA: Código duplicado en una misma clase

Es frecuente -y de lo más común-, que las mismas expresiones, comiencen a duplicarse en diferentes métodos de una misma clase:

```
def set_txt_encabezado(self):
    with open('plantilla.tmpl', 'r') as archivo:
        TEXTO = archivo.read()

    a = TEXTO.replace('X', self.destinatario)
    return "Estimado %s:<br/>" % a

def set_txt_certificacion(self):
    with open('plantilla.tmpl', 'r') as archivo:
        TEXTO = archivo.read()

    a = TEXTO.replace('X', self.destinatario)
    return "Certifica que el %s" % a
```

Las expresiones duplicadas en el código de los diferentes métodos de una misma clase, se solucionan extrayendo el código duplicado de los métodos y colocándolo en un nuevo método de clase:

```
def render(self):
    with open('plantilla.tmpl', 'r') as archivo:
        TEXTO = archivo.read()

    return TEXTO.replace('X', self.destinatario)


def set_txt_encabezado(self):
    return "Estimado %s:<br/>" % self.render()

def set_txt_certificacion(self):
    return "Certifica que el %s" % self.render()
```

PROBLEMA: Código duplicado en clases con la misma herencia

El caso anterior puede darse también, cuando el código se encuentra duplicado en diferentes métodos de clases con la misma herencia:

```
class Carta(Documento):

    #...

    def set_txt_encabezado(self):
        with open('plantilla.tmpl', 'r') as archivo:
```

```
            TEXTO = archivo.read()

        a = TEXTO.replace('X', self.destinatario)
        return "Estimado %s:<br/>" % a


class Certificado(Documento):

    #...

    def set_txt_certificacion(self):
        with open('plantilla.tmpl', 'r') as archivo:
            TEXTO = archivo.read()

        a = TEXTO.replace('X', self.destinatario)
        return "Certifica que el %s" % a
```

En estos casos, en los cuáles existen dos o más clases que heredan de la misma madre, se extrae el código duplicado en los métodos de las clases hijas y con éste, se crea un nuevo método en la clase madre:

```
class Documento(object):

    #...

    def render(self):
        with open('plantilla.tmpl', 'r') as archivo:
            TEXTO = archivo.read()

        return TEXTO.replace('X', self.destinatario)


class Carta(Documento):

    #...

    def set_txt_encabezado(self):
        return "Estimado %s:<br/>" % self.render()


class Certificado(Documento):

    #...

    def set_txt_certificacion(self):
        return "Certifica que el %s" % self.render()
```

PROBLEMA: Código duplicado en varias clases sin la misma herencia

Como era de esperarse, el código también podría aparecer duplicado en diferentes clases pero que no tienen la misma herencia:

```
class Carta {
```

```
    #...

    function set_txt_encabezado() {
        $texto = file_get_contents('plantilla.tmpl');
        $a = str_replace('X', $this->destinatario, $texto);
        return "Estimado $a:<br/>";
    }
}


class Certificado {

    #...

    function set_txt_certificacion() {
        $texto = file_get_contents('plantilla.tmpl');
        $a = str_replace('X', $this->destinatario, $texto);
        return "Certifica que el $a";
    }

}
```

En estos casos, la solución es extraer el código duplicado, crear una nueva clase y con el código extraído, crear un método para esta nueva clase que podrá ser heredada por las anteriores o simplemente, instanciada:

```
class Documento {

    # ...

    function render() {
        $texto = file_get_contents('plantilla.tmpl');
        return str_replace('X', $this->destinatario, $texto);
    }

}

class Carta extends Documento {

    #...

    function set_txt_encabezado() {
        return "Estimado {$this->render()}:<br/>";
    }
}


class Certificado extends Documento {

    #...

    function set_txt_certificacion() {
        return "Certifica que el {$this->render()}";
    }

}
```