Ingeniería Inversa: solo el "qué" puede desvelar el camino hacia el "cómo"

La Ingeniería Inversa aplicada al desarrollo de Software no es ni fácil ni difícil. Su sencillez puede resultar complicada o su complejidad puede resultar sencilla, dependiendo de cada persona. Por lo tanto, se desaconseja la implementación de las técnicas aquí expuestas, si al estudiar este paper se experimenta la sensación de estar "descubriendo errores", puesto que la misma es indicio de una comprensión incorrecta que podría llevar a la adquisición de malas prácticas.

Cuando la ingeniería inversa se aplica para el desarrollo de software, suele ser frecuente que el programador comience el diseño de la aplicación anticipando el cómo la desarrollará, sin haber evaluado previamente qué es lo que de sea lograr. Sin dudas, esto es **lo que más trabajo cuesta a los programadores: entender «el qué» y diferenciarlo del «cómo»**.

A diferencia de la Ingeniería Inversa de hardware, **la fórmula de la Ingeniería Inversa para el desarrollo de Software** no consiste únicamente en seguir una serie de pasos. Por el contrario, cada paso a seguir **consiste en**, paradójicamente, analizar y **deducir cuáles son -justamente- los pasos a seguir**.

> **La mejor forma de organizar el desarrollo es efectuar el análisis de requerimientos por medio de Historias de Usuarios**

Organizar el análisis de requerimientos mediante Historias de Usuarios, es el primer paso para comprender primero **qué es lo que se desea lograr**.

> *Si no se sabe con exactitud qué resultado se espera ver, no hay forma posible de hacer una evaluación que indique si vamos o no por el camino correcto.*

Una Historia de Usuario, en sí misma es un objetivo a alcanzar. Sin embargo, **el programador confunde el objetivo en sí mismo, con el cómo ese objetivo debe ser alcanzado**. Pero para entenderlo mejor, veámoslo con un ejemplo.

Supongamos que tenemos la siguiente Historia de Usuario:

«Como administrador puedo agregar nuevas categorías al catálogo de productos»

No interesa si se está programando orientado a objetos o no o si es mejor implementar un paradigma u otro. Lo que nos interesa ahora es entender cuál es el objetivo. Y a diferencia de lo que la mayoría de los programadores creería, el único objetivo que se encuentra inicialmente es **«insertar nuevas categorías a un catálogo de productos»** el cual se supone implícitamente, que ya existe.

Entonces, lo que se espera ver es **una nueva categoría dentro del sitio donde el mecanismo de persistencia elegido** (una base de datos, por ejemplo) **almacene la información**.

Teniendo en claro qué es exactamente lo que se espera ver, recién allí es cuándo se debe comenzar a pensar en cómo lograrlo.

Esto necesariamente implica tener que **convertir un objetivo en el procedimiento necesario para alcanzar el siguiente objetivo**. De eso se trata la ingeniería inversa. Cada objetivo debe ser a la vez, convertido en su propio cómo y este último, debe arrojar el siguiente objetivo.

Por ejemplo, si insertar categorías es un objetivo, el cómo nos lo tiene que dar el propio objetivo. ¿Qué significa esto? Qué lo primero que debemos hacer es insertar esas categorías tan manualmente como sea posible. Esto es "transformar el objetivo en un cómo alcanzarlo". Suponiendo que trabajamos proceduralmente (a fin de no enredar a los menos expertos con cuestiones inherentes a la programación orientada a objetos) y la base de datos fuese nuestro mecanismo de persistencia, necesitaríamos una función que guardase una categoría en la base de datos.

Cuando te cueste hallar "el qué", escribe un caso de uso. Eso te ayudará a encontrarlo más fácilmente.

Por ejemplo, podemos citar como caso de uso, el administrador que agrega la categoría llamada «Electrónica». Entonces ¿qué debemos hacer? Escribir el mínimo código necesario para insertar la categoría «Electrónica» en la base de datos (recordemos que el objetivo fue transformado en el cómo alcanzarlo):

```
function agregar_categoria() {
    # Definir el query
    $sql = "INSERT INTO categorias (categoria_id, denominacion) VALUES (NULL, ?)";
    # Asignamos los datos manualmente (hardcodeamos)
```

```
        $data = array("s", "Electrónica");
        # Ejecutamos la consulta
        $categoria_id = DBLayer::ejecutar($sql, $data);
        # retornamos el resultado de la ejecución a fin de constatar que la misma haya sido efectiva
        return $categoria_id;
}


# Invocamos a la función y probamos que funcione, es decir, corroboramos que se agregue la denominación Electrónica
# en la tabla categorias
print agregar_categoria();
```

Como se puede observar, simplemente nos limitamos a:

1. **Identificar el primer objetivo:** agregar una nueva categoría;
2. **Alcanzar el objetivo:** convertimos el objetivo en una acción concreta que nos dice como alcanzar el objetivo (insertar un nuevo registro en la tabla categorías)

Una vez hecho esto, se verifica que funciona. Si falla, se corrige tantas veces como sea necesario hasta lograr que funcione.

Nunca intentes adivinar un error. Utiliza el log de errores para saber por qué ha fallado algo.

Ahora, se busca un nuevo objetivo. Antes, habíamos dicho que "el cómo de un objetivo también debería ser quien nos arroje el siguiente objetivo". **Siempre después de *hardcodear* algo, el objetivo siguiente es elegir una parte del código para dejar de *hardcodear***. El ciclo se repite hasta que ya no quede nada que necesite dejar de ser *hardcodeado*. En nuestro caso, estamos insertando a mano, la denominación de la categoría desde el propio código. Nuestro objetivo debería estar relacionado a "dejar de insertar a mano la denominación de la categoría".

Se debe ser sumamente cuidadoso ya que aquí suelen aparecer gran parte de las complicaciones. Obviamente nuestro objetivo es hacer que el valor del campo `denominacion` sea variable (por favor, notar que cito como objetivo que el campo SEA VARIABLE pero en ningún momento menciono algo como "que valor sea ingresado mediante un formulario", es decir, no hablo de cómo lograrlo ni mucho menos, me planteo objetivos ambiciosos con respecto a lo que ya he alcanzado).

El cómo lograrlo, puede variar de programador en programador y cada uno tendrá ideas diferentes. Pero siempre debe establecerse como regla principal, **escribir el mínimo código necesario**.

En este caso puntual, se me ocurre una sola alternativa: para que un valor sea variable, lo defino en una variable (recurro a lo más simple, es decir, busco lo obvio que en definitiva no es más que convertir al objetivo en un cómo). Ahora mi objetivo, es definir el valor de una categoría en una variable (el cómo se transforma en el siguiente objetivo). Y se me ocurren -en principio- dos formas de lograrlo:

1. Definir la variable dentro de la función;

2. Pasar el valor como parámetro;

Frente a dos o más alternativas, se debe aplicar siempre el principio de la Navaja de Ockham, el cual dice que: **la solución más simple suele ser la más probable** (aunque no necesariamente la correcta). Para ello, debo analizar y comparar las alternativas.

Si definiera la variable dentro de la función y quisiera, por ejemplo, insertar 3 categorías, debería editar el archivo 3 veces y ejecutarlo 3 veces, mientras que pasando el valor como parámetro, podría invocar a la función 3 veces. Como esta última requiere menor esfuerzo (y el esfuerzo refleja la complejidad de algo), la elijo:

```
function agregar_categoria($denominacion) {
        $sql = "INSERT INTO categorias (categoria_id, denominacion) VALUES (NULL, ?)";
        $data = array("s", "$denominacion");
        $categoria_id = DBLayer::ejecutar($sql, $data);
        return $categoria_id;
}


print agregar_categoria('Juguetes');
print agregar_categoria('Ferretería');
print agregar_categoria('Bazar');
```

A pesar de todo, sigo *hardcodeando* los datos (aunque ya un poco menos que antes). Ahora, lo que debo lograr es que el valor de la categoría, no sea escrito manualmente en el archivo. La única forma de lograr mi nuevo objetivo, es hacer que el usuario lo ingrese. Seguramente, esto sea uno de los criterios de aceptación de la HU. Incluso, hasta debe mencionar que deberá hacerse mediante un formulario. Pero aún, no he llegado a los criterios de aceptación, así que omitiré pensar en ellos (incluso cuando los conozca).

Omite pensar en aquellas restricciones a las que aún no haz llegado. Deja que el código te guíe. De lo contrario, no estarás implementando la ingeniería inversa sino la tradicional.

Para que el usuario ingrese dicho valor podría hacerlo mediante dos métodos (POST o GET). Sé que POST es el más seguro pero implicaría mucho esfuerzo: crear un formulario HTML y la función que lo muestre o crear una función utilizando cURL para enviar los datos, lo cual implicaría escribirlos a mano nuevamente o enviarlos mediante GET, recogerlos y enviarlos con cURL mediante POST. DEMASIADO complicado. Pasarlo por GET (y listo) resulta mucho más sencillo. Y como bien dijimos, no debemos (ahora) pensar en restricciones futuras. Aún no hemos llegado a la seguridad de la aplicación ni a los criterios de aceptación, pues entonces, adelante con utilizar el método GET.
La función no cambiará; solo lo hará la forma en la que ésta será invocada:

```
print agregar_categoria($_GET['categoria']);
```

Vuelvo a probar, simplemente modificando el valor del parámetro en la URI tantas veces como lo desee:

```
.....categorias.php?categoria=Indumentaria
.....categorias.php?categoria=Deportes
.....categorias.php?categoria=Jardinería
.....categorias.php?categoria=Farmacia
```

Una vez que ya se ha eliminado todo dato *hardcodeado*, puedo pasar a los criterios de aceptación.

Por favor, notar que hasta aquí, hemos comenzado el desarrollo por el final, es decir, por cumplir el primer objetivo el cual es la Historia de Usuario en sí mismo. Esto, justamente, es aplicar la Ingeniería Inversa.

Entre los criterios de aceptación de esta Historia de Usuario, por lo general, suelen encontrarse varios que hacen referencia a las restricciones sobre los datos a agregar. Desde algunos que mencionan cantidades mínimas y máximas de caracteres hasta otros que nos dicen qué hacer si se intentan incluir datos duplicados.

Ten en cuenta que cada criterio de aceptación será un objetivo

Esto significa que debemos tomar uno -y solo uno- incluso aunque parezca "poco". Por ejemplo, imaginemos que entre los criterios de aceptación encontramos los siguientes:

- *Categoría con menos de 3 caracteres retorna mensaje "El nombre de la categoría debe tener al menos 3 caracteres sin incluir los espacios en blanco"*
- *Categoría con más de 30 caracteres retorna mensaje "El nombre de la categoría no puede contener más de 30 caracteres incluyendo espacios en blanco"*
- *etc....*

Ejemplo de Criterios de Aceptación

Lógicamente habrá muchos criterios de aceptación más, pero **se debe elegir solo uno** de ellos -sin excepción- para continuar el desarrollo y sobre el elegido, se repetirán todos los procedimientos anteriores.

Para que no queden dudas sobre cómo aplicar Ingeniería Inversa al desarrollo de Software, propongo un ejercicio sencillo que a mis alumnos suele ayudarlos a esclarecer toda duda. Se trata de desarrollar una librería (o *script*) que retorne una cadena de texto aleatoria (a modo de contraseña) con las siguientes características:

- Debe tener al menos 1 letra mayúscula, 1 minúscula, 1 número y 1 carácter no alfanumérico

- Su longitud debe ser de 8 caracteres
- El orden de aparición de los caracteres debe ser aleatorio (es decir que no puede cumplir un patrón)

Como única ayuda diré que:

- Se debe comenzar *hardcodeando* el resultado. Por ejemplo, comenzar retornando una contraseña a modo de ejemplo, que cumpla con los requisitos anteriores: **aA1#A1a=**

- Del retorno anterior, ir sustituyendo los caracteres *hardcodeados* de a uno por vez respetando el patrón del retorno inicial y dejando para el final, la "mezcla" de caracteres para evitar patrones.

Si sigues los pasos mencionados y no te dejas llevar por suposiciones ni por información que no esté explícita, lograrás entender por completo el procedimiento para desarrollar Software aplicando Ingeniería Inversa.

Contratando un VPS con el enlace de esta publicidad, me ayudas a mantener The Original Hacker :)

Servidores a solo **USD 5 / mes**:

- **20 GB** de disco
- Discos **SSD**
- 1 TB de transferencia
- **512** MB **RAM**
- **Instalación en menos de 1'**

Elige **Ubuntu Server 12.04 LTS** y despreocúpate de la seguridad, optimizándolo con **JackTheStripper**

Luego de instalarlo, **configúralo con JackTheStripper**: eugeniabahit.com/proyectos/jackthestripper

Contratando con este enlace, me ayudas a mantener The Original Hacker: http://bit.ly/promo-digitalocean