Desarrollar Software Aplicando la Ingeniería Inversa: el arte de la ciencia

Desarrollar Software puede convertirse en un verdadero "arte científico" si se es capaz de combinar paralelamente, lógica y pensamiento lateral. Y eso, es lo que la ingeniería inversa nos propone. En esta edición de The Original Hacker, veremos en qué consiste la ingeniería inversa de Software aplicada durante su desarrollo.

Si alguna vez te viste envuelto en una maraña de ideas y procedimientos al momento de desarrollar una determinada funcionalidad y terminaste enredándote tanto que decidiste "dibujar" lo que necesitabas alcanzar y desde allí, fuiste sustituyendo tu "dibujo" por código hasta terminarlo, lo que hiciste, fue aplicar un procedimiento de ingeniería inversa para resolver ese requerimiento que tanto te enredaba.

A la ingeniería inversa muchas veces se le suele tener miedo y esto sucede -creo yo- por la inmensa cantidad de mitos alrededor de este maravilloso concepto pero por sobre todo -y en definitiva-, por la gran ignorancia y desconocimiento sobre su existencia.

> *La ingeniería inversa plantea una forma lateral de alcanzar objetivos, partiendo del resultado -el "qué"- para obtener el modo de alcanzarlo -el "cómo"-*

La ingeniería inversa es amplia. Es amplia como ciencia y como técnica pero también como concepto, ya que si bien posee un único significado, sus aplicaciones son tan variadas y disímiles que es allí donde se produce la distorsión conceptual que tantos miedos genera.

Al proponer el inicio de los procesos partiendo del objeto final, es que comúnmente se la relaciona con una de sus aplicaciones más frecuentes:

- Ingeniería Inversa de Hardware: aquella que, partiendo de un componente físico (hardware) cuyo

código es privativo (no libre, no abierto), intenta obtener un software que permita reemplazar al controlador del fabricante.

Sin embargo, no es únicamente en esta área donde la ingeniería inversa se aplica. La ingeniería inversa tiene muchísimas más aplicaciones, entre ellas:

- Ingeniería Inversa de código: aquella que va "siguiendo la pista" del código fuente de una aplicación ya sea tanto para entender la forma en la que ésta funciona como para detectar fehacientemente vulnerabilidades que no podrían ser halladas mediante herramientas para pruebas de intrusión automatizadas (que tan de moda se han puesto en los últimos tiempos pero que sin embargo, no han sido capaces hasta el momento, de sustituir ni reemplazar la inteligencia humana);

- Ingeniería Inversa de Software: ya sea para descomposición de aplicaciones (generalmente, aplicada para desentrañar un Software cerrado y privativo) o para el caso contrario, la composición de Software, a la cual nos dedicaremos en este artículo.

TDD: El ejemplo perfecto de Ingeniería Inversa de Software aplicada al desarrollo

La técnica extrema TDD *(Test-Driven Development)*, que en español se traduce como **Desarrollo guiado por pruebas**, es un claro ejemplo de cómo la Ingeniería Inversa aplicada al desarrollo de Software logra lógicas realmente simples de comprender y prácticamente unívocas e inequívocas.

Muy frecuentemente, confundimos Unit Testing (pruebas unitarias) con Test-Driven Development y Test-First Programming. Estos errores de concepto pueden generar que jamás se logre entender con exactitud, qué es la Ingeniería Inversa, cómo se aplica y para qué es necesario implementarla. Por ello, me gustaría tratar de explicar los tres conceptos con la esperanza de que por fin, queden claramente diferenciados entre sí.

En principio, podemos decir que Unit Testing, Test-First Programming y TDD, son tres técnicas donde "de menor a mayor" una es requerida por la otra. Es decir que TDD necesita de Test-First Programming y éste, de Unit Testing. Es por ello, que comenzaremos definiéndolos de "menor a mayor".

Unit Testing

Las pruebas unitarias simplemente son funciones destinadas a evaluar una única acción del código fuente. Es así que si en nuestra aplicación tenemos una función destinada a eliminar etiquetas HTML de una cadena de texto, podremos tener, en paralelo 5 métodos por ejemplo, que pasen 5 cadenas de texto diferentes a nuestra función y comprueben que el valor de retorno sea efectivamente el esperado.

Las pruebas unitarias como tales, no requieren de Ingeniería Inversa y de hecho, hasta pueden ser generadas de forma automatizada, por lo que de forma aislada, no pueden considerarse como una técnica extrema. Sin ir más lejos, las pruebas unitarias podrían escribirse una vez que la aplicación se encontrase ya desarrollada.

Test-First Programming

Esta es una técnica que consiste en escribir primero las pruebas unitarias y luego el código de la aplicación.

Esta técnica ya es un poco más compleja que el mero hecho de escribir pruebas unitarias pero sin embargo, no alcanza a ser una técnica de Ingeniería Inversa, puesto que podría facilitarse tranquilamente, el diseño de lo que se quiere lograr (es decir, "el cómo") antes de contar exactamente con el objeto (es decir, "el qué").

TDD – Test-Driven Development

El desarrollo guiado por pruebas, es pura y exclusivamente una técnica extrema de Ingeniería Inversa que plantea pensar primero en lo que se quiere lograr (es decir "el qué") y partiendo de pseudo-prototipos (que a veces no son más que el resultado *hardcodeado* de un ejemplo de lo que se desea lograr) ir implementando el mínimo código necesario para que dicho pseudo-prototipo deje de serlo. Y para escribir el mínimo código necesario, propone utilizar Test-First Programming cómo técnica para la implementación de las pruebas unitarias.

¿Por qué decimos que la técnica de TDD es Ingeniería Inversa pura?

Porque para que ésta esté bien implementada, se debe partir el desarrollo desde lo que se quiere lograr para alcanzar de a poco el cómo lograrlo.

Comprendiendo "El qué": comenzar por el resultado

A lo largo de los años he notado que lo que más le cuesta entender a todas las personas es qué es exactamente a lo que nos referimos cuando hablamos de "resultado", es decir, "qué es exactamente el qué". Y de verdad, que de tan simple que me resulta, a veces creo que esa obviedad es la que lo hace tan difícil de entender como quizás también de explicar. **El resultado es exactamente lo que esperas ver cuando hayas terminado tu aplicación y la utilices por primera vez.**

Paso a paso, un ejemplo muy sencillo y de fácil comprensión sobre cómo se implementaría la Ingeniería Inversa en el desarrollo de Software, podría ser el siguiente:

1. **Se identifica lo que se quiere alcanzar:**
   por ejemplo, un generador de formularios de contacto;

2. **Se identifica aquello que producirá nuestro Software terminado:**
   por ejemplo, un formulario HTML;

3. **Se crea un prototipo del producto de nuestra aplicación:**
   es decir, se crea un archivo HTML con un formulario de contacto a modo de ejemplo;

4. **Se implementa el mínimo código necesario para que ese formulario sea retornado:**
   por ejemplo, desde el lenguaje de programación en el que se vaya a desarrollar la aplicación, se procedería a crear una función que lea el contenido del archivo HTML y lo retorne. A continuación, una instrucción, llamaría a dicha función imprimiendo en pantalla el valor de retorno;

5. **Se procedería a repetir el paso 4, de a un ítem por vez:**
   esto se hará, hasta lograr que el formulario HTML sea generado de forma 100% dinámica;

Cuando además, se implementa **TDD** como técnica para lograr el paso 4, **la diferencia con la guía anterior, es que tras el paso 3, se aplica *Test-First Programming*** para lograr el paso 4.

Troubleshooting: Ingeniería Inversa

Este subtítulo fue puesto ex profeso a fin de que a través del humor - considerando a la Ingeniería Inversa "un problema a resolver" - se puedan plantear formas amenas y "humanas" de llevar a cabo tal técnica.

Problema: se confunde el qué con el cómo

Descripción del problema:
Generalmente, el primer problema al que uno se enfrenta es saber identificar el qué sin pensar en el cómo. Y esto, se soluciona solo y únicamente con un cambio radical de actitud.

Objetivo:

- Perderle el miedo a lo desconocido;
- Aprender a convivir con la incertidumbre y aceptar que el cómo llegará a su debido momento;
- Aceptar que lo simple es preferible a lo complejo y lo complejo es preferible a lo complicado;
- Hacerse a la idea de que todo es posible y que alcanzar objetivos no es un problema: el problema es identificarlos;

Solución:

1. Definir lo que se quiere lograr a modo de Historia de Usuario: como [usuario] puedo [acción];
2. Pensar en la acción de la Historia de Usuario como en una aplicación completa (un todo);
3. Hacerse la pregunta ¿qué voy a obtener cuando la funcionalidad esté operativa? Y responder a ella con una sola frase corta utilizando la menor cantidad de calificativos y de verbos posibles. Lo ideal es centrarse en los sustantivos.

Problema: se tiende a pensar en "el cómo" como un todo y en vez de comenzar el desarrollo por el final se comienza por el principio

Descripción del problema:
Otro de los problemas más frecuentes es que una vez hallado "el qué" no se logra implementar un desarrollo inverso y por el contrario, se comienza a escribir el código de la manera que se lo haría tradicionalmente. Esto genera un estancamiento en el desarrollo y consecuente falta de prolijidad en el código generado.

Objetivo:

- Aprender a pensar lateralmente;
- Implementar la lógica y el pensamiento lateral en paralelo para aplicar la Ingeniería Inversa de forma correcta.

Solución:

- Mientras que en pasos anteriores se definió de forma lineal lo que se pretendía lograr, ahora es momento de razonar de manera lógica para obtener el siguiente paso (como si se tratase de una secuencia a la que se quiere dar continuidad). Es decir, el siguiente paso consiste en escribir el mínimo código necesario para sustituir "algo". Ese "algo" es el que debe estar desencadenado por una secuencia lógica;
- Para encontrar dicha secuencia, una gran parte de las veces, puede ser necesario mirar "el problema" desde diversos ángulos, es decir, analizar lo que se tiene de forma lateral;
- Hallada la secuencia, se debe elegir "uno y solo un" punto de partida y concentrarse solo y únicamente en escribir el código que resuelva ese -y no otro- ítem, evitando pensar en otros ítem o en lo que se necesitará resolver más adelante.

Es importante comprender que la Ingeniería Inversa demandará hacer un uso continuo del Refactoring y tomar la no-necesidad de refactorización del código como un error de diseño prácticamente asegurado.