ESCOM - IPN

Reingeniería y Reutilización

Análisis y Diseño Orientado a Objetos

Oscar Andrés Rosas Hernandez

Mayo 2018

Índice

1. Reautilización

1.1. Introducción

La reutilización es uno de los conceptos de la programación más antiguos y simples de entender, pero complejo al poner en práctica. El uso de software existente para desarrollar uno nuevo ha sido empleado desde los primeros días de la programación, siempre se han reusado partes de un código, planillas, funciones o procedimientos. Esta es la idea: parte o todo el código de un programa de computadora escrito una vez, sea o pueda ser usado en otros programas.

En sí, consiste en la creación de sistemas de software a partir de un software existente, en lugar de tener que rediseñar desde el principio.

En el siglo pasado resultaba imposible poder representar artefactos de software por su propio contenido (por ejemplo, buscar diagramas de clase similares a uno dibujado en otra herramienta), lo cual impedía que cualquiera pudiera buscarlos sin conocimiento previo de su existencia. Por este motivo, para reutilizar hubo que realizar el proceso inverso: primero parar la organización para ver qué podía ser reutilizable, después hacerlo reutilizable, lo siguiente era obligar a que todos conocieran lo que existe para reutilizar en la organización (puesto que si no, NO sabrían que existía), y luego intentar obligar a que reutilizaran lo existente por lo que tendrían que saber operar con ellos de antemano.

1.2. Ventajas y Desventajas

- Ventajas:
  - Reducción de costos.
  - Reducción de tiempo de desarrollo.
  - Reducción de esfuerzos durante la programación.
  - Facilita el mantenimiento con la apropiada documentación.
- Desventajas:
  - Ambigüedad del método a reutilizar.
  - Alto conocimiento en el lenguaje de desarrollo para implementarse de manera eficaz.
  - Conocer en su totalidad el objetivo final del programa así como contemplar sus limitantes.
  - El ritmo de avance del siglo XXI en programación hace más difícil implementarlo debido a que se puede encontrar desactualizada rápidamente.

1.3. Tipos de Reutilización

- Conocimiento: se reutiliza el conocimiento y la experiencia de otros desarrolladores.
- Sistemática: se diseñan componentes genéricos para que sean reutilizados con facilidad.
- Bottom-Up: se desarrollan pequeños componentes para una determinada aplicación.
- Top-Down: se van desarrollando las piezas necesarias poco a poco.
- Reutilización de código: librerías de funciones, editores, inclusión de ficheros, mecanismos de herencia en POO, componentes, etc.
- Reutilización de diseños: no volver a inventar arquitecturas.
- Reutilización de especificaciones: reutilización de las abstracciones del dominio.

1.4. Opiones

A menudo pensamos en el código como en algo muy valioso, y no hay nada mas alejado de esto.

El código tiene muy poco valor fuera del contexto de la empresa que lo creó. Hay que comenzar a entender que en el caso que el código no sea hecho explícitamente para ser genérico, su uso se torna muy problemático, incluso dentro de una misma empresa, cuando se intenta hacer el reuso de código entre diferentes proyectos.

Hay muchos equipos de desarrollo interno en empresas que no se dedican al desarrollo de software (y en las que si también) que gastan mucho tiempo construyendo frameworks caseros, bibliotecas que serán utilizados según ellos en varios proyectos:

Cuanto mayor es la cantidad de código acumulado, más la empresa está amarrada a los desarrolladores actuales (y mas ellos ocupan sus mentes con información aplicable solamente a una empresa o un equipo), y lo mas difícil será traer nuevos desarrolladores.

Tales frameworks y bibliotecas normalmente tienen una curva de aprendizaje muy grande para nuevos empleados especialmente porque la documentación es algo virtualmente ignorado- y ellos en forma muy rara pueden ser usados para cualquier otra cosa sin refactorizaciones significativas (porque ellos no fueron hechos para ser reutilizados de hecho, o lo fueron de manera tan arbitraria y superficial como fue posible...

Podemos concluir que los frameworks que son lo suficientemente abstractos y genéricos son mas caros de diseñar y desarrollar que las soluciones que serán utilizadas una única vez, y ya que el tiempo y costo para que un desarrollador absorva la complejidad de un

nuevo framework es dificilmente llevada a consideración al hacer un análisis de costo-beneficio del reúso del código, el reúso de código tiene mucho menos sentido del que puede imaginar. No tenemos que estar en contra del reúso del código en conjunto con frameworks; él es mucho mas positivo con relación a reutilizar código de bibliotecas que son estandares de mercado, por ejemplo. La verdad, es que tenemos que defender la adopción de frameworks que sean estándares de mercado (como los de código abierto), que por necesidad presentan abstracción y encapsulamiento apropiado, como alternativa a frameworks complejos desarrollados internamente.

Por todas estas razones, podemos estar seguros que el riesgo al robo de código es mínimo, y mucho menor al comunmente imaginado, y de esta forma las empresas no deberían tener miedo de compartir su código cuando eso tiene sentido.

Bueno, vamos a presentar un análisis realista sobre el valor de nuestro código, según Carl Lewis:

"No es nada intuitivo que algo tan difícil y caro de crear, pueda valer tan poco. Creo que por eso a tantas organizaciones les gusta fingir que su código es mucho mas valioso de lo que es en realidad".

2. Reingeniería

2.1. Introducción

Reingeniería del software se puede definir como: "modificación de un producto software, o de ciertos componentes, usando para el análisis del sistema existente técnicas de Ingeniería Inversa y, para la etapa de reconstrucción, herramientas de Ingeniería Directa, de tal manera que se oriente este cambio hacia mayores niveles de facilidad en cuanto a mantenimiento, reutilización, comprensión o evaluación."

Cuando una aplicación lleva siendo usada años, es fácil que esta aplicación se vuelva inestable como fruto de las múltiples correcciones, adaptaciones o mejoras que han podido surgir a lo largo del tiempo. Esto deriva en que cada vez que se pretende realizar un cambio se producen efectos colaterales inesperados y hasta de gravedad, por lo que se hace necesario, si se prevé que la aplicación seguirá siendo de utilidad, aplicar reingeniería a la misma.

2.2. Reestructuración de Código

El tipo más común de reingeniería es la reestructuración de código, se puede hacer con módulos individuales que se codifican de una manera que dificultan comprenderlos, probarlos y mantenerlos.

Llevar a cabo esta actividad requiere analizar el código fuente empleando una herramienta de reestructuración, se indican las violaciones de las estructuras de programación estructurada, y entonces se reestructura el código (esto se puede hacer automáticamente). El código reestructurado resultante se revisa y se comprueba para asegurar que no se hayan introducido anomalías.

Se actualiza la documentación interna del código.

Referencias

[1] http://tekina.info/solid-principles/

[2] https://www.quora.com/Object-Oriented-Design-What-is-the-significance-of-reuse

[3] https://medium.com/@sadatnazrul/software-development-tips