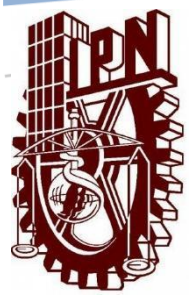

Instituto Politécnico Nacional

Escuela Superior de Cómputo

REINGENIERÍA Y REUTILIZACIÓN

Investigación

Análisis y Diseño Orientado a Objetos

Profesor: Chadwick Carreto Arellano

Grupo: 2CM7

Fecha: 21 / Mayo /2018

Alumno:

Calva Hernández José Manuel 2017630201

Contenido

Introducción

El proceso de evolución de un sistema implica entender que el programa necesita ser cambiado e implementar esos cambios. Sin embargo, muchos sistemas, especialmente los antiguos, son difíciles de entender y cambiar. Los programas pudieron haber sido optimizados para el rendimiento o el espacio utilizado a costa del entendimiento o, con el paso del tiempo, la estructura inicial del programa pudo ser corrompida por una serie de cambios.

Para hacer fácil de mantener los sistemas de software, nosotros podemos hacer una reingeniería sobre estos para mejorar su estructura y entendimiento. La reingeniería toma tiempo, cuesta cantidades significativas de dinero y absorbe recursos que de otro modo pueden ocuparse en preocupaciones inmediatas. Por todas estas razones, la reingeniería no se logra en pocos meses o incluso en algunos años. La reingeniería de los sistemas de información es una actividad que absorberá recursos de tecnología de la información durante muchos años. Por esto, toda organización necesita una estrategia pragmática para la reingeniería de software.

Esto es una actividad de reconstrucción, que se puede ver con una analogía, la reconstrucción de una casa. Considere la siguiente situación. Usted compra una casa en otro estado. En realidad, nunca ha visto la propiedad, pero la adquirió a un precio sorprendentemente bajo, con la advertencia de que es posible que deba reconstruirla por completo. ¿Cómo procedería?

- Antes de comenzar a reconstruir, parecería razonable inspeccionar la casa. Para determinar si necesita reconstruirse, usted (o un inspector profesional) crea una lista de criterios, de modo que su inspección sea sistemática.
- Antes de demoler y reconstruir toda la casa, asegúrese de que la estructura es débil. Si la casa es estructuralmente sólida, acaso sea posible “remodelar” sin reconstruir (a un costo mucho más bajo y en mucho menos tiempo).
- Antes de comenzar a reconstruir, asegúrese de entender cómo se construyó la original. Eche un vistazo detrás de las paredes. Entienda cómo están el alambrado, la plomería y la estructura interna. Incluso si tira todo a la basura, la comprensión que obtenga le servirá cuando comience la construcción.
- Si comienza a reconstruir, use solamente los materiales más modernos y más duraderos. Esto puede costar un poco más ahora, pero le ayudará a evitar costos y tardados mantenimientos posteriores.
- Si decide reconstruir, sea disciplinado en ello. Use prácticas que resultarán en alta calidad, hoy y en el futuro.

Aunque estos principios se enfocan en la reconstrucción de una casa se aplican igualmente bien a la reingeniería de los sistemas y aplicaciones basados en cómputo.

Reingeniería

La reingeniería de software tiene que ver con reestructurar y redocumentar el software para hacerlo más fácil de entender y cambiar. Puede incluir no sólo estas dos actividades, también otras como refactorizar la arquitectura del sistema, trasladar el programa a un lenguaje de programación moderno y modificar y actualizar la estructura y los valores de los datos en el sistema. La funcionalidad del programa no es cambiada y, normalmente, se evitará hacer cambios grandes en la arquitectura del sistema.

Beneficios

Existen dos grandes beneficios en la reingeniería de software por sobre el reemplazamiento del sistema, estos son:

1. Reducción de riesgo: Hay un gran riesgo en volver a desarrollar software crítico en ciertos negocios. Pueden cometerse errores en las especificaciones del sistema o pueden generarse problemas en el desarrollo. Retrasos en la introducción de un nuevo software pueden significar pérdidas e incluso costos extras para el negocio.
2. Reducción de costos: A pesar de que el costo inicial de desarrollar un software considerando la reingeniería en él puede ser un poco mayor que una simple, este costo es significativamente menor que el desarrollo de desarrollar un nuevo software cuando el antiguo deje de ser funcional o no pueda ser mantenible por más tiempo.

Proceso de reingeniería

Hay distintas formas de tomar el proceso de reingeniería según distintos autores, se mencionarán dos por ver las distintas perspectivas, según Ian Sommerville, el proceso de reingeniería sería el siguiente:

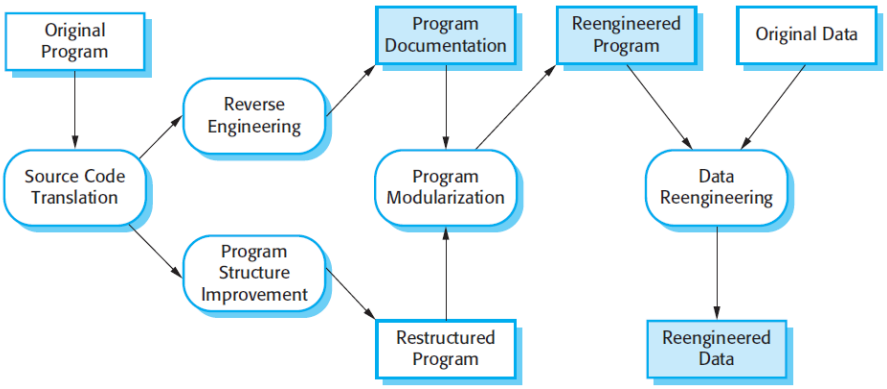

Figura 1. El proceso de reingeniería

La figura aquí mostrada muestra el proceso general de reingeniería. La entrada del proceso es el programa de legado y la salida es una versión mejorada y reestructurada del mismo programa. Las actividades que en este proceso son las siguientes:

1. *Traducción de código fuente*: Utilizando una herramienta de traducción, el programa se convierte de un antiguo lenguaje de programación a una versión más moderna del mismo lenguaje o a un lenguaje diferente.

2. *Ingeniería inversa*: El programa se analiza y se extrae información de este. Esto ayuda a documentar su organización y funcionalidad. De nuevo, este proceso suele ser completamente automático.

3. *Mejora de la estructura del programa*: La estructura de control del programa se analiza y modifica para facilitar su lectura y comprensión. Esto puede ser parcialmente automatizado, pero generalmente se requiere alguna intervención manual.

4. *Modularización del programa*: Las partes relacionadas del programa se agrupan y, cuando corresponde, se elimina la redundancia. En algunos casos, esta etapa puede implicar la refactorización arquitectónica (implica el proceso de reestructurar código existente sin cambiar su faceta externa). Este es un proceso manual.

5. *Reingeniería de datos*: Los datos procesados por el programa cambian para reflejar los cambios del programa. Esto puede significar la redefinición de esquemas de bases de datos y la conversión de bases de datos existentes a la nueva estructura. Por lo general, también debe limpiar los datos. Esto implica encontrar y corregir errores, eliminar registros duplicados, etc. Hay herramientas disponibles para respaldar la reingeniería de datos.

El proceso de reingeniería no necesariamente debe seguir o pasar por todos los pasos descritos anteriormente, esto puede hacer variar el costo de la reingeniería dependiendo de cuantos de los procesos se realicen y el nivel de automatización que se tenga para cada uno, porque de tener que hacer todos de forma manual, el costo será considerablemente mayor al requerir mayor tiempo y mano de obra que si fuesen automatizados.

El mayor problema con la reingeniería de software es que hay limitaciones prácticas para lo mucho que se puede mejorar un sistema con la reingeniería. Cambios mayores a la arquitectura del sistema o una reorganización radical de los datos del sistema pueden no ser llevados a cabo de manera automática, por lo tanto, es muy costoso. Por otra parte, la reingeniería puede mejorar el mantenimiento de un sistema a largo plazo, sin embargo, este sistema puede ser menos mantenible que uno desarrollado de nuevo por completo usando métodos de ingeniería modernos.

Otro enfoque distinto de este proceso de reingeniería nos lo presenta Roger Pressman y es el siguiente:

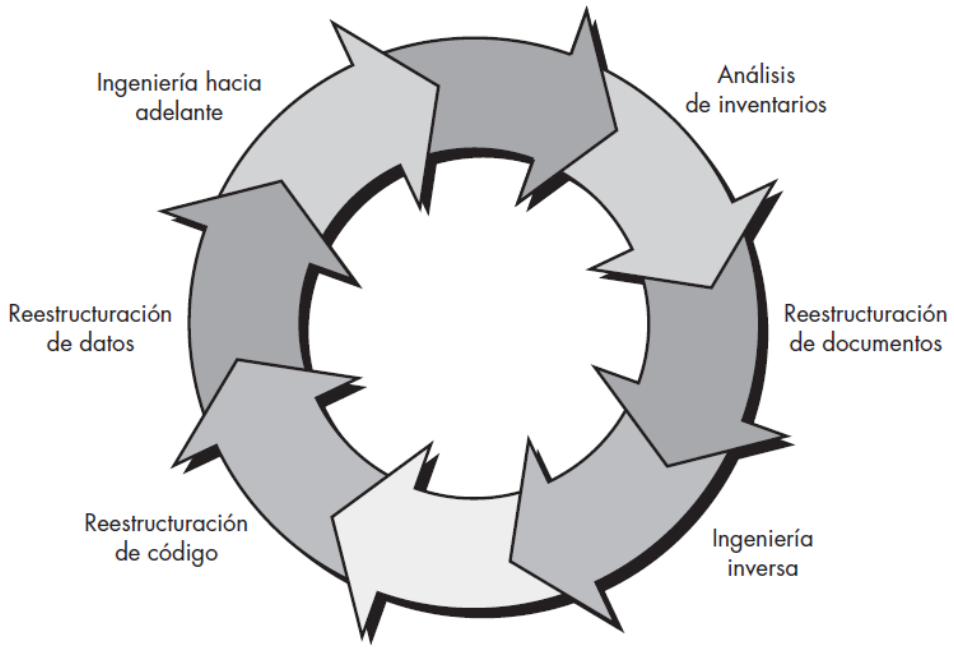

Figura 2. Modelo de proceso de reingeniería de software

El paradigma de reingeniería que se muestra en la figura anterior es un modelo cíclico. Esto significa que cada una de las actividades presentadas como parte del paradigma puede revisarse. Para algún ciclo particular, el proceso puede terminar después de cualquiera de estas actividades.

Análisis de inventarios. Toda organización de software debe tener un inventario de todas las aplicaciones. El inventario puede ser nada más que un modelo de hojas de cálculo que contenga información que ofrezca una descripción detallada (por ejemplo, tamaño, edad, importancia empresarial) de cada aplicación activa. Al ordenar esta información de acuerdo con importancia empresarial, longevidad, mantenibilidad actual, soportabilidad y otros importantes criterios locales, aparecen los candidatos para reingeniería. Entonces pueden asignarse recursos a esas aplicaciones.

Es importante observar que el inventario debe revisarse con regularidad. El estado de las aplicaciones (por ejemplo, importancia empresarial) puede cambiar con el tiempo y, como resultado, cambiarán las prioridades para aplicar la reingeniería.

Reestructuración de documentos. La documentación débil es el distintivo de muchos sistemas heredados. Pero, ¿qué puede hacer con ella? ¿Cuáles son sus opciones?

1. La creación de documentación consume demasiado tiempo. Si el sistema funciona puede elegir vivir con lo que tiene. En algunos casos, éste es el enfoque correcto. No es posible volver a crear documentación para cientos de programas de cómputo. Si un programa es relativamente estático, se aproxima al final de su vida útil y es improbable que experimente cambio significativo, ¡déjelo así!

2. La documentación debe actualizarse, pero su organización tiene recursos limitados. Use un enfoque “documente cuando toque”. Acaso no sea necesario volver a documentar por completo una aplicación. En vez de ello, aquellas porciones del sistema que en el momento experimenten cambio se documentan por completo. Con el tiempo, evolucionará una colección de documentación útil y relevante.

3. El sistema tiene importancia empresarial y debe volver a documentarse por completo. Incluso en este caso, un enfoque inteligente es recortar la documentación a un mínimo esencial. Cada una de estas opciones es viable. Su organización de software debe elegir aquella que sea más adecuada para cada caso.

Ingeniería inversa. El término ingeniería inversa tiene su origen en el mundo del hardware. Una compañía desensambla un producto de hardware de otra empresa con la intención de entender los “secretos” de diseño y fabricación de su competidor. Dichos secretos podrían entenderse fácilmente si se obtuvieran las especificaciones de diseño y fabricación. Pero esos documentos son propiedad de la empresa competidora y no están disponibles para la compañía que hace la ingeniería inversa. En esencia, la ingeniería inversa exitosa deriva en una o más especificaciones de diseño y fabricación para un producto al examinar especímenes reales del mismo.

La ingeniería inversa para el software es muy similar. No obstante, en la mayoría de los casos, el programa que se va a someter a ingeniería inversa no es de un competidor: es el propio trabajo de la compañía (con frecuencia, elaborado muchos años atrás). Los “secretos” por entender son oscuros porque jamás se desarrollaron especificaciones. Por tanto, la ingeniería inversa para software es el proceso de analizar un programa con la intención de crear una representación del mismo en un nivel superior de abstracción que el código fuente. La ingeniería inversa es un proceso de recuperación de diseño. Las herramientas de ingeniería inversa extraen información de diseño de datos, arquitectónico y procedimental de un programa existente.

Reestructuración de código. El tipo más común de reingeniería (en realidad, en este caso es cuestionable el uso del término reingeniería) es la reestructuración de código. Algunos sistemas heredados tienen una arquitectura de programa

relativamente sólida, pero los módulos individuales fueron codificados en una forma que los hace difíciles de entender, poner a prueba y mantener. En tales casos, el código dentro de los módulos sospechosos puede reestructurarse.

Para realizar esta actividad se analiza el código fuente con una herramienta de reestructuración. Las violaciones a los constructos de programación estructurada se anotan y luego el código se reestructura (esto puede hacerse automáticamente) o incluso se reescribe en un lenguaje de programación más moderno. El código reestructurado resultante se revisa y pone a prueba para garantizar que no se introdujeron anomalías. La documentación de código interna se actualiza.

Reestructuración de datos. Un programa con arquitectura de datos débil será difícil de adaptar y mejorar. De hecho, para muchas aplicaciones, la arquitectura de información tiene más que ver con la viabilidad a largo plazo de un programa que con el código fuente en sí.

A diferencia de la reestructuración de código, que ocurre en un nivel de abstracción relativamente bajo, la reestructuración de datos es una actividad de reingeniería a gran escala. En la mayoría de los casos, la reestructuración de los datos comienza con una actividad de ingeniería inversa. La arquitectura de datos existente se diseca y se definen modelos de datos necesarios. Se identifican los objetos y atributos de datos, y se revisa la calidad de las estructuras de datos existentes.

Cuando la estructura de datos es débil (por ejemplo, si se implementan archivos planos, cuando un enfoque relacional simplificaría enormemente el procesamiento), los datos se someten a reingeniería.

Puesto que la arquitectura de datos tiene una fuerte influencia sobre la arquitectura del programa y sobre los algoritmos que los pueblan, los cambios a los datos invariablemente resultarán en cambios arquitectónicos o en el nivel de código.

Ingeniería hacia adelante. En un mundo ideal, las aplicaciones se reconstruirían usando un “motor de reingeniería” automático. El programa antiguo se alimentaría en el motor, se analizaría, se reestructuraría y luego se regeneraría de manera que mostrara los mejores aspectos de la calidad del software. A corto plazo es improbable que tal “motor” aparezca, pero los proveedores introdujeron herramientas que proporcionan un subconjunto limitado de dichas capacidades y que abordan dominios de aplicación específicos (por ejemplo, aplicaciones que se implementan usando un sistema de base de datos específico). Más importante, dichas herramientas de reingeniería se vuelven cada vez más sofisticadas.

La ingeniería hacia adelante no sólo recupera información de diseño del software existente, sino que también usa esta información para alterar o reconstituir el sistema existente con la intención de mejorar su calidad global. En la mayoría de los casos, el software sometido a reingeniería vuelve a implementar la función del sistema existente y también añade nuevas funciones y/o mejora el rendimiento global.

Reutilización

La administración de la reutilización se define como los criterios para volver a usar el producto del trabajo (incluso los componentes del software) y establece mecanismos para obtener componentes reutilizables. Así, una definición más acertada puede ser: el proceso de crear sistemas software a partir de software preexistente, en lugar de crearlos empezando de cero:

Para que un componente sea, en efecto, reutilizable, debe encapsular su funcionalidad, y debe ofrecerla al exterior a través de una o más interfaces que el propio componente debe implementar. Así, cuando el sistema en el que incluimos el componente desea utilizar una de las funcionalidades que este ofrece, se la pide a través de la interfaz. Pero, del mismo modo, el componente puede también requerir otras interfaces para, por ejemplo, comunicar los resultados de su cómputo.

La ingeniería del software basada en componentes (CBSE: component-based software engineering) se ocupa del desarrollo de sistemas software a partir de componentes reutilizables, como en el sencillo ejemplo que acabamos de presentar. Pero la CBSE, sin embargo, es solo una de las líneas de trabajo de la comunidad científica y de la industria del software para favorecer la reutilización y, de este modo, evitar la reinvención continua de la rueda o, en nuestro entorno, impedir el análisis, el diseño, la implementación y la prueba de la misma solución, desarrollándola una y otra vez en mil y un proyectos distintos.

De este modo, la reutilización abarata los costes del desarrollo: en primer lugar, porque no se necesita implementar una solución de la que ya se dispone; en segundo lugar, porque aumenta la productividad, al poder dedicar los recursos a otras actividades más en línea con el negocio; en tercer lugar, porque probablemente el elemento que reutilizamos ha sido suficientemente probado por su desarrollador, con lo que los testers podrán dedicarse a probar otras partes más críticas del sistema, obteniendo entonces unos niveles de calidad mucho más altos que si el sistema se desarrolla desde cero.

Ventajas

- Reducir el tiempo de desarrollo.
- Reducir los costos.
- Incrementar la productividad.
- No tener que reinventar las soluciones.
- Facilitar la compartición de productos del ciclo de vida.
- Mayor fiabilidad
- Mayor eficiencia (Aunque al principio pueda parecer que no)
- Consistencia y la familiaridad, los patrones dentro del software serán más consistentes, tendiendo a facilitar el mantenimiento del producto.

Desventajas

- Necesidad de invertir antes de obtener resultados.
- Carencia de métodos adecuados.
- Necesidad de formar al personal.
- Convencer a los “manager”.

- Dificultad para institucionalizar los procesos.

Proceso de desarrollo

Al igual que con la reingeniería, hay distintas aproximaciones para el proceso de desarrollo, describiremos dos modelos a continuación, el primero de ellos es descrito por Sametinger y se esquematiza de la siguiente manera:

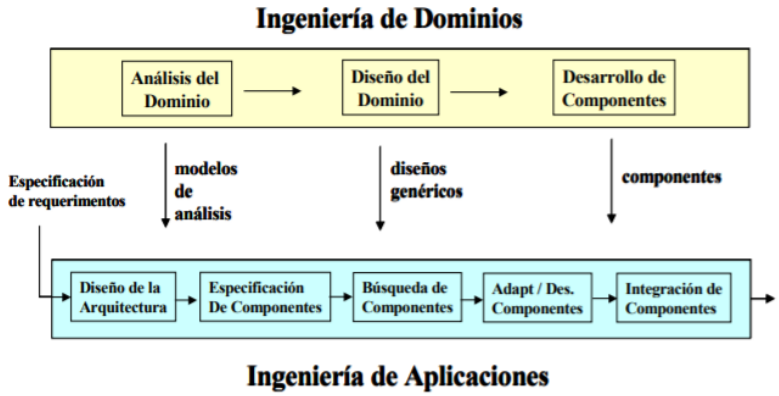

Figura 3. Modelo de procesos gemelos para el desarrollo de software basado en componentes

El modelo de procesos denominado *ciclo de vida gemelo (twin life cycle)*, divide el proceso de desarrollo de software en dos grandes bloques paralelos. El primer bloque, conocido como Ingeniería de Dominios, contempla los procesos necesarios para desarrollar activos de software reutilizables en un dominio particular. El segundo bloque es denominado Ingeniería de Aplicaciones, su propósito es el desarrollo de aplicaciones basado en la reutilización de activos de software producidos a través de la Ingeniería de Dominios.

Un modelo alternativo al modelo de Ingeniería de Aplicaciones es el modelo WATCH. Este modelo combina los procesos más relevantes de la Ingeniería de Software Orientada a Objetos con el modelo de Ingeniería de Aplicaciones del ciclo de vida gemelo. Una característica importante de este modelo es la integración de los procesos gerenciales con los procesos técnicos del desarrollo de software basado en componentes. Esta integración facilita la labor del líder del proyecto, al describir cómo se debe llevar a cabo una gestión del proyecto integrada a los procesos técnicos del desarrollo de software.

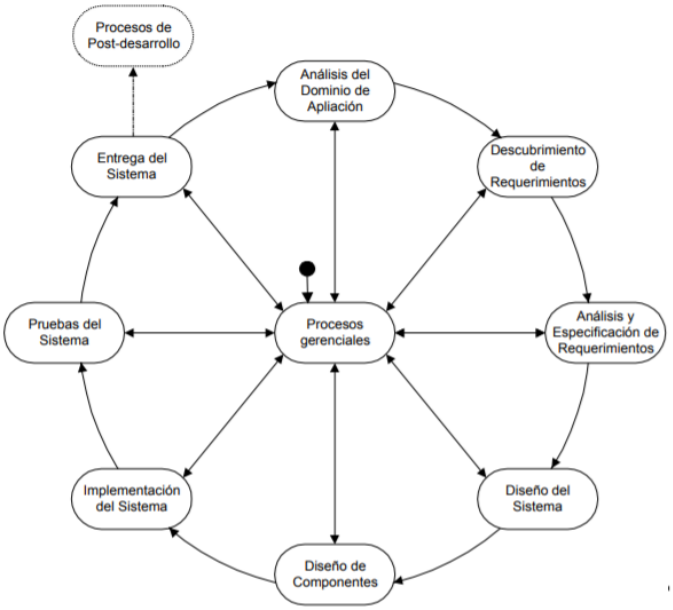

Figura 4. El modelo de procesos WATCH

La estructura del método WATCH se ilustra en la figura anterior. Esta estructura emplea la metáfora de un reloj de pulsera para describir el orden de ejecución de los procesos técnicos de desarrollo de aplicaciones, indicando además cómo los procesos gerenciales controlan o coordinan el orden de ejecución de los procesos técnicos. Los procesos gerenciales están ubicados en el centro de la estructura para indicar explícitamente que ellos programan, dirigen y controlan el proceso de desarrollo. Los procesos técnicos están ubicados en el entorno siguiendo la forma que tiene el dial de un reloj. Ello indica que el orden de ejecución de las fases técnicas se realiza en el sentido de las agujas del reloj. Los procesos gerenciales pueden, sin embargo, invertir el orden de ejecución para repetir algunas de las fases anteriores.

Las tres primeras fases del modelo son similares a los modelos de procesos tradicionales. Las restantes se pueden describir como:

- *Análisis del contexto:* Permite que el grupo de desarrollo adquiera un conocimiento adecuado del dominio o contexto del sistema en desarrollo.
- *Descubrimiento, Análisis y Especificación de Requerimientos*: Se encargan de identificar las necesidades y requerimientos de los usuarios, así como analizarlos, especificarlos gráficamente y documentarlos.
- *Diseño del sistema*: Establece y describe la arquitectura del software. Describe cada uno de los componentes que requiere tal estructura y cómo esos componentes se interconectan para interactuar. El grupo de desarrollo debe, a partir de esta arquitectura, determinar cuáles componentes se pueden reutilizar y cuáles requieren ser desarrollados en su totalidad. Los componentes reutilizados deben ser adaptados, para satisfacer los requerimientos del sistema.
- *Implementación*: Los componentes nuevos, deben ser diseñados, codificados y probados separadamente.
- *Pruebas del sistema*: Permiten detectar errores en la integración de los componentes.
- *Entrega*: Se encarga de la instalación, adiestramiento de usuarios y puesta en operación del sistema.

Bibliografía

Sommerville, I. (2011). *Software engineering* (9th ed.). Boston: Pearson.

Pressman, R. (2010). *Ingeniería de Software. Un enfoque práctico* (7th ed.). Ciudad de México: Mc Graw Hill Educación.

Webgrafía

Escobar, J. (2018). Reutilización de Software. Retrieved from http://jurifa-ingenieriadesoftware.blogspot.mx/2012/09/reutilizacion-de-software.html

Montilva, J., Arapé, N., & Colmenares, J. (2003). http://juancol.me/rsrc/sw-basado-en-comp-CAC2003.pdf. Retrieved from http://juancol.me/rsrc/sw-basado-en-comp-CAC2003.pdf

Polo, M. (2010). Desarrollo de software basado en reutilización. Retrieved from https://www.exabyteinformatica.com/uoc/Informatica/Tecnicas_avanzadas_de_ingenieria_de_software/Tecnicas_avanzadas_de_ingenieria_de_software_(Modulo_2).pdf