



CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

Índice Tema 13

Introducción.

1. La evolución de los sistemas de información.
 - 1.1. El caso de los sistemas heredados o «legacy systems».
2. Concepto de mantenimiento del software.
3. Tipos de mantenimiento del software.
 - 3.1. El mantenimiento correctivo.
 - 3.2. Los mantenimientos adaptativo y perfectivo.
 - 3.3. El mantenimiento preventivo.
4. La mantenibilidad del software.
5. La gestión del mantenimiento del software.
6. El coste de mantenimiento del software.
 - 6.1. Estimación de los costes de mantenimiento del software.
7. Ingeniería inversa y reingeniería.
 - 7.1. Ingeniería inversa.
 - 7.2. Objetivos y beneficios de la ingeniería inversa.
 - 7.3. Reingeniería.
8. Abandono del sistema.
9. Evaluación post-implementación.





CENTRO DE ESTUDIOS FINANCIEROS

VIRIATO, 52	28010 MADRID	914 44 49 20
PONZANO, 15	28010 MADRID	914 44 49 20
G. DE GRÀCIA, 171	08012 BARCELONA	934 15 09 88
ALBORAYA, 23	46010 VALENCIA	963 61 41 99

www.cef.es

info@cef.es

TEMA 13

Instalación y cambio. Estrategias de sustitución. Recepción e instalación. Evaluación post-implementación. Mantenimiento.

INTRODUCCIÓN.

El proceso de Desarrollo de un Sistema de Información, según el estándar que propone el IEEE, se puede considerar que está compuesto por cuatro procesos principales:

1. Proceso de selección de un modelo de ciclo de vida, esto es, identificar y seleccionar el ciclo de vida adecuado para el sistema que se va a construir.
2. Procesos de gestión del proyecto, que son aquellos que crean la estructura del proyecto y aseguran el nivel adecuado de gestión del mismo durante todo el ciclo de vida del software. Comprenden actividades de planificación, estimación de recursos, seguimiento y control y evaluación del proyecto; esta última, mediante la gestión de la calidad del software.
3. Procesos orientados al desarrollo del software, que son los que producen, instalan, operan y mantienen el software y lo retiran de su uso. Estos procesos se clasifican en:
 - a) Procesos de predesarrollo, que abarcan desde el reconocimiento del problema, hasta la determinación de los requisitos funcionales a nivel de sistema, pasando por el estudio de la viabilidad de su solución automatizada.
 - b) Procesos de desarrollo propiamente dicho, que son los que se deben realizar para la construcción del producto software. Definen qué información hay que obtener y cómo estructurar los datos, qué algoritmos usar para procesar los datos y cómo implementarlos, y qué interfaces desarrollar para operar con el software y cómo hacerlo
 - c) Procesos de post-desarrollo, que son aquellos que se deben realizar para instalar, operar, soportar, mantener y retirar un producto software.

4. Procesos integrales del proyecto, es decir, aquellos que aseguran la terminación y calidad del proyecto. Estos procesos son simultáneos y complementarios a los orientados al desarrollo, e incluyen actividades imprescindibles para que el sistema construido sea fiable y sea utilizado al máximo de sus capacidades.

La finalidad de este tema es el estudio de los procesos de post-desarrollo del sistema de información, en concreto, el mantenimiento del software y la problemática relativa a la sustitución del sistema.

Un sistema de información no puede concebirse como algo estático que permanece invariable desde que se crea hasta su extinción. Bien al contrario, los productos software requieren una evolución continua durante todo su ciclo de vida para irse adaptando a las nuevas necesidades, ya sean del mercado, del entorno operativo o de los clientes y usuarios.

La evolución del sistema de información tiene como finalidad última su adecuación al entorno o «medio ambiente», es decir, su adaptación para que pueda realizar nuevas funciones, trabajar más efectivamente o trabajar más correctamente (con menos errores).

En organizaciones con sistemas antiguos o heredados, la necesidad de adaptar estos sistemas puede sobrepasar el concepto clásico de mantenimiento y convertirse en una labor de reingeniería que requiere de profundos conocimientos del sistema establecido, de cara a su mejora o incluso a su abandono a favor de un nuevo sistema que cumpla con los requisitos necesarios para un rendimiento óptimo.

En definitiva, la necesidad ineludible de evolución, o incluso de sustitución, de los sistemas de información requiere un análisis completo de los cambios o adaptaciones necesarias, de sus motivaciones, de qué estrategias de sustitución llevar a cabo, y de cómo instalar y adquirir el nuevo sistema o los nuevos componentes para mejorar el sistema existente. Asimismo, también es imprescindible una metodología fiable que permita gestionar los cambios y evaluar el correcto funcionamiento del sistema, en tanto en cuanto responde a las expectativas esperadas, una vez realizados dichos cambios.

Todo ello va a ser objeto de desarrollo en este tema, el cual pretende poner de manifiesto:

1. Una serie de ideas generales respecto a la evolución de los sistemas, con referencia especial al caso de los sistemas heredados.
2. El concepto de mantenimiento del software, con sus diversos tipos y la problemática que le afecta: la mantenibilidad del software, el coste del mantenimiento y la gestión del mantenimiento.
3. El concepto de Reingeniería y de Ingeniería Inversa.
4. Los aspectos a considerar para el abandono del sistema.

1. LA EVOLUCIÓN DE LOS SISTEMAS DE INFORMACIÓN.

Como consecuencia de la rápida evolución que sufre el mundo de la informática, tanto en el ámbito del hardware como del software, es muy importante para cualquier organización plantearse la necesidad de adaptar sus sistemas a estos continuos cambios y exigencias. Estas exigencias conllevan la necesidad de obtener el máximo rendimiento de los sistemas, lo cual implica, tanto las labores fundamentales de adaptación y mantenimiento del hardware, como muy particularmente del software. Se-

gún esto, dos fuerzas dirigen la evolución de los sistemas informáticos: las nuevas aplicaciones, esto es, el nuevo software, y el nuevo hardware, pero, en cualquier caso, la evolución de ambos está íntimamente ligada.

La evolución de un sistema es un concepto amplio que abarca desde una simple modificación para corregir un «bug» de un programa hasta una reimplantación completa del sistema. Las actividades de evolución que se pueden realizar sobre un sistema son de tres tipos: mantenimiento, reingeniería y abandono.

La siguiente figura ilustra cómo son aplicadas cada una de las actividades de evolución durante el ciclo de vida de un sistema.

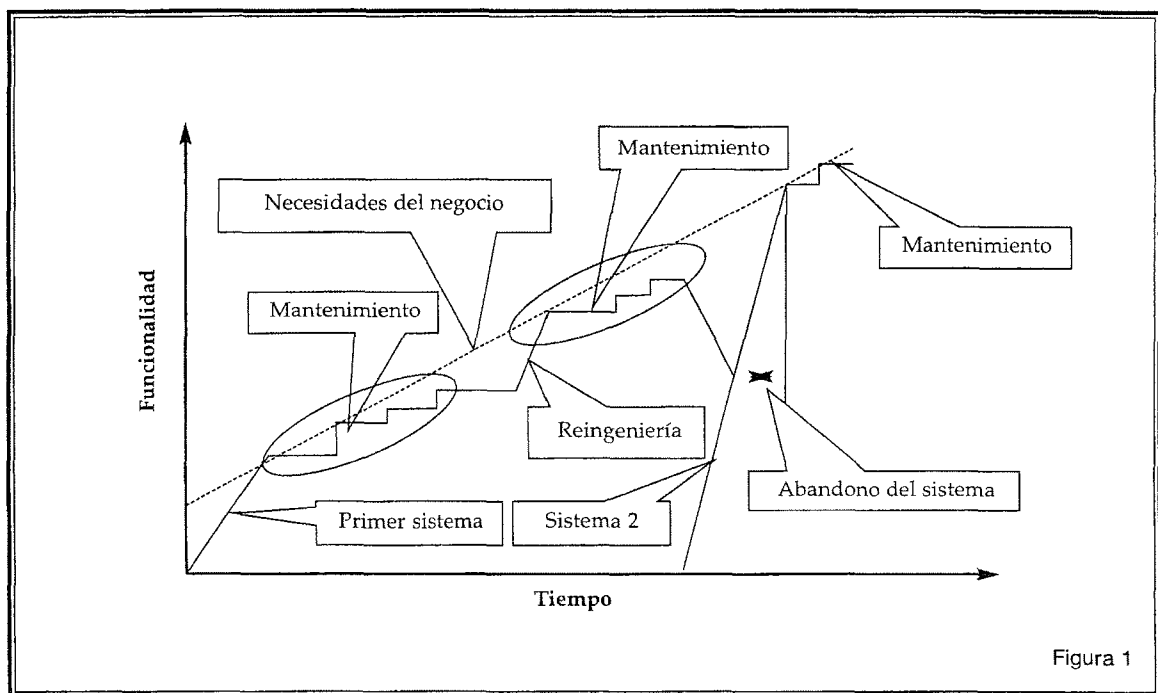


Figura 1

Así pues, como causas genéricas para evolucionar se pueden considerar las siguientes:

- La detección de defectos de desarrollo.
- La aparición de fallos (bugs) una vez implantado el sistema, lo que implica un mantenimiento de emergencia (mantenimiento correctivo).
- Un cambio drástico en el entorno.
- La posibilidad de disponer de nuevo hardware y/o nuevo software.
- La necesidad de seguridad del sistema ante futuros problemas, y en general, la necesidad de perfeccionar el sistema.

Los cambios pueden realizarse de forma gradual y constante, o bien de forma abrupta.

- En el primer caso, el cambio se realiza de manera suave y no supone romper con los fundamentos estructurales del sistema, ya sea en aspectos hardware o software. Es un cambio progresivo o evolutivo que no supone un cambio a un nuevo sistema sino actualizar y mejorar progresivamente el actual.
- En el caso de una sustitución abrupta, el cambio del sistema es total o casi total. Es un cambio brusco que supone implantar un nuevo sistema de información prescindiendo prácticamente de todo el hardware y/o el software del sistema anterior.

Pero, en cualquier caso, durante el proceso de cambio del sistema hay que prever cuál va a ser la gestión de su evolución posterior. La gestión de la evolución debe consistir en dar una respuesta rápida, preparada y eficiente a los cambios que se produzcan en el entorno ya sean éstos de índole tecnológico o de gestión del propio negocio.

1.1. EL CASO DE LOS SISTEMAS HEREDADOS O «LEGACY SYSTEMS».

La mayor parte de los grandes sistemas de información que están hoy funcionando en las empresas del país fueron desarrollados en los años ochenta. La irrupción de las tecnologías relacionadas con Internet, el paradigma de la Orientación a Objetos, los componentes distribuidos y la nueva mentalidad empresarial que intenta ofrecer mejores servicios a sus clientes y durante más tiempo, han provocado que la información que permanecía en los viejos sistemas y que es totalmente aprovechable, sea objeto de diversos tratamientos para su recuperación. Estas necesidades de evolución y adaptación a los nuevos requerimientos tecnológicos y de negocio empujan al sistema a una nueva situación a la que no es posible llegar a través del mantenimiento clásico.

En una primera aproximación, la solución a estos problemas se puede presentar a través de dos caminos: un nuevo desarrollo que incorpore nuevas tecnologías y funcionalidades o por medio de la aplicación de reingeniería al sistema actual (sistema heredado o «legacy system»).

Ante una situación como la descrita anteriormente, una vez analizado el «legacy system», las organizaciones tienen que decidir el camino a tomar para actualizar el sistema, determinación que se resume en un cambio para adecuarlo a las nuevas necesidades, el cual puede concretarse en una de las opciones siguientes:

- Reingeniería del sistema.
- Abandonar el sistema y sustituirlo por otro nuevo.
- Optar por una solución híbrida entre las dos anteriores.

En cualquier caso, como denominador común de todas las opciones anteriores, la organización debe incluir, dentro del proyecto correspondiente, una buena gestión de la evolución del software que se produzca, o dicho en palabras de Lehman, «es necesario dotar a las organizaciones de metodologías para llevar a cabo las tareas de mantenimiento y evolución del sistema de información».

1. REINGENIERÍA.

La definición dada por el Reengineering Center del Software Engineering Institute de la Universidad Carnegie Mellon es la siguiente:

Reingeniería es la transformación sistemática de un sistema existente a una nueva forma para realizar mejoras de la calidad en operación, capacidad del sistema, funcionalidad, rendimiento o capacidad de evolución a bajo coste, con un plan de desarrollo corto y con bajo riesgo para el cliente.

En esta definición se destaca el hecho de que la reingeniería es la mejora de sistemas existentes de modo que la inversión resulte muy rentable. Si la reingeniería no tiene un coste bajo, no está acabada en poco tiempo, no tiene poco riesgo o no ofrece un valor añadido para el cliente, hay que considerar la posibilidad de un nuevo desarrollo.

Los criterios para aplicar la reingeniería se basan tanto en técnicas heurísticas (como la edad del código o el coste del personal de mantenimiento), como en modelos de coste más sofisticados.

2. ABANDONO.

Un sistema se abandona cuando no es capaz de seguir el ritmo de las necesidades del negocio y su reingeniería no es posible o no se puede hacer con un coste efectivo. Normalmente, estos sistemas adolecen de documentación, de actualización y de capacidad de expansión.

El abandono de un sistema comporta riesgos puesto que es una actividad que consume muchos recursos y que implica a los mantenedores actuales del sistema, y puesto que abandonar el sistema no significa empezar un desarrollo desde cero. No se puede pensar que los datos que contiene el sistema actual van a desecharse y que el conocimiento que conservan sus expertos va a perderse. El abandono del sistema debe venir precedido por un análisis exhaustivo que permita decidir el método que se va a utilizar para la migración de la información del sistema actual al nuevo, incluyendo datos, funciones de negocio y arquitectura del sistema.

Resumiendo, en la elección de un nuevo sistema habrá que considerar factores como:

- Los motivos para el cambio.
- La capacidad de adquisición del usuario.
- La necesidad del cambio.
- La estimación del tiempo en realizarse el cambio.

2. CONCEPTO DE MANTENIMIENTO DEL SOFTWARE.

El mantenimiento se puede considerar como la primera forma de evolución de los sistemas. Históricamente, el término «mantenimiento» se ha aplicado al proceso de modificar un programa cuando se ha entregado a producción y está en uso. Estas modificaciones pueden implicar cambios sencillos, para corregir errores de codificación, cambios mayores, para corregir errores de diseño, e incluso, reescrituras del programa, para corregir errores de especificación o introducir nuevos requerimientos funcionales no contemplados.

En general, el mantenimiento persigue una mejora de la calidad de las aplicaciones, ahora bien, no materializando todas las solicitudes de cambio, sino, a efectos de eficacia, planificando, analizando y estudiando el impacto y la rentabilidad de las modificaciones al sistema al ritmo de las exigencias de la empresa u organización. Quiere esto decir que habrá de existir una Gestión de Mantenimiento, que defina un conjunto de técnicas para realizar las modificaciones, con el fin de detectar las desviaciones del diseño original, corregir los errores detectados y permitir la inclusión de nuevas funcionalidades.

A la hora de definir lo que es el mantenimiento y su ámbito de estudio, existen diferentes opiniones. Para unos, el mantenimiento se limita a la corrección de errores, pues consideran que cualquier mejora de otra índole supone un rediseño y no un puro mantenimiento; para otros, el mantenimiento consiste en un conjunto de tareas que tienen como fin introducir un cambio en el sistema, ya sea de mejora o corrección. Nosotros, sin embargo, vamos a asumir una definición más acorde con las nuevas tendencias, en concreto, la definición que da el IEEE (Institute for Electrical and Electronic Engineering).

Mantenimiento es el conjunto de modificaciones de un producto software, realizado después de su desarrollo e implementación, para la corrección de errores, para mejorar el rendimiento u otros atributos del software, o para adaptar el producto a los cambios del entorno.

A la vista de la definición aportada, se observa que el mantenimiento tiene lugar una vez que se distribuye el sistema para su explotación, y podríamos describirlo mediante las siguientes cuatro actividades:

1. La primera es la corrección de errores, puesto que no es asumible que las pruebas efectuadas durante el desarrollo hayan descubierto todos ellos, y con el uso del sistema se producirán todavía algunos. Es el llamado «mantenimiento correctivo», que puede considerarse como un mantenimiento de emergencia.
2. La segunda actividad que contribuye a la definición del mantenimiento se produce por el rápido cambio inherente a cualquier aspecto de la Informática. El hardware evoluciona rápidamente ofreciendo mejores entornos de producción y haciendo que el software desarrollado no soporte esta evolución y haya de modificarse.
3. La tercera actividad se produce cuando el sistema tiene éxito, puesto que a medida que se usa se reciben nuevas peticiones de los usuarios sobre la inclusión de nuevas funcionalidades o mejora de las existentes.
4. Por último, otra actividad de mantenimiento se da cuando se cambia el sistema para mejorar una futura facilidad de mantenimiento o fiabilidad.

En definitiva, se puede concluir que las causas que originan el mantenimiento son de dos tipos: externas, derivadas de necesidades no contempladas en el origen del sistema, o internas, que son las derivadas de errores en el desarrollo, o por la necesidad de actualizar un sistema que se va quedando obsoleto.

3. TIPOS DE MANTENIMIENTO DEL SOFTWARE.

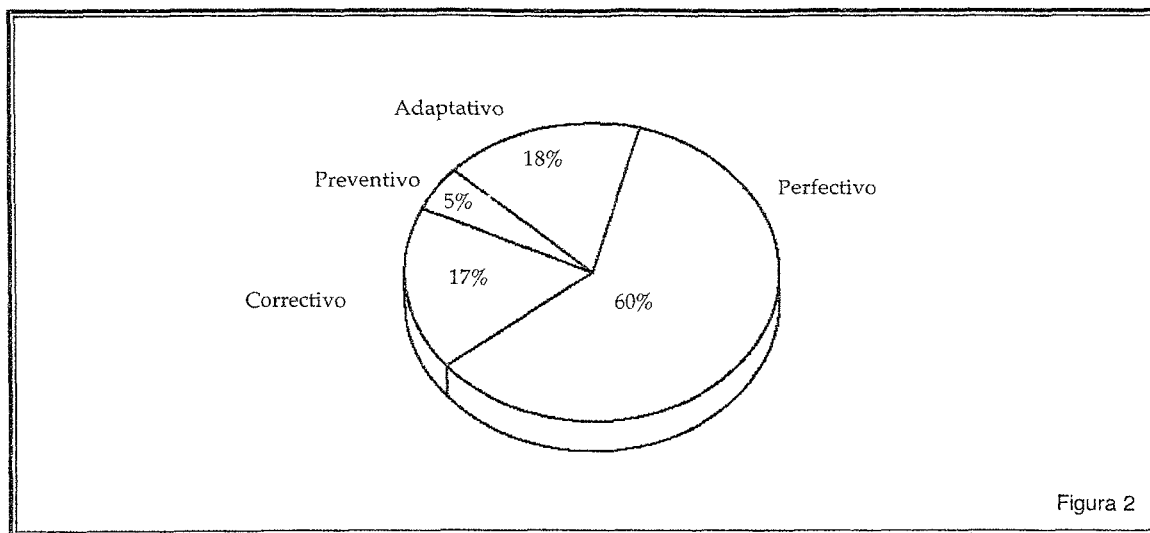
En función de la causa a la que se deban las modificaciones del software, se distinguen los diversos tipos o categorías de mantenimiento del software. Dependiendo del autor de que se trate, aparecen

distintos criterios de clasificación, si bien, cualquiera que se haga, siempre abarca las mismas posibilidades de mantenimiento del software.

Las clasificaciones de los tipos de mantenimiento más usuales son las que se exponen en la tabla que figura a continuación:

AUTOR	TIPOS DE MANTENIMIENTO
MÉTRICA v.3	<ol style="list-style-type: none"> 1. Correctivo: abarca aquellos cambios precisos para corregir errores del producto software. 2. Adaptativo: son las modificaciones que afectan a los entornos en los que el sistema opera. Por ejemplo, cambios de configuración del hardware, software de base, gestores de base de datos, comunicaciones, etc. 3. Evolutivo: son las incorporaciones, modificaciones y eliminaciones necesarias en un producto software para cubrir la expansión o cambio en las necesidades del usuario. 4. Perfectivo: son las acciones llevadas a cabo para mejorar la calidad interna de los sistemas en cualquiera de sus aspectos: reestructuración del código, definición más clara del sistema y optimización del rendimiento y eficiencia.
BENNET, LIENTZ, SWANSON	<ol style="list-style-type: none"> 1. Correctivo: abarca todas aquellas modificaciones que impliquen la corrección de un error. 2. Adaptativo: son las modificaciones llevadas a cabo como resultado de cambios producidos en el entorno externo. 3. Perfectivo: es consecuencia de cambios en los requisitos de los usuarios. 4. Preventivo: es la realización del mantenimiento anticipándose a problemas futuros.
PRESSMAN	<ol style="list-style-type: none"> 1. Correctivo: abarca todas aquellas modificaciones que impliquen la corrección de errores no descubiertos. 2. Adaptativo: son las modificaciones debidas a los cambios inherentes a cualquier aspecto de la informática, principalmente, las llevadas a cabo como resultado de cambios producidos en el entorno externo. 3. Perfectivo: es consecuencia de cambios en los requisitos de los usuarios, e incluye también el mantenimiento que se realiza para proporcionar una base mejor para futuras mejoras.

Dada la diversidad de denominaciones a la hora de nombrar los distintos tipos de mantenimiento, tomando como referencia la clasificación de Bennet, Lientz y Swanson, distintos estudios demuestran que la causa principal de la evolución y el mantenimiento del software es la incorporación de nuevos requerimientos de usuario, esto es, el mantenimiento perfectivo, que consume en torno al 60 por 100 del total de los recursos dedicados al mantenimiento. Por su parte, tal como se refleja en la siguiente figura, esos mismos estudios indican que el mantenimiento correctivo consume aproximadamente el 17 por 100 de los recursos y el adaptativo, el 18 por 100.



3.1. EL MANTENIMIENTO CORRECTIVO.

Este tipo de mantenimiento incluye todo lo concerniente a la corrección de errores o defectos en un sistema existente. Cualquier funcionamiento incorrecto del sistema respecto a su definición inicial, ya sea por un error del software, del hardware o de la documentación, debe ser corregido mediante el mantenimiento correctivo.

Algunos de los errores que provocan este tipo de mantenimiento son: fallos en la codificación de los programas, resultados no coincidentes con las especificaciones iniciales, fallos en el hardware, falta de ajuste de la documentación, requisitos y diseño con el software desarrollado, confusa documentación de usuario, etc. La mayoría de los problemas anteriores se deben, como es lógico, a errores en las especificaciones de requisitos, errores en el diseño y errores de codificación.

Un problema asociado al mantenimiento correctivo es que la corrección de errores introduce nuevos problemas, en concreto, el incremento de las pruebas y el que la corrección de un defecto tiene una gran probabilidad de introducir otro defecto, probabilidad que es directamente proporcional al número de líneas de código implicadas en el cambio.

3.2. LOS MANTENIMIENTOS ADAPTATIVO Y PERFECTIVO.

Estos tipos de mantenimiento se emplean para adaptar y mejorar las funcionalidades del software, hardware y su documentación ante nuevos marcos cambiantes. Ambos comienzan cuando se solicita una petición de cambio de adaptación o mejora, e implica una revisión del sistema completo. Esto quiere decir que las fases de estos tipos de mantenimiento son, en esencia, igual que el ciclo de vida de un nuevo desarrollo.

Los beneficios que se derivan de estos tipos de mantenimiento son, principalmente, el aumento del rendimiento del sistema, la mejora de la productividad y, sobre todo, el aumento de la satisfacción del cliente o usuario. Algunos factores que pueden inducir a realizar, según el caso, mantenimiento adaptativo o mantenimiento perfectivo, son las necesidades internas, la competitividad y los requisitos externos. Ahora bien, hay que tener en cuenta que el integrar cambios en un sistema existente puede resultar en algunos casos más laborioso que desarrollar por completo un nuevo diseño, por lo que se habrán de analizar y evaluar detalladamente los mismos.

3.3. EL MANTENIMIENTO PREVENTIVO.

Este tipo de mantenimiento se ocupa fundamentalmente de refinar la calidad del software o la de la documentación. Para ello se emplean una serie de técnicas como la Ingeniería Inversa y la Reingeniería, que serán objeto de estudio más adelante.

El ciclo del mantenimiento preventivo difiere en su arranque de los ciclos de los otros tipos de mantenimiento. Mientras que éstos arrancan tras la llegada de peticiones de mejora o corrección, el ciclo del mantenimiento preventivo lo hace tras un estudio de posibilidades de mejora en los diferentes módulos de un sistema.

El mantenimiento preventivo consta de dos procesos:

1. La identificación de los módulos candidatos a ser mejorados, antes de que sea planificada la nueva versión.
2. La correcta identificación de los problemas de calidad para maximizar los beneficios.

Entre los beneficios que aporta este tipo de mantenimiento, cabe citar: la reducción de la exposición al riesgo, la reducción de los costes de mantenimiento, la disposición de más tiempo para mejoras y nuevos desarrollos y la mejora de la mantenibilidad del sistema.

4. LA MANTENIBILIDAD DEL SOFTWARE.

La mantenibilidad del software o, si se quiere, la facilidad de mantenimiento, se define como el grado de posibilidad de comprender, corregir, adaptar y/o mejorar el software.

La importancia de la mantenibilidad es tal, que se convierte en un objetivo clave que guía el uso de cualquier metodología de Ingeniería del Software.

Los dos aspectos fundamentales en el estudio de la facilidad de mantenimiento del software son los factores que la afectan y la cuantificación de la misma.

Los factores que afectan directamente a la facilidad de mantenimiento del software reflejan, por una parte, las características de los recursos hardware y software que se utilizan durante el desarrollo y, por otra, indican la necesidad de la estandarización de los métodos, de los recursos y de los enfoques. Los principales factores son los siguientes:

- Disponibilidad de una plantilla de software cualificada.
- Estructura comprensible del sistema y facilidad de uso del mismo.
- Uso de lenguajes de programación y de sistemas operativos estandarizados.
- Disponibilidad de facilidades de depuración de código.
- Disponibilidad de casos de prueba.
- Estructura estandarizada de la documentación.
- Disponibilidad del grupo de personas que hayan desarrollado originalmente el software.

Dado que la facilidad de mantenimiento debe ser una característica esencial de cualquier software, los factores anteriores deben ser incorporados durante la fase de desarrollo. Concretando más:

- a) Durante la revisión de los requisitos hay que anotar las áreas de futuras mejoras, discutir la portabilidad del software, y considerar las interfaces del sistema que puedan tener impacto sobre el mantenimiento.
- b) Durante las revisiones del diseño debe evaluarse el diseño de datos, el arquitectónico y el procedimental, para que sea fácil de modificar, sea modular y funcionalmente independiente.
- c) En las revisiones de código se debe cuidar el estilo y la documentación interna.
- d) Finalmente, cada paso de prueba debe proporcionar documentación sobre las partes del programa que puedan necesitar un mantenimiento de prevención, antes de que el software sea lanzado formalmente.

En cuanto a la medida de la facilidad de mantenimiento, al igual que sucede con la calidad o con la fiabilidad, es un término difícil de cuantificar. No obstante, se puede medir indirectamente la mantenibilidad del software considerando un conjunto de atributos que afectan a la actividad de mantenimiento y que se pueden medir directamente. Así, existen un conjunto de métricas relacionadas con el esfuerzo gastado durante el mantenimiento, a través de las cuales se puede evaluar la mantenibilidad. Algunas de estas métricas son:

- Tiempo de reconocimiento del problema.
- Tiempo de recolección de herramientas de mantenimiento.
- Tiempo de análisis del problema.
- Tiempo de especificación de los cambios.
- Tiempo de corrección o modificación.
- Tiempo de prueba local y de prueba global.
- Tiempo de revisión del mantenimiento.
- Tiempo total de recuperación.

Además de estas medidas orientadas al tiempo, se pueden considerar medidas de la estructura del diseño y métricas de la complejidad del software.

5. LA GESTIÓN DEL MANTENIMIENTO DEL SOFTWARE.

Todo sistema está sujeto a cambios, tanto durante su desarrollo como cuando ya está en explotación. La gestión de estos cambios y su impacto en la configuración inicial del sistema es lo que abarca la «Gestión de la Configuración», que estudiaremos detalladamente en el siguiente tema. Íntimamente relacionada con ella y parte fundamental de la misma es la Gestión del Mantenimiento del Software, que tratamos seguidamente.

Las empresas son cada vez más conscientes del tiempo y los recursos que consume el mantenimiento, por lo que su planificación y gestión en general comienza a ser un factor importante para la

productividad de los sistemas de información. Las tareas de mantenimiento del software comienzan mucho antes de que se haga una petición de mantenimiento. Inicialmente se debe establecer una organización de mantenimiento, se deben definir procedimientos de evaluación y de información, y se debe especificar una secuencia estandarizada de sucesos para cada petición de mantenimiento. Además, se debe establecer un sistema de registro de información de las actividades de mantenimiento y definir criterios de revisión y de evaluación.

La siguiente figura representa, de forma esquemática, la organización del mantenimiento.

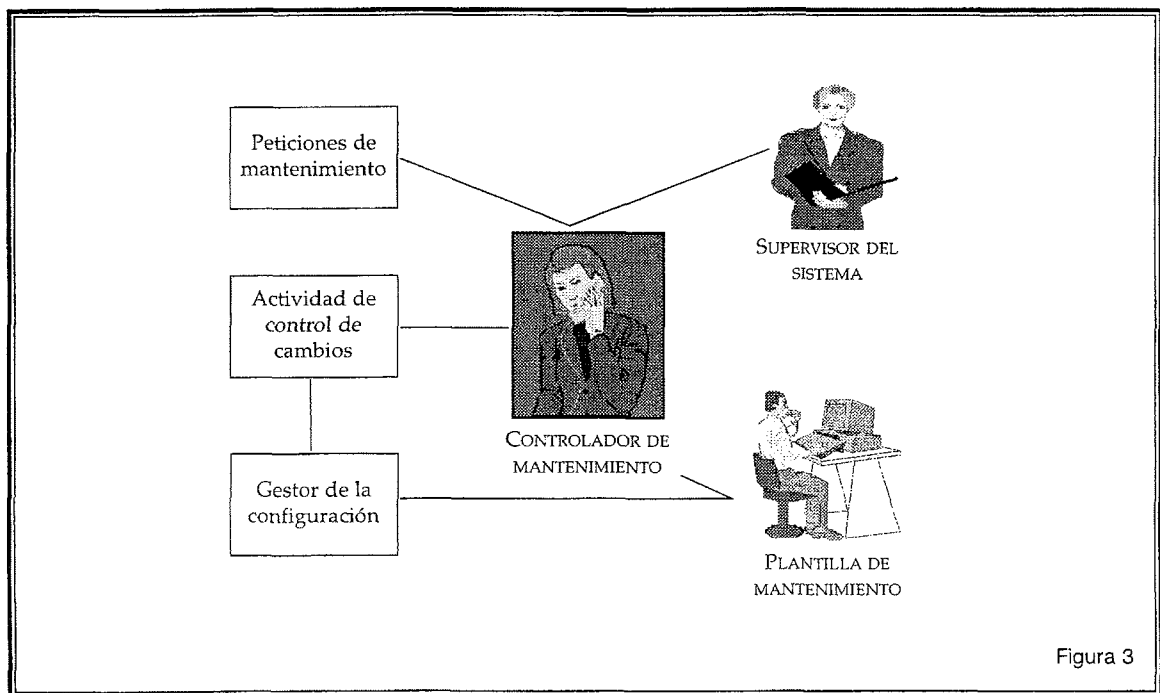


Figura 3

Las peticiones de mantenimiento se dirigen hacia un controlador de mantenimiento, que remite cada petición a un supervisor del sistema para que la evalúe. Una vez hecha la evaluación, una autoridad de control de cambios (consejo de control de cambios) debe determinar las acciones a llevar a cabo. (Cada uno de los puestos de trabajo representados en la figura anterior sirve para establecer un área de responsabilidad en el mantenimiento, evitándose así la confusión).

Todas las peticiones de mantenimiento deben ser presentadas de forma estandarizada mediante un formulario de petición de mantenimiento, que es un documento generado externamente y que se usa como base para la planificación de las tareas de mantenimiento. Internamente, la organización de software desarrollará un informe de cambios en el software indicando: el esfuerzo requerido para satisfacer el formulario de petición de mantenimiento, la naturaleza de las modificaciones solicitadas y la prioridad de la petición.

La información relativa al desarrollo de las actividades de mantenimiento debe ser convenientemente registrada para poder ser evaluada y obtener así determinadas medidas del rendimiento del mantenimiento.

Otro aspecto que afecta al mantenimiento y su gestión es el mantenimiento de código ajeno, llamado así por referirse a programas que han sido desarrollados hace muchos años (diez o más) y en los

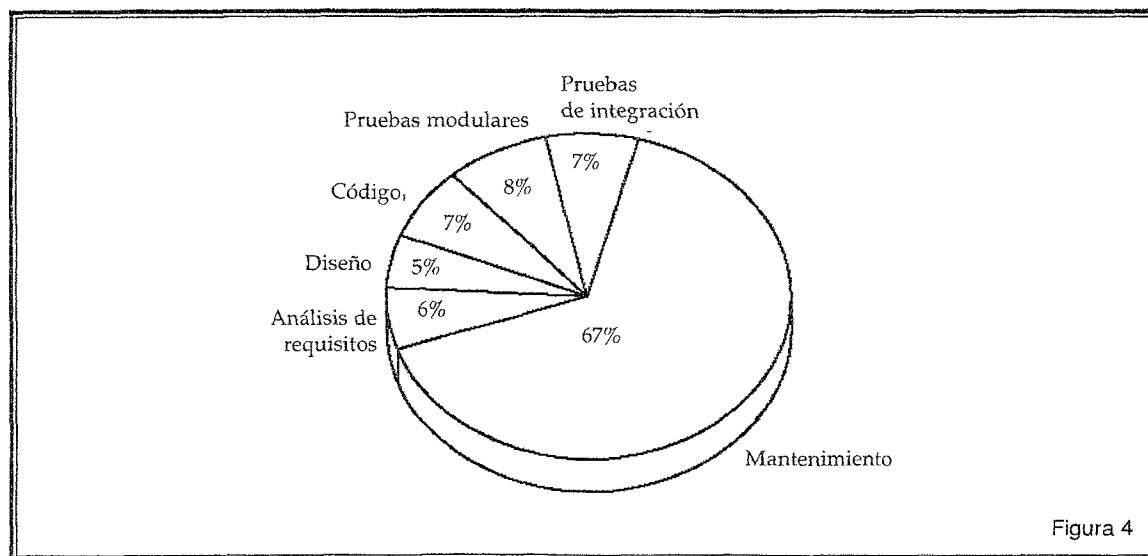
que los miembros del equipo técnico actual no trabajaron en su desarrollo. Normalmente, para estos programas la documentación es incompleta y no suele existir un registro de los cambios anteriores. Ante esto, ¿qué se puede hacer con el código ajeno? Evidentemente, no es nada fácil su mantenimiento, no obstante, Yourdon hace algunas sugerencias muy útiles para estos casos, como, por ejemplo, estudiar antes el programa, familiarizarse con su flujo de control general, evaluar las características de la documentación existente, no eliminar el código hasta estar totalmente seguros, insertar comprobaciones de errores, etc.

Finalmente, es importante tener en cuenta los efectos secundarios del mantenimiento, puesto que cada vez que se modifica el software, la posibilidad de error aumenta. A estos efectos, Freedman y Weinberg definen tres categorías de efectos secundarios:

1. Efectos secundarios sobre el código, que van desde los pequeños errores que se detectan y median durante las pruebas de regresión, hasta los problemas que pueden hacer que falle la operación del software.
2. Efectos secundarios sobre los datos, ya que durante el mantenimiento es corriente que se hagan cambios sobre determinados elementos de una estructura de datos o sobre la propia estructura en sí, y, en estos casos, puede ocurrir que el diseño del software no cuadre con los datos y aparezcan errores.
3. Efectos secundarios sobre la documentación, que se dan cuando no se reflejan los cambios del código fuente en la documentación del diseño y en los manuales de usuario. Estos efectos se reducen revisando la configuración entera del software antes de lanzar una nueva versión.

6. EL COSTE DE MANTENIMIENTO DEL SOFTWARE.

El mantenimiento representa el porcentaje más alto del coste de todo el ciclo de vida de un sistema. El coste de mantenimiento del software ha crecido rápidamente durante los últimos años. De suponer entre el 35 y el 40 por 100 del presupuesto de sistemas de información de una organización en los años 70, se ha pasado a cifras próximas al 70 por 100, por lo que es necesario tomar medidas muy serias a efectos de su disminución.



Si los cambios se pudieran anticipar en el diseño, se podrían prever; pero el problema es que el elevado número de cambios que el software soporta a lo largo de su ciclo de vida, hace imposible esta previsión. Por tanto, el mantenimiento debe tener una consideración especial porque: consume una gran parte del coste total del ciclo de vida del software; imposibilita un cambio rápido y fiable en el software haciendo perder oportunidades de negocio; dificulta la adopción de nuevas tecnologías; y ocasiona daños en la integridad del sistema.

Así pues, los costes de mantenimiento del software son los más cuantiosos del desarrollo y uso de un sistema, y además, son muy difíciles de estimar, ya que estos costes están relacionados con varios factores relativamente impredecibles y que no tienen ninguna relación con ninguna característica técnica del sistema de software. Los costes monetarios son, evidentemente, lo que más interesa; sin embargo, existen otros costes menos tangibles que también pueden ser causa de problemas. Dentro de estos «costes intangibles» podemos citar:

- La insatisfacción de un cliente o usuario cuando una petición lógica de modificación no puede ser atendida en un plazo razonable de tiempo.
- La disminución de la calidad global del software debido a los errores latentes que introducen los cambios en el mismo.
- Los trastornos en otros esfuerzos de desarrollo al tener que dedicar una parte importante de la plantilla a trabajar en tareas de mantenimiento.

El coste de mantenimiento del software se ve afectado por una serie de factores, tanto directamente relacionados con el propio software, como ajenos al mismo. Los principales son los siguientes:

A) Factores relacionados con el software:

- La interdependencia entre los módulos.
- El estilo y el lenguaje de programación.
- La calidad y cantidad de la documentación del programa.

B) Factores ajenos al software:

- Las especificaciones funcionales de los usuarios.
- La estabilidad del personal dedicado al proyecto.
- La dependencia del programa de su entorno de desarrollo.
- La estabilidad del hardware.

6.1. ESTIMACIÓN DE LOS COSTES DE MANTENIMIENTO DEL SOFTWARE.

Aunque existen varias técnicas para estimar el coste de mantenimiento, como el Método del Rectángulo de Probabilidades de Jones, sólo nos referiremos al método de Boehm o Modelo Cocomo de estimación de costes de mantenimiento.

A partir de los datos de 63 proyectos pertenecientes a varias áreas de aplicación, Boehm estableció una fórmula para estimar los costes de mantenimiento, en función de una cantidad llamada «Tráfico de Cambio Anual» (TCA), que se define como: La fracción de instrucciones fuente de un producto software, que sufren modificaciones durante un año, ya sea por adición o por alteración. El método de Boehm utiliza el TCA y el esfuerzo de desarrollo, estimado o real, en personas-mes, para hallar el esfuerzo de mantenimiento anual (EMA). La expresión matemática es:

$$EMA = t \times TCA \times E$$

donde: EMA = Esfuerzo de mantenimiento anual (en Personas-mes)

t = Número de años (normalmente, uno)

TCA = Tráfico de cambio anual.

E = Esfuerzo de desarrollo del software (en Personas-mes)

Por ejemplo: Un proyecto SW necesitó un esfuerzo de desarrollo de 300 personas-mes. Se estima que el 20 por 100 del código podría verse alterado en un año. Estimar el Esfuerzo de mantenimiento anual.

Aplicando la fórmula de Boehm (Modelo COCOMO para estimar costes de mantenimiento) se tiene: $EMA = t \times TCA \times E = 1 \times 0,20 \times 300 = 60$ personas-mes.

Otro ejemplo. A un proyecto de 32.000 LDC se le añaden, durante el primer año de mantenimiento, 4.000 LDC y se le modifican 2.400 LDC. Sabiendo que el Esfuerzo de desarrollo del proyecto resultó ser de 250 personas-mes, hallar el Esfuerzo de Mantenimiento.

Lo primero será calcular el Tráfico de cambio anual.

$$TCA = \frac{LDC \text{ añadidas} + LDC \text{ modificadas}}{LDC \text{ proyecto}}$$

Es decir, $TCA = (4.000 + 2.400) / 32.000 = 0,20$

Y entonces, $EMA = t \times TCA \times E = 1 \times 0,20 \times 250 = 50$ personas-mes.

7. INGENIERÍA INVERSA Y REINGENIERÍA.

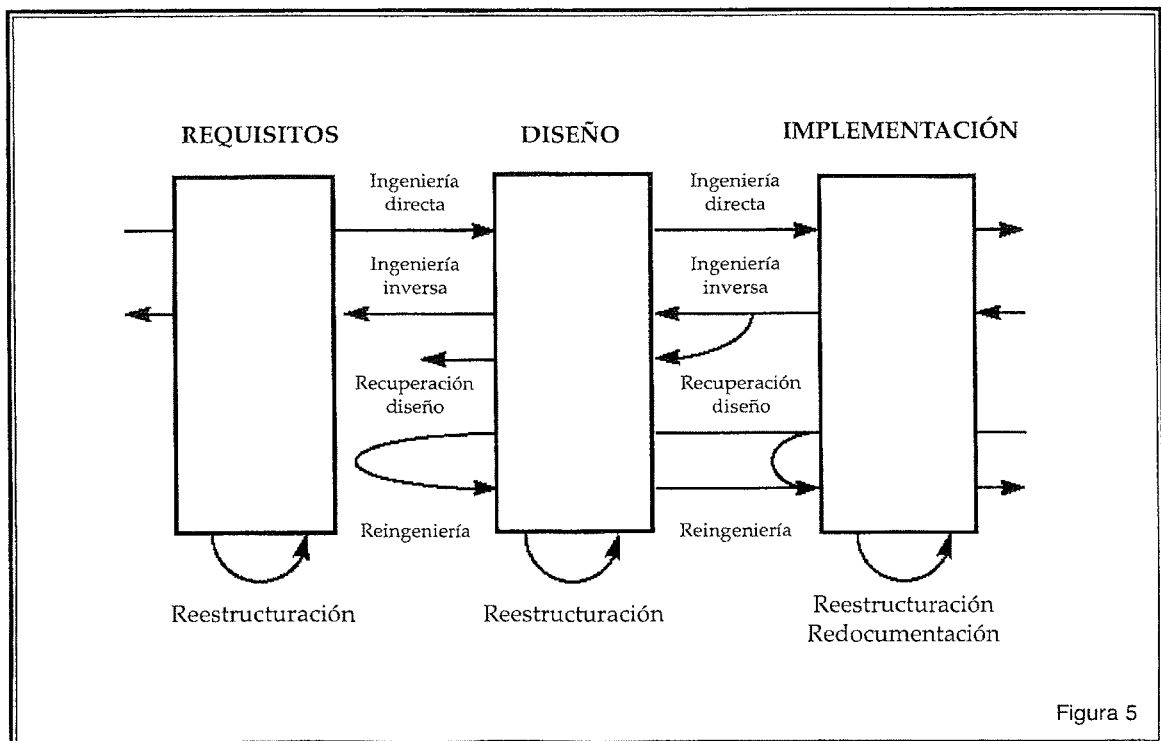
Las técnicas de mantenimiento clásico tienden a moverse a los niveles de abstracción más bajos del sistema, por lo que para cubrir la necesidad de una evolución más amplia del sistema surge la reingeniería. La reingeniería trabaja todos los niveles de abstracción (desde la implementación hasta el diseño) para conseguir realizar cambios más drásticos preservando los valores de negocio del sistema.

La reingeniería de sistemas puede clasificarse según los niveles de conocimientos requeridos para llevar a cabo el proyecto. La reingeniería que requiere conocimientos a bajos niveles de abstracción (código fuente) se llama ingeniería inversa o modernización de caja blanca y aquella que sólo requiere el conocimiento de las interfaces del sistema se llama reingeniería propiamente dicha o modernización de caja negra.

No obstante, a fin de fijar una terminología, hasta no hace mucho imprecisa y muy dependiente del organismo o autor que tratara el tema, vamos a definir primeramente una serie de conceptos siguiendo la terminología adoptada por el IEEE.

1. Ingeniería Inversa (Reverse Engineering): es el proceso de analizar un sistema para identificar los componentes y las interrelaciones entre ellos, creando representaciones del sistema en otra forma distinta a la original o bien a un nivel superior de abstracción.
2. Reingeniería (Reengineering): es el examen y modificación de un sistema para ser reconstruido de una forma nueva y además realizar la implantación derivada de esa nueva forma. La Reingeniería normalmente incluye alguna forma de Ingeniería Inversa y va seguida de alguna forma de Ingeniería «hacia adelante» o también de una Reestructuración.
3. Reestructuración (Restructuring): es la transformación de una forma de representación del sistema en otra distinta, pero del mismo nivel de abstracción, sin modificar el comportamiento externo del sistema.
4. Ingeniería Hacia Adelante (Forward Engineering): es el proceso que va desde un alto nivel de abstracción, que es independiente de la implementación concreta, hasta la propia implementación física del sistema. Es decir, es la Ingeniería del Software en su vertiente restringida al nuevo desarrollo.
5. Reingeniería de Empresas (Business Process Reengineering): es la aplicación del concepto de Reingeniería al campo económico, y se desarrolla alrededor de tres actividades clave: Rediseñando los procesos básicos de trabajo para alcanzar los objetivos del negocio; utilizando las nuevas tecnologías para concebir, diseñar y poner en marcha nuevas actividades; y cambiando la forma en la que trabajan los empleados.

La siguiente figura muestra las relaciones entre los términos asociados con la reingeniería.



7.1. INGENIERÍA INVERSA.

Generalmente, muchos de los problemas de evolución aparecen en sistemas complejos con un número elevado de líneas de código y escasa o nula documentación. Ante la necesidad de recuperación del diseño, la arquitectura y la funcionalidad del sistema desde el código para modernizar el lenguaje de programación o el soporte de los datos, surge la ingeniería inversa, la cual se ocupa de estudiar un sistema de información en el orden inverso establecido en el ciclo de vida habitual; esto es, partiendo del código fuente, se trata de identificar los componentes del sistema y las relaciones existentes entre ellos. Hasta su llegada, el ciclo de vida del software era, en teoría, en una sola dirección; ahora, se puede hablar de dos direcciones: forward o hacia adelante, que es la tradicional, y downward o hacia atrás, que es la de la Ingeniería Inversa.

La Ingeniería Inversa, actualmente conocida como modernización de caja blanca (White-Box Modernization), se fundamenta en la identificación de los componentes del sistema y sus relaciones para conseguir una representación a niveles mayores de abstracción.

Chikofsky y Cross definen la Ingeniería Inversa como: el proceso de analizar un sistema para identificar sus componentes e interrelaciones entre los mismos, y crear representaciones del sistema a un alto nivel de abstracción. Esta definición implica, por tanto, la creación de una representación a un nivel más alto del examinado (diseño desde implementación), esto es, la Ingeniería Inversa es un proceso de recuperación de diseño, que implica la extracción del diseño de los elementos de un sistema existente.

La Ingeniería Inversa no es una «caja mágica» donde la entrada es un listado fuente indocumentado y no estructurado, y su salida es una documentación completa del sistema. Efectivamente, la Ingeniería Inversa puede extraer información de diseño a partir del código fuente, pero su resultado varía fuertemente en función de los siguientes elementos:

- El nivel de abstracción del proceso de Ingeniería Inversa, y de las herramientas que se usen.
- La completud del proceso, es decir, el nivel de detalle que proporcione la Ingeniería Inversa en el nivel de abstracción.
- La interactividad o grado en el que la persona se integra con las herramientas automáticas para crear un proceso efectivo de Ingeniería Inversa.
- La direccionalidad del proceso, que si es en los dos sentidos, la información extraída del código fuente alimenta una herramienta de Reingeniería que permite reestructurar o regenerar el programa antiguo.

La Ingeniería inversa no implica la modificación del sistema, ni la generación de nuevos sistemas; sin embargo, existen una serie de técnicas intrínsecamente relacionadas con ella, que pueden considerarse como distintas formas de Ingeniería Inversa. Éstas son las siguientes:

1. Redocumentación: es la producción de una representación semántica de un sistema a cualquier nivel de abstracción que se requiera. Las herramientas usadas parten del código fuente existente, para producir diagramas de flujo de datos, modelos de datos, etc.
2. Recuperación del diseño: es el proceso mediante el que se realiza una aproximación al diseño del sistema final.

3. Reestructuración: es la transformación de un sistema en otro sin alterar su significado o funcionalidad.
4. Reingeniería: es el proceso de Ingeniería Inversa a un sistema en un determinado nivel de abstracción y, a partir de los resultados obtenidos, reconstruir el sistema mediante ingeniería hacia adelante. La Reingeniería no sólo recupera la información de diseño de un software existente, sino que usa ésta para alterar o reconstruir el sistema existente, en un esfuerzo por mejorar la calidad general.

7.2. OBJETIVOS Y BENEFICIOS DE LA INGENIERÍA INVERSA.

El objetivo primordial de la Ingeniería Inversa es proporcionar una base para el mantenimiento y futuros desarrollos. Este objetivo general se puede traducir en los siguientes objetivos parciales: facilitar la reutilización; proporcionar documentación que no existe, o actualizar la existente; migrar a otra plataforma hardware o software, cuando sea necesario; y llevar el sistema bajo el control de un entorno CASE.

A la vista de estos objetivos, los sistemas candidatos a aplicarles la Ingeniería Inversa reúnen algunas de las siguientes características:

- Las especificaciones de diseño y la documentación, no existen o están incompletas.
- El código no es estructurado.
- El sistema necesita un excesivo mantenimiento correctivo.
- Algunos módulos se han hecho muy complejos debido a los sucesivos cambios realizados.
- Se necesita una migración hacia una nueva plataforma de hardware o de software.

Ahora bien, aplicar la Ingeniería Inversa supone un enorme esfuerzo y, por tanto, se hace necesario evaluar exhaustivamente y siendo muy realistas, los casos en los que es rentable su aplicación. A estos efectos, algunos aspectos que se deben considerar son los siguientes:

- Algunos programas no se usan con mucha frecuencia y no es probable que vayan a cambiar.
- La disponibilidad de herramientas CASE apropiadas para Ingeniería Inversa.
- El coste en esfuerzo y en dinero puede ser prohibitivo.

No obstante, la Ingeniería Inversa produce una serie de beneficios, que también deben ser considerados. Los más importantes son los siguientes:

- Disminuye los costes del mantenimiento, al reducir el tiempo dedicado a entender el software existente para, a su vez, entender el cambio que se desea realizar y para facilitar la solución del problema.
- Mejora la calidad del software, ya que los sistemas candidatos a aplicarles Ingeniería Inversa son, en general, sistemas de baja calidad.

- Aporta una mayor ventaja competitiva, dado que al facilitar el mantenimiento, permite un mayor poder de reacción ante los cambios solicitados por los usuarios.

7.3. REINGENIERÍA.

La Reingeniería engloba un gran conjunto de actividades y estrategias tanto para la reducción del esfuerzo de mantenimiento como para la reutilización. Estas actividades están relacionadas con: las mejoras del software; la comprensión del software; y la captura, conservación y extensión del conocimiento del software.

Basándose en estas ideas, Chikofsky define la reingeniería como el «examen y alteración de un sistema para reconstruirlo de una nueva forma y la subsiguiente implementación de esta nueva forma». Arnold, por su parte, señala que reingeniería es «cualquier actividad que mejore nuestro entendimiento sobre el software y prepare o mejore el propio software, normalmente para su facilidad de mantenimiento, reutilización o evolución».

La importancia de las técnicas de reingeniería del software estriban en que reducen los riesgos evolutivos de una organización, ayudan a las organizaciones a recuperar sus inversiones en software, hacen el software más fácilmente modificable, amplían la capacidad de las herramientas CASE y son un catalizador para la automatización del mantenimiento del software.

Por otra parte, con el paso del tiempo, se producen tres cambios en el desarrollo de sistemas que afectan a los aspectos de su evolución. Éstos son: la importancia del reflejo de la arquitectura en la documentación del sistema; la aparición del paradigma de objetos y los componentes distribuidos.

Aunque en un principio se pensó que el paradigma de objetos sería la solución al problema de los sistemas heredados, se vio pronto que los problemas se agravaron al adaptar sistemas no concebidos para este paradigma a lenguajes como el C++ haciendo mal uso de mecanismos como la encapsulación y, sobre todo, de la herencia. En definitiva, nos encontramos ante un nuevo tipo de sistemas heredados que es necesario someter a técnicas de reingeniería para su mantenimiento y evolución. Esta reingeniería orientada a objetos puede desglosarse en los siguientes pasos:

- Análisis de requerimientos que identifiquen las metas perseguidas por la reingeniería.
- Captura del modelo por medio del conocimiento del sistema heredado su documentación utilizando patrones de ingeniería inversa.
- Detección del problema con la ayuda de herramientas de inspección, métrica y software de visualización.
- Análisis del problema seleccionando la estructura que puede resolverlo.
- Reorganización seleccionando la transformación óptima para el sistema heredado.
- Propagación del cambio con metodología de reingeniería.

Para aplicar la reingeniería a estos sistemas se emplean técnicas métricas, de visualización de programas, de abstracción y remodelación del código. Tanto para reingeniería como para ingeniería inversa, se crean patrones para la resolución de problemas relacionados con estas técnicas.

En base al conocimiento del sistema, los datos, las funcionalidades y las interfases, se desarrollan nuevas técnicas de reingeniería no basadas en el conocimiento del código sino en el examen del comportamiento de las entradas y salidas del sistema desarrollando nuevos patrones de reingeniería y sentando las bases de la reingeniería basada en el concepto de encapsulado (wrapping). Idealmente, wrapping es una reingeniería en la que sólo se analizan las interfases (las entradas y salidas) del sistema heredado ignorando los detalles internos. Constituye la reingeniería de caja negra (Black-Box Reengineering). Esta solución no es aplicable siempre y a veces requiere el concurso de la ingeniería inversa para el conocimiento interno del sistema. La reingeniería basada en wrapping puede realizarse a nivel funcional, de datos o de interfase.

Si bien la Reingeniería no tiene una metodología establecida, ni siquiera un procedimiento formal a aplicar que esté generalmente admitido, sí se pueden sugerir una serie de pasos que facilitan la realización de esta tarea. Estos pasos, que constituyen un Plan de trabajo para las actividades de Reingeniería, son los siguientes:

1. Evaluación del potencial de Reingeniería. Es decir, estudiar sobre qué partes del sistema es más adecuado concentrar el esfuerzo de las modificaciones. Para ello es necesario:
 - a) Inventariar las peticiones de modificación y clasificarlas como: formales (si afectan a la forma) y funcionales (si afectan a las funciones del sistema).
 - b) Distinguir entre las oportunidades de Reingeniería y las nuevas oportunidades de desarrollo. Para ello, además del coste, se considerarán tres criterios fundamentales:
 - Funciones de los sistemas actuales que se corresponden con los sistemas propuestos.
 - Grado en que los sistemas actuales proporcionan datos a los sistemas propuestos.
 - Grado en que los sistemas actuales dependen de la tecnología utilizada para obtener las funciones.
2. Establecimiento de las herramientas adecuadas, ya que muchas de las tareas requeridas pueden automatizarse mediante el empleo de un conjunto de herramientas como: auditores de código fuente, formateadores, generadores de código, generadores de juegos de prueba, comparadores de ficheros, etc.
3. Reestructuración. Este paso, que es automatizable mediante el empleo de herramientas especializadas, consiste en sustituir unos programas por otros que, si bien realizan las mismas funciones, son más comprensibles y de mayor calidad. La finalidad fundamental de este paso es la adaptación del código fuente para facilitar su lectura.
4. Ingeniería Inversa. En el caso más frecuente, en este paso se pretende crear la documentación de un sistema ya existente. Para ello es necesario identificar los distintos componentes del sistema, sus interrelaciones y producir la documentación de acuerdo con unas determinadas reglas de normalización.
5. Reingeniería. Esta etapa consiste en el examen y alteración de un sistema para reconstruirlo en una nueva forma. Si no se dispone de la documentación adecuada del sistema objeto de la Reingeniería, habrá que realizar una Ingeniería Inversa previamente. El proceso completo es muy difícilmente automatizable, pero sí puede serlo por partes.

6. Evaluación de las consecuencias de la Reingeniería. Una vez realizada la reingeniería hay que revisar, desde la perspectiva de la Calidad del Software, el nuevo código para comprobar diversos aspectos de su estructura, documentación, comentarios, descripción de datos, interfaces, manejo de errores, consistencia y completud.

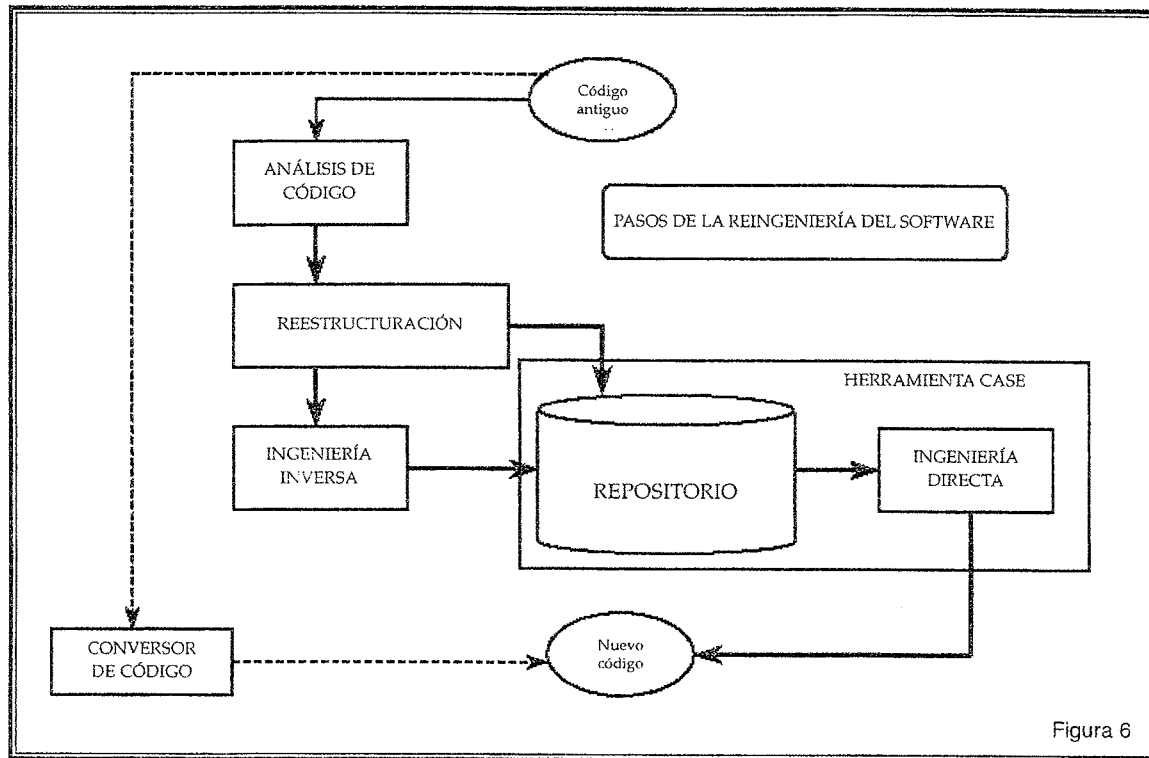


Figura 6

8. ABANDONO DEL SISTEMA.

Un sistema se abandona cuando no es capaz de seguir el ritmo de las necesidades del negocio y su reingeniería no es posible o no se puede hacer a un coste efectivo. El abandono del sistema comporta dos riesgos principales, que es una actividad que consume muchos recursos, y que abandonar el sistema no debe significar empezar un desarrollo desde cero.

Para decidir el proceso de abandono del sistema es necesario descubrir el núcleo del sistema (funcionalidades y datos) utilizando análisis para averiguar el nivel de abstracción en el que hay que moverse y qué técnica es más adecuada. A esta búsqueda de recursos valiosos del sistema heredado se la denomina mining legacy system assets, utilizando el término adapting legacy system assets para los niveles más altos de abstracción.

En general, son necesarios cuatro pasos para minar los recursos de un sistema:

1. Recogida preliminar de información.
2. Tomar decisiones con respecto a los recursos a minar y las estrategias básicas a utilizar y las opciones técnicas a aplicar.

3. Realizar un análisis técnico detallado de los componentes del sistema y sus relaciones e interfaces.
4. Llevar a cabo la rehabilitación de los recursos seleccionados.

La tarea más complicada corresponde a decidir las estrategias y las opciones técnicas que se van a aplicar a los recursos seleccionados. OAR (Option Analysis for Reengineering) proporciona ayuda para tomar dos decisiones fundamentales: determinar el nivel de análisis adecuado al problema en los niveles de código, funcional y de arquitectura; y determinar la estrategia de reingeniería a aplicar.

OAR utiliza el modelo de herradura (horseshoe) para establecer el contexto donde tomar estas decisiones. En el modelo se representan tres niveles de abstracción:

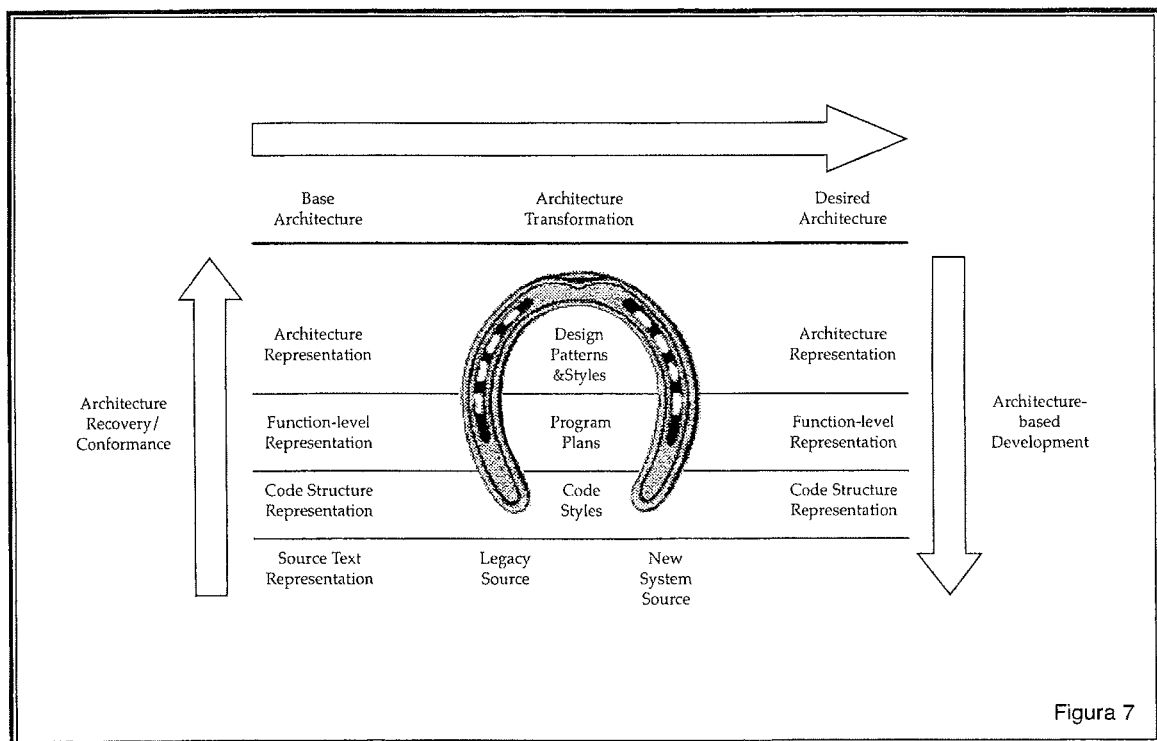


Figura 7

- Representación a nivel de código-estructura, que contiene código fuente y artefactos como árboles de sintaxis y gráficos de ejecución. Dentro de este nivel existen dos subniveles: cadenas de texto y estructuras de código.
- Representación a nivel de función, que describe las relaciones entre programas (llamadas entre ellos), datos (relación entre funciones y datos) y archivos (relaciones agrupadas de funciones y datos).
- Representación a nivel de concepto, que agrupa a los artefactos de los niveles de código y funciones en subsistemas.

9. EVALUACIÓN POST-IMPLEMENTACIÓN.

La revisión y evaluación de los sistemas de información implantados tiene la finalidad de detectar su comportamiento y consistencia mediante la aplicación de técnicas adecuadas que permitan observar si el sistema se está ajustando a las necesidades.

Mediante la revisión de los sistemas se implica el proceso de medir y verificar principios para determinar si el plan, la política y el sistema son los mejores.

El objetivo básico de la evaluación post-implementación es detectar las deficiencias y errores que se deberán corregir, y que pueden ser los siguientes:

- Desviación del sistema implantado.
- Aumento de operaciones de la organización.
- Cambio de políticas que afectan directa o indirectamente al sistema.
- Discrepancia en las operaciones.
- Cambio de condiciones en las que se ejecutaba o realizaba el sistema.
- Falta de preparación del personal para utilizar el sistema.

Por otra parte, debido a la gran diversidad y disparidad de sistemas existentes, se hace necesario disponer de una guía para realizar la evaluación y poder detectar las deficiencias anteriormente señaladas.

La Teoría de Colas es una formulación matemática que permite evaluar muchos tipos de sistemas y su optimización. Esta teoría se aplica a sistemas en los que interactúan dos procesos normalmente aleatorios, un proceso de llegada de clientes y un proceso de servicio de clientes, en los que existen fenómenos de acumulación de clientes en espera de servicio y donde existen reglas definidas (prioridades) para la prestación del servicio. Gran parte de los sistemas de información responden a este modelo, por lo que la Teoría de Colas es muy usada en su evaluación y optimización.

Para los problemas que no respondan a este modelo se pueden aplicar otras técnicas de evaluación y optimización como la heurística, los modelos lineales y la simulación.

BIBLIOGRAFÍA

- Ingeniería del Software. Un enfoque práctico. Roger S. Pressman. Ed. McGraw Hill.
- Software Maintenance Management. B. P. Lientz y E. B. Swanson. Ed. Addison Wesley.
- Reverse engineering and design recovery: A taxonomy. E. Chikofsky y J. Cross. Ed. IEEE Software.
- Software Reengineering. R. S. Arnold. Ed. IEEE Computer Society Press.

- The Design of a Robust Persistence Layer For Relational Databases. S. Ambler.
- Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información. Métrica versión 3. Ministerio para las Administraciones Públicas.
- Temario de las pruebas selectivas para ingreso en el Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado. ASTIC.
- Temario de las pruebas selectivas para el acceso, por promoción interna, al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado. Ministerio para las Administraciones Públicas.
- Temario del Máster en Ingeniería del Software. Facultad de Informática. Universidad Politécnica de Madrid. Ed. Centro de Estudios Financieros.



