



INGENIERÍA DEL SOFTWARE II

Tema 1

Mantenimiento del Software

Univ. Cantabria – Fac. de Ciencias

Francisco Ruiz, Macario Polo



Objetivos (i)

Adquirir una buena base conceptual, metodológica y aplicada sobre el problema del Mantenimiento del Software, la etapa más costosa del ciclo de vida de un producto software

“El efecto 2000 y la adaptación al Euro son casos paradigmáticos que demostraron la gran importancia de este proceso para la industria del software”



Objetivos (ii)

- ¿Qué haremos para conseguirlo?
 - Analizar la importancia económica del Mantenimiento del Software (MS),
 - Estudiar sus características, causas, costes y soluciones,
 - Relacionar el Mantenimiento y la Calidad de un producto software,
 - Encuadrar el Mantenimiento en el ciclo de vida del software, es decir, establecer su relación con otros procesos,
 - Conocer los principales estándares internacionales,



Objetivos (iii)

- ¿Qué haremos para conseguirlo?
 - Presentar las principales técnicas y herramientas disponibles,
 - Presentar los principales métodos de medida y estimación del mantenimiento,
 - Revisar los tipos de herramientas existentes, y
 - Analizar metodologías específicas para MS.



Contenidos

1. Introducción: definición, conceptos, tipos de mantenimiento, costes.
2. Dificultades y soluciones.
3. Modelos de calidad, mantenibilidad y mantenimiento, medida de la mantenibilidad, estándares. (en papel)
4. Estándares - ISO/IEC 14764 (en papel).
5. Soluciones técnicas.
 1. Reingeniería.
 2. Ingeniería inversa.
 3. Reestructuración.
 4. Ingeniería inversa de bases de datos
 5. Detección de clones.
6. Soluciones metodológicas y de gestión. (en papel)
 1. La metodología Agil-MANTEMA.



Bibliografía Básica

- Piattini, M. G., Ruiz, F., Polo, M., et al.
Mantenimiento del Software. Modelos, técnicas y métodos para la gestión del cambio.
Ed. Ra-Ma. Madrid, España 2000.
- Polo, M., Piattini, M. y Ruiz, F.
Advances in Software Maintenance Management: technologies and solutions.
Idea Gorup Pub. USA 2003.
- Pigoski, T. M.
Practical Software Maintenance. Best Practices for Managing Your Investment.
Ed. John Wiley & Sons. USA, 1996.
- ISO/IEC 14764
Software Engineering - Software Maintenance (draft)
ISO/IEC JTC1/SC7 Secretariat. Canadá, 1998.
- IEEE, std 1219
Standard for Software Maintenance.
IEEE Computer Society Press. USA, 1993.
- ISO/IEC 12207
Information Technology - Software life cycle processes.
ISO/IEC JTC1/SC7 Secretariat. Canadá, 1995.



Índice – Parte 1

- Introducción.
 - Ingeniería, Crisis y Mantenimiento del Software
 - El ciclo de vida del software
 - Concepto de Mantenimiento del Software
- Costes y Causas del MS
 - El efecto Iceberg: costes intangibles
 - Causas del alto coste del MS
- Tipos de Mantenimiento
 - Mantenimiento Correctivo
 - Mantenimiento Adaptativo
 - Mantenimiento Perfectivo
 - Mantenimiento Preventivo
- Actividades del MS
 - Actividades según el Tipo de Mantenimiento



Ingeniería, Crisis y Mantenimiento del Software

- Frente a la considerable velocidad con que se ha desarrollado la ingeniería de computadores (hardware), el desarrollo del software ha sufrido un **retraso histórico** en cuanto a la elaboración y disposición de un cuerpo de doctrina tecnológico (metodologías y herramientas) y científico (modelos o teorías en los que basar lo anterior).
- En 1970 ya se había popularizado el término **Crisis del Software** para referir esta situación. Los síntomas de esta crisis han estado repercutiendo desde entonces en la industria de desarrollo de software y todavía se sienten sus efectos. Para resolver el problema surgió un área de la informática que recibe el nombre de Ingeniería del Software.
- Una de las principales causas de esta situación ha sido la **poca importancia** que se le ha dado al proceso de **Mantenimiento del Software** desde todos los colectivos afectados (gestores de empresas, responsables de centros de proceso de datos, informáticos y usuarios).



La ingeniería del software

- Entre las numerosas definiciones de ingeniería del software existentes en la bibliografía es interesante la formulada por McDermid [1991]: *"Ingeniería del software es la ciencia y arte de especificar, diseñar, llevar a cabo y desarrollar -con economía, prontitud y elegancia- programas, documentación y procedimientos operativos mediante los cuales los computadores pueden ser útiles para el ser humano"*.
- En esta definición llama la atención la inclusión de los aspectos artísticos (creatividad) y económicos. También refleja claramente que la ingeniería del software abarca la obtención de "productos" adicionales al código de los programas.



El ciclo de vida del software

- La complejidad del proceso de producción de software se intenta abordar mediante la descomposición en diversas etapas. Esta descomposición ha recibido el nombre de Ciclo de Vida del Software. Los diversos modelos de ciclo de vida que han sido propuestos plantean variantes a partir de las siguientes fases principales:
 - Análisis y Definición de Requisitos.
 - Especificación.
 - Diseño.
 - Programación (escritura del código).
 - Prueba e instalación.
 - Operación y **mantenimiento**.
- Las **tareas de mantenimiento** son las últimas en realizarse.



Concepto de Mantenimiento del Software

- Aun cuando son las últimas en el ciclo de vida del software, las **actividades de mantenimiento** no son las menos importantes. Muy al contrario, a continuación veremos que el mantenimiento del software se ha convertido en la principal actividad en cuanto a recursos necesarios y costes.
- Según la terminología ANSI-IEEE, el mantenimiento del software es: *“la modificación de un producto software después de su entrega al cliente o usuario para corregir defectos, para mejorar el rendimiento u otras propiedades deseables, o para adaptarlo a un cambio de entorno”*.



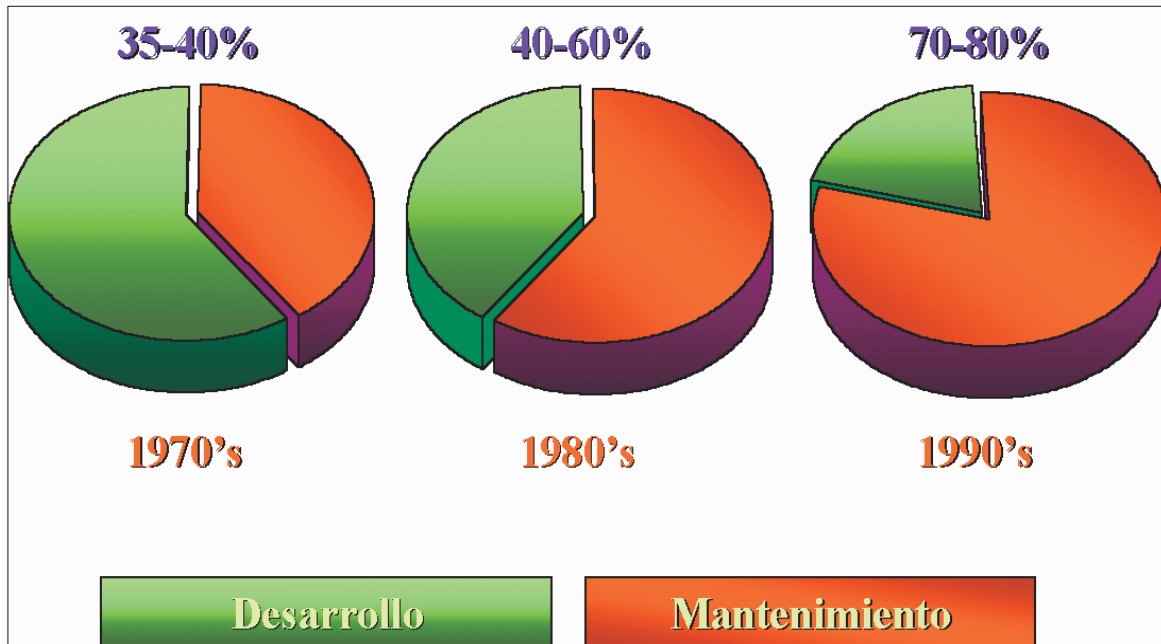
Costes del MS (i)

- Múltiples estudios señalan que el mantenimiento es la **parte más costosa** del ciclo de vida del software. Estadísticamente está comprobado que el coste de mantenimiento de un producto software a lo largo de toda su vida útil supone más del doble que los costes de su desarrollo. La tendencia es creciente con el paso del tiempo:

<u>Referencia</u>	<u>Fechas</u>	<u>% Mantenimiento</u>
[Pressman, 1993]	años 70	35%-40%
[Lientz y Swanson, 1980]	1976	60%
[Pigoski, 1997]	1980-1984	55%
[Pressman, 1993]	Años 80	60%
[Rock-Evans y Hales, 1990]	1987	67%
[Schach, 1990]	1987	67%
[Pigoski, 1997]	1985-1989	75%
[Frazer, 1992]	1990	80%
[Pressman, 1993]	Años 90	90%



Costes del MS (ii)



Costes del MS (iii)

- Existen empresas que se acercan a porcentajes del 95% de los recursos dedicados al mantenimiento, con lo cual se hace imposible el desarrollo de nuevos productos software. Esta situación se conoce como ***Barrera de Mantenimiento***.
- En general, el porcentaje de recursos necesarios para mantenimiento se incrementa a medida que se produce más software (ver figura).
- Los estudios sobre la situación del mercado comercial del MS son relativamente escasos:

El MS supone, cada vez más, un mercado importantísimo.



El efecto *Iceberg*: costes intangibles (i)

- Cuando se planifican los costes de mantenimiento, los analistas-programadores experimentados tienen la impresión de que el MS es algo **descontrolado** y que nunca se sabe qué va a pasar (es algo así como predecir el futuro). Parece como si fuese un **iceberg** del cual sólo se percibe una pequeña parte, pero bajo cuya superficie se esconde una gran cantidad de problemas potenciales y de costes encubiertos.
- En la parte sumergida de este iceberg se ocultan otros costes, menos tangibles que los monetarios, pero que pueden ser causa de muchas preocupaciones.
- Un **coste intangible** del MS se encuentra en las oportunidades de desarrollo que se han de posponer o que se pierden, debido a que los recursos disponibles están dedicados a las tareas de mantenimiento.



El efecto *Iceberg*: costes intangibles (ii)

- Otros costes intangibles son los siguientes:
 - **Insatisfacción del cliente** cuando no se puede atender en un tiempo aceptable una petición de reparación o modificación que parece razonable.
 - Los **errores ocultos** introducidos al cambiar el software durante el mantenimiento reducen la calidad global del producto.
 - **Perjuicio en otros proyectos** de desarrollo cuando la plantilla tiene que dejarlos, total o parcialmente, para atender peticiones de mantenimiento.
- En suma, un coste final del mantenimiento del software es la **reducción que se produce en la productividad de los informáticos** al iniciar el mantenimiento de aplicaciones antiguas.
- Algunos estudios han calculado **reducciones de la productividad** - medida en LDC por persona y mes - **de 40 a 1**, es decir, el coste de mantener (modificar) una línea de código puede llegar a ser 40 veces más alto que el de escribirla durante el proceso de desarrollo.



Causas del alto coste del MS (i)

- Son varias las **causas** de que en la mayoría de las organizaciones actuales se requiera mucho trabajo de mantenimiento:
 - Una gran cantidad del software que existe actualmente ha sido **desarrollado hace más de 10 años**. Aunque estos programas fuesen creados utilizando las mejores técnicas de diseño y codificación existentes en su momento (la mayoría no lo fueron), se construyeron con restricciones de tamaño y espacio de almacenamiento y se desarrollaron con herramientas tecnológicamente desfasadas.
 - Estos programas han sufrido una o varias **migraciones** a nuevas plataformas o sistemas operativos.
 - Y han experimentado **múltiples modificaciones** para mejorarlos y adaptarlos a las nuevas necesidades de los usuarios.
 - Todos estos cambios se realizaron sin tener en cuenta la arquitectura general del sistema (no se aplicaron **técnicas de ingeniería inversa** o reingeniería).

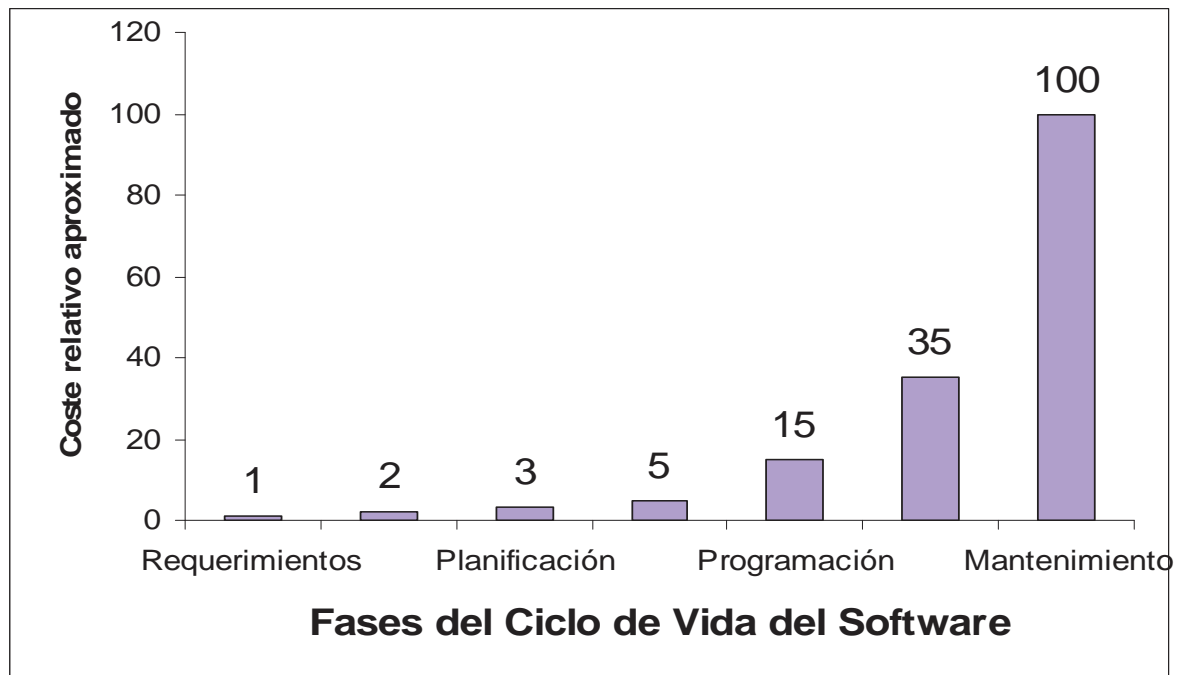


Causas del alto coste del MS (ii)

- El resultado de todo lo anterior, es la existencia de **sistemas software con una baja calidad**:
 - diseño pobre de las estructuras de datos,
 - mala codificación,
 - lógica defectuosa, y
 - documentación escasa.
- Pero que tienen que seguir funcionando, y por tanto, tienen que ser mantenidos
 - **baja calidad => mayores costes de mantenimiento.**
- Otra causa directa de los grandes costes del MS es que el **coste relativo de reparar un defecto** aumenta considerablemente en las últimas etapas del ciclo de vida del software, de forma que la relación entre el coste de detectar y reparar un defecto en la fase de análisis de requisitos y en la fase de mantenimiento es de 1 a 100 respectivamente.



Causas del alto coste del MS (iii)



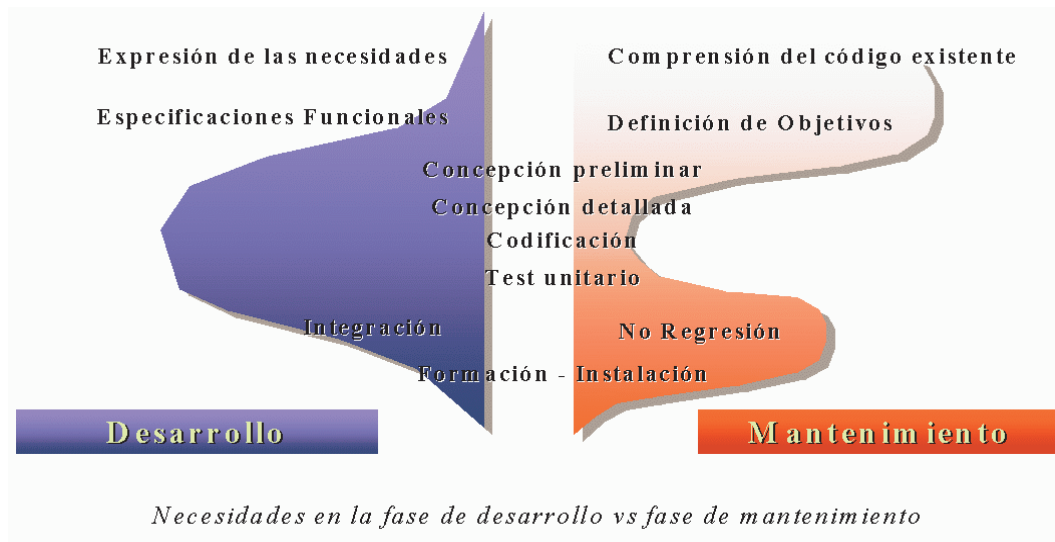
Causas del alto coste del MS (iv)

- Algunas de las razones por las que es **menos costoso detectar y corregir un error** durante las etapas iniciales del ciclo de vida que durante las etapas últimas son:
 - Es más fácil cambiar la **documentación** (por ejemplo, los documentos de especificación o de diseño) que modificar el código.
 - Un cambio durante una fase tardía puede requerir que sea modificada la documentación de todas las fases anteriores.
 - Es más fácil **encontrar un defecto** durante la fase en la cual se ha introducido el defecto que tratar de detectar y corregir los efectos provocados por el defecto en una fase posterior.
 - La causa de un defecto puede esconderse en la inexistencia o falta de actualización de los documentos de especificación o diseño.



Causas del alto coste del MS (v)

- Los costes del mantenimiento se incrementan al utilizar **técnicas y metodologías poco actas**, casi siempre pensadas para las fases previas del ciclo de vida.



Francisco Ruiz - IS2

1.21



Tipos de Mantenimiento (i)

- En la definición de mantenimiento aparecen indicados, directa o indirectamente, **cuatro tipos** de mantenimiento:
 - Corregir defectos → *correctivo*
 - Mejorar el rendimiento → *preventivo/perfectivo* u otras propiedades
 - Adaptar a un cambio de entorno → *adaptativo*
- Las definiciones de ISO e IEEE no coinciden.
- En algunas propuestas más modernas, se proponen algunas subdivisiones (metodología MANTEMA).

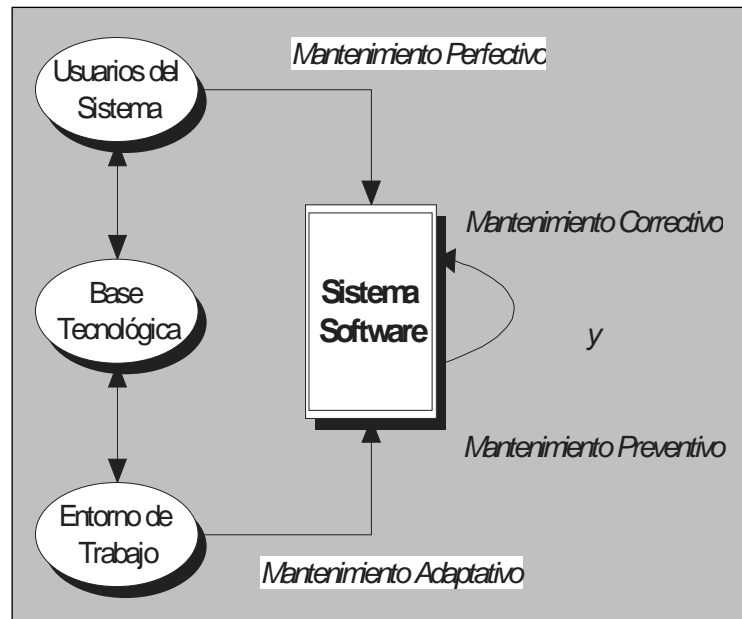
Francisco Ruiz - IS2

1.22



Tipos de Mantenimiento (ii)

- Un resumen del papel que representa cada tipo de mantenimiento aparece en la figura:
 - Mientras que el cambio tecnológico afecta indirectamente a los sistemas software, el entorno de trabajo y los usuarios lo hacen directamente, produciendo demandas de mantenimiento adaptativo y perfectivo respectivamente.



Tipos de Mantenimiento (iii)

- En **MANTEMA** se trabaja con los siguientes tipos:
 - **No Planificable (NP):**
 - **Correctivo Urgente (UC):** localizar y eliminar los posibles defectos que bloquean el programa o los procesos de funcionamiento de la empresa.
 - **Planificable (P):**
 - **Correctivo No Urgente (NUC):** localizar y eliminar los posibles defectos de los programas que no son bloqueantes.
 - **Perfectivo (PER):** añadir al software nuevas funcionalidades solicitadas por los usuarios.
 - **Adaptativo (A):** modificar el software para adaptarlo a cambios en el entorno de trabajo (hardware o software).
 - **Preventivo (PRE):** modificar el software para mejorar sus propiedades (calidad, mantenibilidad, etc.).



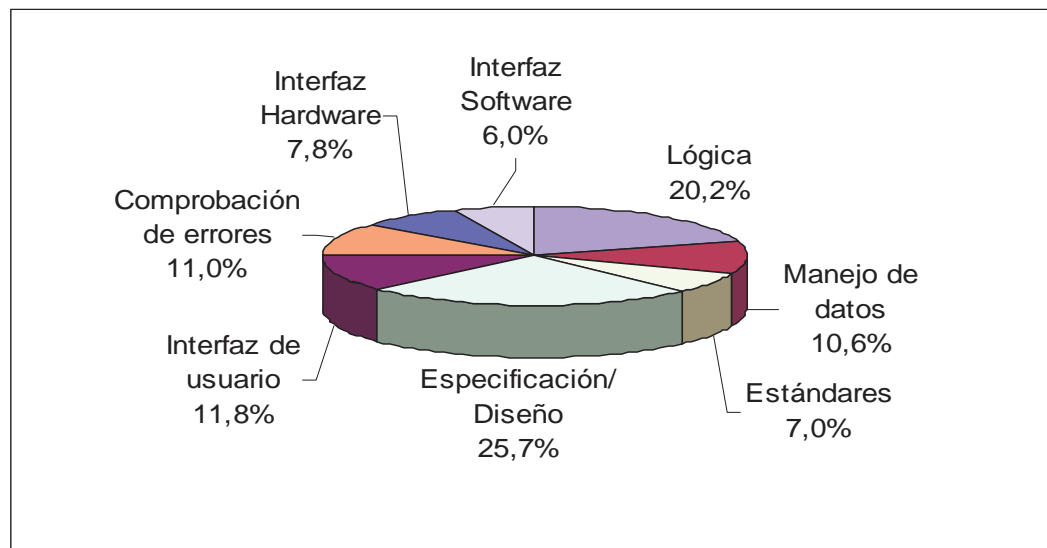
Mantenimiento Correctivo (i)

- A pesar de las pruebas y verificaciones que aparecen en etapas anteriores del ciclo de vida del software, los programas pueden tener defectos. El mantenimiento correctivo tiene por objetivo **localizar y eliminar los posibles defectos** de los programas.
- Un **defecto** en un sistema es una característica del sistema con el potencial de causar un fallo.
- Un **fallo** ocurre cuando el comportamiento de un sistema es diferente del establecido en la especificación. Entre otros, los fallos en el software pueden ser de:
 - Procesamiento, por ejemplo, salidas incorrectas de un programa.
 - Rendimiento, por ejemplo, tiempo de respuesta demasiado alto en una búsqueda de información.
 - Programación, por ejemplo, inconsistencias en el diseño de un programa.
 - Documentación, por ejemplo, inconsistencias entre la funcionalidad de un programa y el manual de usuario.



Mantenimiento Correctivo (ii)

- En la figura se muestra una distribución de las **causas de los defectos** según un estudio estadístico realizado en 1994.





Mantenimiento Adaptativo (i)

- Este tipo de mantenimiento consiste en la modificación de un programa debido a ***cambios en el entorno*** (hardware o software) en el cual se ejecuta.
- Los cambios pueden afectar a:
 - el sistema operativo (cambio a uno más moderno),
 - la arquitectura física del sistema informático (paso de una arquitectura de red de área local a Internet/Intranet),
 - o al entorno de desarrollo del software (incorporación de nuevos elementos o herramientas como ODBC).
- La envergadura del cambio necesario puede ser muy diferente: desde un pequeño retoque en la estructura de un módulo hasta tener que reescribir prácticamente todo el programa para su ejecución en un ambiente distribuido en una red.



Mantenimiento Adaptativo (ii)

- Los ***cambios en el entorno software*** pueden ser de dos clases:
 - En el entorno de los ***datos***, por ejemplo, al dejar de trabajar con un sistema de ficheros clásico y sustituirlo por un sistema de gestión de bases de datos relacionales.
 - En el entorno de los ***procesos***, por ejemplo, migrando a una nueva plataforma de desarrollo con componentes distribuidos, Java, ActiveX, etc.
- Este tipo de mantenimiento es cada vez ***más frecuente*** debido principalmente al cambio, cada vez más rápido, en los diversos aspectos de la informática: nuevas generaciones de hardware, nuevos sistemas operativos -o versiones de los antiguos-, y mejoras en los periféricos o en otros elementos del sistema (frente a esto, la vida útil de un sistema software puede superar fácilmente los diez años).



Mantenimiento Perfectivo

- **Cambios en la especificación**, normalmente debidos a cambios en los requerimientos de un producto software, implican un nuevo tipo de mantenimiento llamado perfectivo. La casuística es muy variada. Desde algo tan simple como cambiar el formato de impresión de un informe, hasta la incorporación de un nuevo módulo funcional. Podemos definir el mantenimiento perfectivo como el conjunto de actividades para **mejorar o añadir nuevas funcionalidades** requeridas por el usuario.
- Algunos autores dividen este tipo de mantenimiento en dos:
 - Mantenimiento de Ampliación: orientado a la incorporación de nuevas funcionalidades.
 - Mantenimiento de Eficiencia: que busca la mejora de la eficiencia de ejecución.



Mantenimiento Preventivo

- Este último tipo de mantenimiento consiste en la modificación del software para **mejorar las propiedades** de dicho software (por ejemplo, aumentando su calidad y/o su mantenibilidad) sin alterar sus especificaciones funcionales. Algunas maneras de hacerlo son:
 - incluir sentencias que comprueben la validez de los datos de entrada,
 - reestructurar los programas para mejorar su legibilidad, o
 - incluir nuevos comentarios que faciliten la posterior comprensión del programa.
- En algunos casos se ha planteado el Mantenimiento para la Reutilización, consistente en modificar el software (buscando y modificando componentes para incluirlos en bibliotecas) para que sea más fácilmente reutilizable. En realidad este tipo de mantenimiento es preventivo, especializado en mejorar la propiedad de reusabilidad del software.

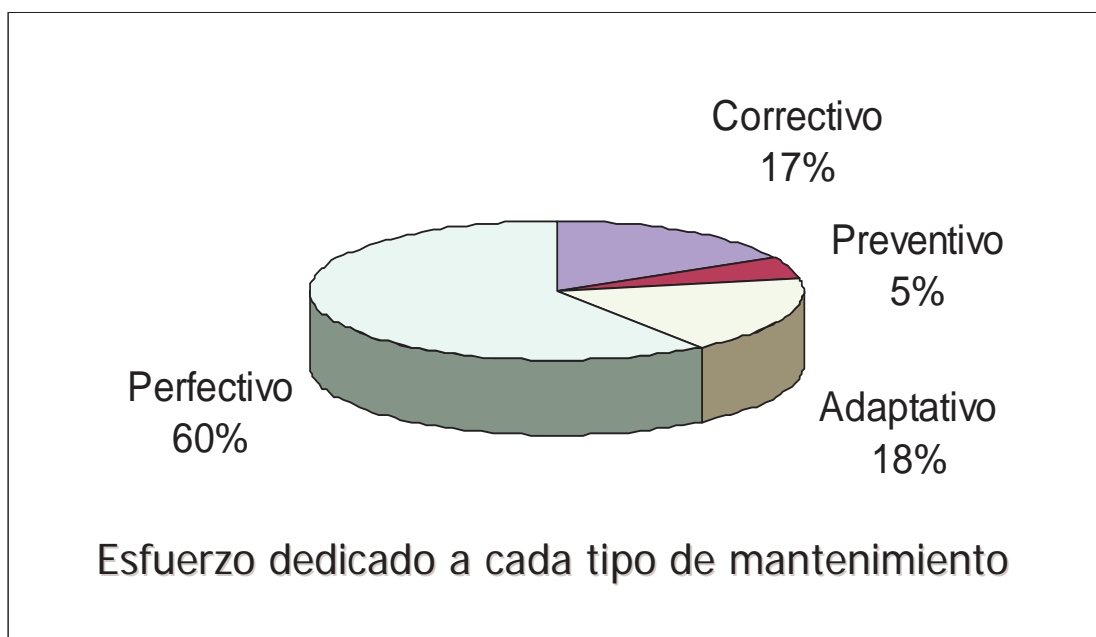


Actividades del MS (i)

- El desconocimiento de las actividades que implica el MS puede inducir a minusvalorar su importancia.
- Lo primero que se suele asociar con el MS es la corrección de errores de los programas. Por esta causa, la impresión mas generalizada entre los gestores, usuarios, e incluso entre los propios informáticos, es que la mayor parte del mantenimiento que se realiza en el mundo es de tipo correctivo.
- Sin embargo, los principales estudios realizados sobre el tema indican que esta impresión es equivocada, y establecen que ***el mantenimiento perfecto es el tipo más habitual*** (ver figura).
- El establecimiento de **analogías** entre el MS y el **mantenimiento del hardware** puede conducir a confusión, ya que el software, a diferencia del hardware, no se desgasta y, por tanto, la principal actividad asociada con el mantenimiento del hardware -reemplazar o reparar las piezas estropeadas o defectuosas- no es aplicable al software.



Actividades del MS (ii)





Actividades del MS (iii)

- Las actividades de MS se pueden agrupar en ***tres categorías*** funcionales:
 - **Comprensión del software y de los cambios a realizar:** para poder modificar un programa, los programadores necesitan conocer su funcionalidad y objetivos, su estructura interna y los requisitos de operación. De no ser así, se corre un gran riesgo de introducir nuevos defectos que en el futuro supondrán un coste de mantenimiento adicional.
 - **Modificación del software:** para incorporar los cambios necesarios se deben crear y modificar las estructuras de datos, la lógica de los procesos, las interfaces y la documentación. Los programadores deben conocer lo mejor posible las repercusiones que tienen en el sistema los cambios que están realizando, con el fin de evitar al máximo posible los efectos secundarios.
 - **Realización de pruebas:** para validar los cambios se deben realizar pruebas selectivas que nos permitan comprobar la corrección del software. Esta actividad es necesaria siempre, ya que incluso un cambio muy pequeño no verificado puede producir defectos en el software que reduzcan su calidad y fiabilidad.



Actividades del MS (iv)

- En la tabla se indican las proporciones del tiempo de mantenimiento que supone cada actividad específica según McClure [1992]:

<u>Categoría</u>	<u>Actividad</u>	<u>% Tiempo</u>
a) Comprensión del software y de los cambios a realizar	Estudiar las peticiones	18%
	Estudiar la documentación	6%
	Estudiar el código	23%
b) Modificación del software	Modificar el código	19%
	Actualizar la documentación	6%
c) Realización de pruebas	Diseñar y realizar pruebas	28%

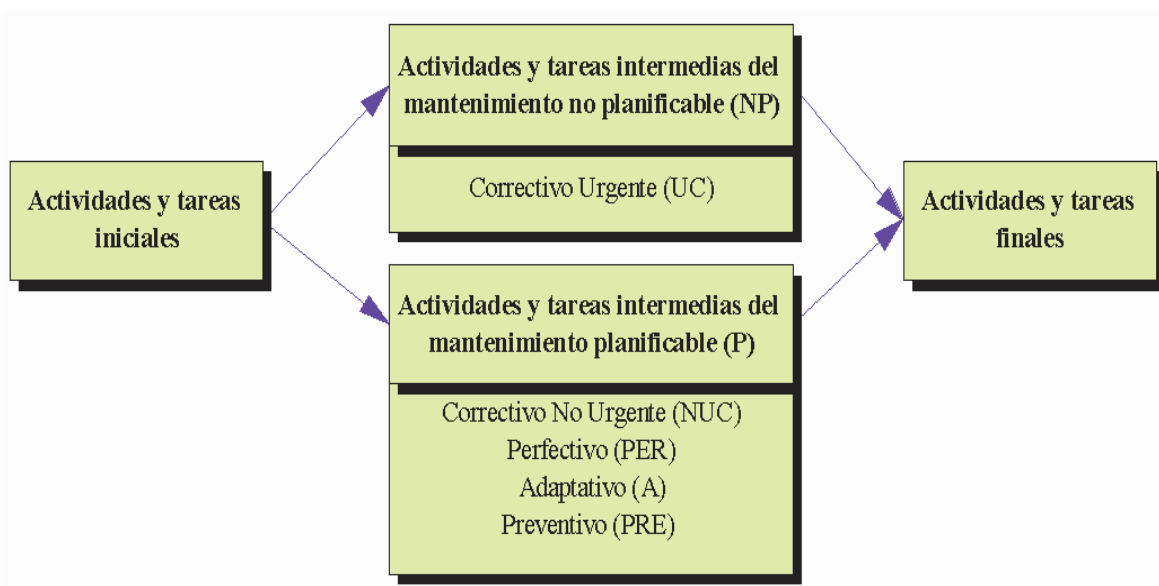


Actividades por Tipo de Mantenimiento (i)

- Según el tipo de mantenimiento, se deberán realizar un conjunto de actividades y tareas diferentes.
- El mantenimiento UC (MANTEMA) tiene diferencias claras con los demás tipos debido a que la urgencia necesaria para reparar los errores no permite realizar ninguna actividad de planificación.
- Se pueden definir subconjuntos de actividades y tareas comunes a varios tipos de mantenimiento, cumpliéndose que:
 - Existe un conjunto de actividades y tareas "*iniciales*" comunes a todos los tipos de mantenimiento.
 - Existe un conjunto de actividades y tareas " *finales*" comunes a todos los tipos de mantenimiento.
 - Las actividades "*intermedias*" de los tipos de mantenimiento P son bastante coincidentes.



Actividades por Tipo de Mantenimiento (ii)



Tipos de mantenimiento y actividades en MANTEMA



Índice - Parte 2

- Dificultades del MS
 - Código Heredado
 - Leyes del Mantenimiento del Software
 - Problemas inherentes al MS
 - Efectos secundarios
 - sobre el código
 - sobre los datos
 - sobre la documentación
- Soluciones al Problema del MS
 - de Gestión
 - Mejora de los recursos dedicados al mantenimiento
 - Gestión de la Calidad
 - Gestión Estructurada del MS
 - Organización del equipo humano
 - Documentación de los cambios
 - Soluciones Técnicas
 - Soluciones - Conclusiones



Dificultades del MS

- La problemática del mantenimiento se resume en realizar el mantenimiento del software de forma tan rigurosa que **la calidad no se deteriore** como resultado de este proceso.
- La pregunta a formular es la siguiente:
¿cómo debe mantenerse el software para preservar su fiabilidad?
- Las principales dificultades que hacen que la respuesta a esta pregunta no sea fácil y esté muy condicionada son:
 - La existencia de los llamados "*legacy code*" (código heredado).
 - Problemas inherentes al mantenimiento del software.
 - Efectos secundarios o laterales no previstos ni deseados.



Código Heredado (i)

- Con el paso de los años se ha ido produciendo un volumen muy grande de software. En la actualidad, la mayor parte de éste software está formado por **código antiguo "heredado"** (del inglés *legacy code*); es decir, código de aplicaciones desarrolladas hace algún tiempo, con técnicas y herramientas en desuso y probablemente por personas que ya no pertenecen al colectivo responsable en este momento del mantenimiento del software concreto.
- En muchas ocasiones, la situación se complica porque el código heredado fue objeto de múltiples actividades de mantenimiento. La opción de **desechar este software y reescribirlo** para adaptarlo a las nuevas necesidades tecnológicas o a los cambios en la especificación es muchas veces inadecuada por la gran **carga financiera** que supuso el desarrollo del software original y la necesidad económica de su amortización.



Código Heredado (ii)

- Los problemas específicos del mantenimiento de código heredado han sido caracterizados en las llamadas ***Leyes del Mantenimiento del Software***, propuestas por Lehman:
 - *Continuidad del Cambio.*
 - *Incremento de la Complejidad.*
 - *Evolución del Programa.*
 - *Conservación de la Estabilidad Organizacional.*
 - *Conservación de la Familiaridad.*



Leyes del Mantenimiento del Software (i)

- **Continuidad del Cambio:** Un programa utilizado en un entorno del mundo real esta **destinado a cambiar**, ya que, en caso contrario, será utilizado cada vez menos en dicho entorno (tan pronto como un programa ha sido escrito, está ya desfasado).
 - Las razones que conducen a esta afirmación son varias:
 - a los usuarios se les ocurren nuevas funcionalidades cuando comienzan a utilizar el software;
 - nuevas características en el hardware pueden permitir mejoras en el software;
 - se encuentran defectos en el software que deben ser corregidos;
 - el software debe instalarse en otro sistema operativo o máquina;
 - o el software necesita ser más eficiente.



Leyes del Mantenimiento del Software (ii)

- **Incremento de la Complejidad:** A la par que los cambios transforman los programas, su **estructura** se hará progresivamente **más compleja** salvo que se haga un esfuerzo activo para evitar este fenómeno. Esto significa que al realizar cambios en un programa (excluyendo el mantenimiento preventivo), la estructura de dicho programa se hace más compleja cuando los programadores no pueden o no quieren usar técnicas de ingeniería del software.
- **Evolución del Programa:** La evolución de un programa es un **proceso autorregulado**. Las medidas de determinadas propiedades (tamaño, tiempo entre versiones y número de errores) revelan estadísticamente determinadas tendencias e invariantes.



Leyes del Mantenimiento del Software (iii)

- **Conservación de la Estabilidad Organizacional:** A lo largo del tiempo de vida de un programa, la carga que supone el desarrollo de dicho programa es aproximadamente constante e independiente de los recursos dedicados.
- **Conservación de la Familiaridad:** Durante todo el tiempo de vida de un producto software, el incremento en el número de cambios incluidos con cada versión (release) es aproximadamente constante.



Problemas inherentes al MS (i)

- Además de las dificultades de mantenimiento que señalan las leyes anteriores, existen otros problemas clásicos que complican el mantenimiento y que son debidos a la propia naturaleza del proceso:
 - De carácter técnico:
 - Ausencia metodológica.
 - Tendencia a la desestructuración.
 - Disminución de la comprensibilidad.
 - Poca participación de los usuarios.
 - De gestión.
- Todos estos problemas se pueden atribuir -parcialmente- al gran número de programas existentes que han sido desarrollados sin utilizar la ingeniería del software (no es una panacea, pero aporta soluciones parciales a los diversos problemas planteados con el MS).



Problemas inherentes al MS (ii)

- **Ausencia metodológica:** A menudo, el mantenimiento es realizado de una manera *ad hoc* en un estilo libre establecido por el propio programador. No en todas las ocasiones esta situación es debida a la falta de tiempo para producir una modificación diseñada cuidadosamente. Prácticamente todas las metodologías se han centrado en el desarrollo de nuevos sistemas y no han tenido en cuenta la importancia del mantenimiento. Por esta razón, no existen o son poco conocidos los métodos, técnicas y herramientas que proporcionan una solución global al problema del mantenimiento.
- **Tendencia a la desestructuración:** Cambio tras cambio, los programas tienden a ser menos estructurados. Esto se manifiesta en una documentación desfasada, código que no cumple los estándares, incremento en el tiempo que los programadores necesitan para entender y comprender los programas o en el incremento en los efectos secundarios producidos por los cambios. Todas estas situaciones implican casi siempre unos costes de MS muy altos.



Problemas inherentes al MS (iii)

- **Disminución de la comprensibilidad:** Es muy habitual que los sistemas que están siendo sometidos a mantenimiento sean cada vez más difíciles de cambiar. Esto se debe al hecho de que los cambios en un programa por actividades de mantenimiento dificultan la posterior comprensión de la funcionalidad del programa (por ejemplo, el programa original puede basarse en decisiones de programación no documentadas a las que no puede acceder el personal de mantenimiento). En estas situaciones, es normal que el software no pueda ser cambiado sin correr el riesgo de introducir efectos laterales no deseados debidos a interdependencias entre variables y procedimientos que el mantenedor no ha detectado.
- **Poca participación de los usuarios:** La falta de una metodología adecuada suele conducir a que los usuarios participen poco durante el desarrollo del sistema software. Esto tiene como consecuencia que, cuando el producto se entrega a los usuarios, no satisface sus necesidades y se tienen que producir esfuerzos de mantenimiento mayores en el futuro.



Problemas inherentes al MS (iv)

- **Problemas de gestión:** Además de los problemas de carácter técnico anteriores, también pueden existir problemas de gestión.
 - Muchos programadores consideran el trabajo de mantenimiento como una **actividad inferior -menos creativa-** que les distrae del trabajo - mucho más interesante- del desarrollo de software.
 - Esta visión puede verse reforzada por las condiciones laborales y salariales y crea una **baja moral** entre las personas dedicadas al mantenimiento.
 - Como resultado de lo anterior, cuando se hace necesario realizar mantenimiento, en vez de emplear una estrategia sistemática, las correcciones tienden a ser realizadas con **precipitación**, sin pensarse de forma suficiente, no documentadas adecuadamente y pobremente integradas con el código existente.
 - No es extraño, pues, que el propio mantenimiento conduzca a la introducción de **nuevos errores e ineficiencias** que conducen a nuevos esfuerzos de mantenimiento con posterioridad.



Efectos secundarios

- La **posibilidad de error** al cambiar un procedimiento lógico tan **complejo** como el que constituyen la mayor parte de los programas actuales es **muy grande**. Por esta razón, una de las principales dificultades del MS es el riesgo del llamado **efecto bola de nieve**, de manera que los cambios producidos por una petición de mantenimiento introducen efectos secundarios que implicarán nuevas peticiones de mantenimiento en el futuro. Estos efectos secundarios suponen nuevos defectos que aparecen como consecuencia de las modificaciones realizadas.
- Según las consecuencias que se derivan, los efectos secundarios del MS son de tres clases:
 - efectos sobre el **código**,
 - efectos sobre los **datos**, y
 - efectos sobre la **documentación**.



Efectos secundarios sobre el código

- Todos los desarrolladores de software han "*sufrido*" en algún momento los problemas originados por olvidar añadir un ";" o por confundir un signo de puntuación con otro. Las consecuencias de estos "*despistes*" pueden ser muy importantes y sirven para corroborar que los efectos secundarios por cambios en el código son **difíciles de prever**.
- Las modificaciones en el **código fuente** que tienen una **mayor probabilidad de inducir** a nuevos errores son:
 - Cambios en el diseño que suponen muchos cambios en el código.
 - Eliminación o modificación de un subprograma.
 - Eliminación o modificación de una etiqueta.
 - Eliminación o modificación de un identificador.
 - Cambios para mejorar el rendimiento.
 - Modificación de la apertura/cierre de ficheros.
 - Modificación de operaciones lógicas.



Efectos secundarios sobre los datos

- Las estructuras de datos constituyen una parte fundamental y básica en cualquier producto software, por lo que cualquier cambio que se produzca en ellas puede conducir a **fallos importantes del sistema**.
- Los efectos secundarios de este tipo pueden aparecer debido a los siguientes cambios:
 - Redefinición de constantes locales o globales.
 - Modificación de los formatos de registros o archivos.
 - Cambio en el tamaño de una matriz u otras estructuras similares.
 - Modificación de la definición de variables globales.
 - Reinicialización de indicadores de control o punteros.
 - Cambios en los argumentos de los subprogramas.
- Para reducir esta clase de efectos secundarios es importante una **correcta documentación** de todos los datos, incluyendo tablas de referencias cruzadas datos-procesos.



Efectos secundarios sobre la documentación

- Los efectos secundarios de esta clase se producen cuando los **cambios sobre el código** de una aplicación **no se reflejan** en la documentación de diseño y/o en la documentación de usuario. Si la documentación técnica no se corresponde con el estado actual del software, se producirán efectos secundarios debidos a una incorrecta caracterización de las propiedades de dicho software. Por otro lado, la estima que los usuarios tendrán del producto software se reducirá considerablemente si comprueban que la documentación no se adapta a los ejecutables.
- Los cambios que con mayor probabilidad pueden producir efectos secundarios sobre la documentación son:
 - Modificar el formato de las entradas interactivas.
 - Nuevos mensajes de error no documentados.
 - Tablas o índices no actualizados.
 - Texto no actualizado correctamente.



Soluciones al Problema del MS

- Las diversas propuestas para resolver este problema pueden dividirse en dos categorías:
 - **Soluciones de Gestión** (organizativas):
 - Mejora de los recursos dedicados al mantenimiento
 - Gestión de la calidad
 - Gestión estructurada del proceso (Metodologías)
 - Organización del equipo humano
 - Documentación de los cambios
 - **Soluciones Técnicas** (herramientas y Metodologías).



Soluciones de Gestión

- En términos financieros, el MS puede ser visto como un **continuo consumidor de recursos**, mientras que los beneficios no están claros ni cuantificados. Para ayudar a evitar esta situación se necesita un **mayor apoyo por parte de la dirección** de las organizaciones para las actividades de mantenimiento. Para ello es necesario que los **gestores veteranos** (seniors) de las organizaciones sean **conscientes** de:
 - La importancia de las tecnologías de la información para la organización; y
 - Que el software es un **activo corporativo** que puede suponer una ventaja competitiva.
- Los gestores que estén descontentos con la situación y que quieran cambiarla, tendrán que adquirir un compromiso personal y visible con las soluciones organizativas propuestas. Tales soluciones se concretan fundamentalmente en dos aspectos: **los recursos y la calidad**.



Mejora de los recursos dedicados al MS

*El recurso fundamental y clave para el MS es el **humano**.*

- Por tanto, una manera de mejorar el mantenimiento podría ser **constituir un grupo separado** de programadores dedicados a mantener código antiguo.
- Sin embargo, debido al carácter poco atractivo de este trabajo, es habitual que el **personal nuevo** recién incorporado sea asignado a esta actividad.
- Estos programadores **inexpertos** deben intentar comprender la lógica de diseño del sistema, a pesar de que no pueden comprender el modelo conceptual del software debido a que carecen de experiencia de uso de las técnicas de ingeniería del software y de conocimiento del dominio de lo que el programa realiza. Así, raramente saben cómo encontrar y corregir defectos o realizar modificaciones.



Gestión de la Calidad (i)

- **Gestión de la calidad:** El aumento de los recursos humanos y económicos dedicados al MS puede suponer una solución a corto plazo, pero para resolver el problema a **largo plazo** se hace necesario adoptar una aproximación que permita **mejorar la calidad del proceso** en su conjunto.
- Entre las mejores técnicas de gestión de la calidad del software se incluyen:
 - Uso de técnicas estándares para la descomposición del software en entidades funcionales;
 - Empleo estricto de estándares de documentación del software;
 - Diseño paso a paso en cada nivel de descomposición del software;
 - Uso de código estructurado; y
 - Definición de todas las interfaces y estructuras de datos importantes antes de comenzar el diseño detallado.



Gestión de la Calidad (ii)

- **¿qué es Calidad?:** es la adecuación de un producto o proceso a las especificaciones previamente establecidas.
- Los métodos para **augmentar la calidad**, tanto de un producto software como del proceso de su producción, se parecen cada vez más a los empleados en la industria en general:
 - Adecuación a **estándares preestablecidos**;
 - QSA (**Quality Software Assurance**);
 - TQ (**Total Quality**)
- Adicionalmente, pueden utilizarse **métricas** de producto (para medir los atributos del producto software) y métricas de procesos (para evaluar la calidad del proceso).
- Otra forma de poder mejorar la calidad es utilizar **mejores herramientas** de desarrollo de software (por ejemplo, un entorno único que integre editor, compilador y depurador).



Gestión Estructurada del MS (i)

- Es muy importante emplear una **gestión estructurada** del proceso de MS.
- Este Mantenimiento Estructurado aparece como resultado de la **aplicación de una metodología** de ingeniería del software.
- La existencia de técnicas y herramientas adecuadas para Gestionar la *Configuración del Software* reduce la cantidad de esfuerzo requerido en el mantenimiento y mejora la calidad general de los cambios.

Configuración del Software: Documentación e información sobre los requerimientos, especificación, diseño y pruebas en cada una de las versiones para cada uno de los elementos software.

Elemento Software: cada uno de los componentes de un producto software: código de un módulo, fichero ejecutable, fichero de ayuda en línea, especificación de requisitos, documento de análisis, esquema de la base de datos, etc.



Gestión Estructurada del MS (ii)

- Cuando el mantenimiento no es estructurado, se sufren las **consecuencias de la falta de metodología**:
 - **dolorosa evaluación** del código (muchas veces poco legible),
 - **complicada comprensión** del sistema por la pobre documentación interna (desconocimiento de la estructura del programa, las estructuras de datos globales, las interfaces y otros requisitos de diseño y/o rendimiento),
 - **dificultad para descubrir** las consecuencias de los cambios en el código, y por último,
 - **imposibilidad de realizar** pruebas de regresión (repetición de pruebas anteriores) al no existir ningún registro de pruebas.
- Antes de optar por deshacerse de un programa y reescribirlo de nuevo, es necesario hacer un estudio detallado para evaluar las ventajas e inconvenientes de una y otra opción.



Gestión Estructurada del MS (iii)

- Pueden **atenuarse las dificultades** al mantener código heredado siguiendo algunas sugerencias propuestas por Yourdon :
 - Prevenir antes que curar: obtener la mayor información posible sobre el programa antes de que surjan las emergencias de mantenimiento.
 - Conocer y entender el flujo de control general del programa. En caso de que no exista, dibujar los diagramas de estructura y de flujo de alto nivel.
 - Evaluar la documentación.
 - Añadir comentarios al código para facilitar su entendimiento posterior.
 - Utilizar las ayudas que proporcionan los compiladores: listados de referencias cruzadas, tablas de símbolos, etc.
 - Al realizar cambios, respetar el estilo y formato previos.
 - Señalar las instrucciones cambiadas en el código.
 - Asegurarse antes de eliminar código (guardando una copia por si acaso).
 - Utilizar variables propias para evitar los posibles efectos secundarios que pueden surgir al utilizar las variables existentes previamente.
 - Llevar un registro completo de todas las actividades de mantenimiento.
 - Añadir comprobación de errores.

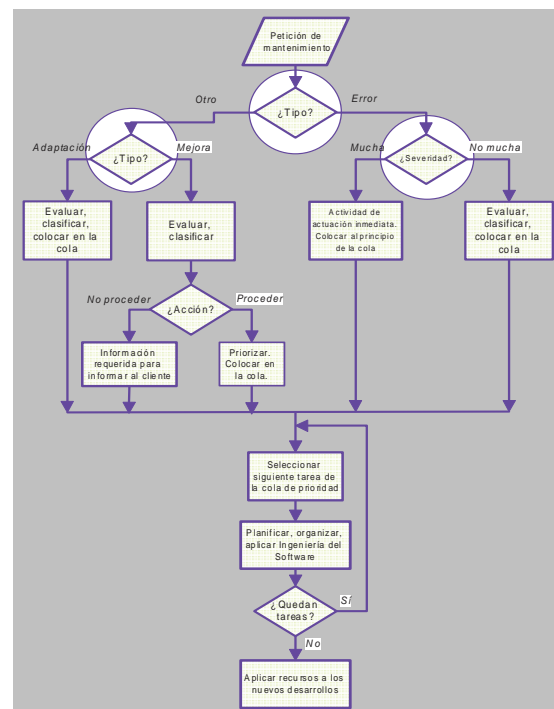


Gestión Estructurada del MS (iv)

- Debe emplearse una metodología para gestionar y realizar el MS.
- Las características y necesidades del mantenimiento de software son diferentes que las del desarrollo. Por tanto es necesario disponer de metodologías específicas:

MANTEMA

- Es una necesidad manifestada por las empresas de servicios informáticos relacionados con el mantenimiento (outsourcing, externalización, etc.).





Organización del equipo humano

- Puesto que las tareas relacionadas con el mantenimiento comienzan mucho antes de que se realice la primera petición de mantenimiento, es muy aconsejable *establecer una organización del equipo de mantenimiento desde las fases iniciales del desarrollo*, definiendo claramente las personas que participarán en cada actividad para tratar de evitar que el mantenimiento se realice "como se pueda".
- Esta organización puede ser creada formalmente o simplemente constituirse de hecho, pero, en cualquier caso, se deberán *establecer claramente los procedimientos de evaluación, control, supervisión e información de cada petición de mantenimiento*.
- **Asignar responsabilidades antes de comenzar las actividades de mantenimiento** reduce considerablemente la confusión y ayuda a evitar en gran parte las incomodidades de la persona que se siente "mareada" al cambiarla apresuradamente de tarea: de desarrollo a mantenimiento y viceversa.



Documentación de los cambios

- *Es muy importante realizar una correcta documentación de los cambios*: Por esta razón, es conveniente que las **peticiones de mantenimiento** se realicen utilizando un **formulario estandarizado**. Así mismo, el equipo de mantenimiento deberá elaborar un informe de cambios para cada petición de mantenimiento que deberá incluir un estudio del esfuerzo requerido para satisfacer la petición, la naturaleza de las modificaciones necesarias, y la prioridad (urgencia) del cambio.
- Las principales informaciones que se pueden registrar para cada cambio, son:
 - Información del programa.
 - Tamaño (LDC) del programa fuente.
 - Tamaño del ejecutable.
 - Lenguaje de programación utilizado.
 - Fecha de instalación del programa.
 - Número de fallos.
 - Número de sentencias añadidas, eliminadas y modificadas en el cambio.
 - Número de personas-hora.
 - Identificación del responsable del cambio (ingeniero de software).
 - Identificación de la petición de mantenimiento.
 - Tipo de mantenimiento.
 - Fechas de comienzo y final del mantenimiento.
 - Beneficios netos que supone el cambio.



Soluciones Técnicas (i)

- Las soluciones técnicas al problema del MS son de dos clases:

herramientas y métodos

- Las primeras sirven para soportar de forma más efectiva y cómoda los segundos.
- Estas herramientas han sido diseñadas para ayudar al personal de mantenimiento a comprender el programa y a probar sus modificaciones para asegurar que no han sido introducidos errores.
- Muchas de estas herramientas son iguales o similares a las utilizadas para la prueba (test) del software: formateador, analizador estático, estructurador, documentador, depurador interactivo, generador de datos de prueba y comparador.



Soluciones Técnicas (ii)

- Los principales **métodos** empleados en el MS son:
 - **Reingeniería**: consiste en el examen y modificación de un sistema para reconstruirlo en una nueva forma.
 - **Ingeniería Inversa**: es el proceso de analizar un sistema para identificar sus componentes y las interrelaciones que existen entre ellos, así como para crear representaciones del sistema en otra forma o en un nivel de abstracción más elevado.
 - **Reestructuración** del software: consiste en la modificación del software para hacerlo más fácil de entender y cambiar o menos susceptible de incluir errores en cambios posteriores. Se diferencia de la ingeniería inversa en que el software reestructurado tiene el mismo nivel de abstracción que el original.
- Se estudian en otra parte.



Soluciones - Conclusiones

- Necesidad de emplear **Metodologías**, si puede ser, específicas para el proceso de MS.
- Necesidad de **Herramientas**:
 - Muchas metodologías han fracasado por no disponer de herramientas que permitan su automatización.
 - Además, hacen falta **medidas** para poder tomar decisiones.
 - Para ello debemos disponer de **métricas** que nos permitan medir los aspectos de interés en MS.
- Utilizar técnicas de Control y Gestión de la **Calidad**:
 - Gestión de **Riesgos**,
 - Gestión de **Configuraciones**,
 - **Auditoria**.



Índice – Parte 5

- Soluciones técnicas.
 - Reingeniería.
 - Ingeniería inversa.
 - Reestructuración.
- Ingeniería inversa de bases de datos
- Detección de clones



Ingeniería inversa

Análisis de un sistema, de manera que se produce una representación a alto nivel del propio sistema.



Reingeniería

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 evolución.

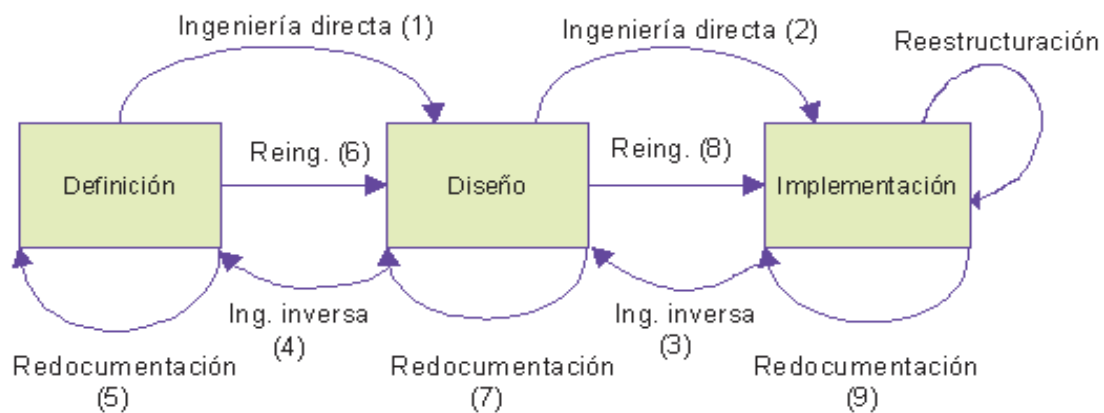


Reestructuración

Cambio de representación de un producto software, pero dentro del mismo nivel de abstracción.



Gráficamente...





Soluciones técnicas: motivaciones

- Obtención de documentación
- Transformación de aplicaciones *procedimentales* al paradigma orientado a objeto
- Transformación de sistemas transaccionales a cliente/servidor
- Paso de un lenguaje a otro, dentro del mismo paradigma
- ...



Ingeniería inversa (I)

¿De qué?

• De código

```

#include <stdio.h>
#include <string.h>
#include <vector>
#include "maquina.h"

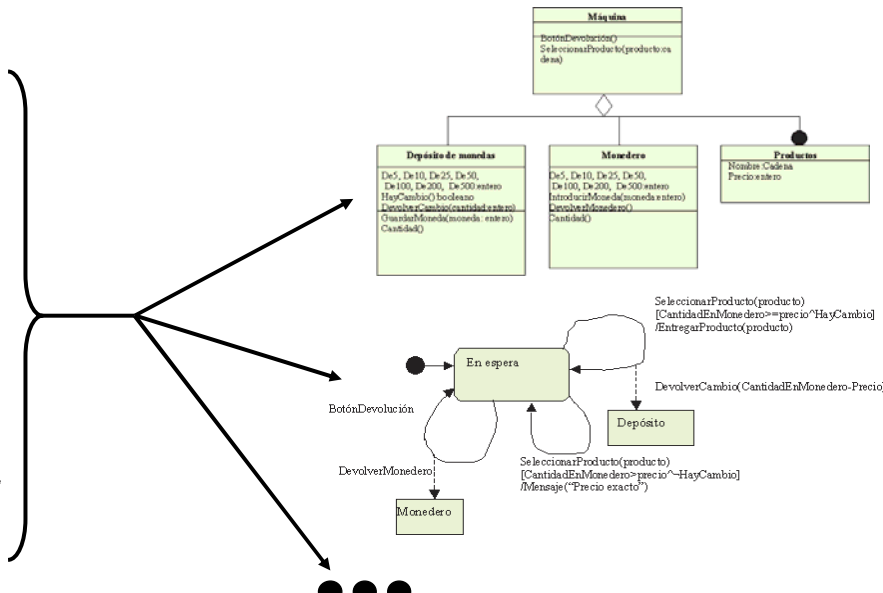
Maquina::Maquina() {
    m_deposito=0;
    m_monederos=vector<Monedero>();
    m_producto="";
}

void Maquina::Mostrar() {
    for (int i=0; i<m_monederos.size(); i++)
        printf("Monedero %d: %s, a %d ptas", i,
            m_monederos[i].nombre(), m_monederos[i].precio());
    printf("\n");
}

void Maquina::GuardarMonederoEnDeposito() {
    int i;
    for (i=0; i<m_monederos.size(); i++)
        m_deposito+=m_monederos[i].precio();
    for (i=0; i<m_monederos.size(); i++)
        m_monederos[i].precio();
    for (i=0; i<m_monederos.size(); i++)
        m_monederos[i].precio();
    for (i=0; i<m_monederos.size(); i++)
        m_monederos[i].precio();
    for (i=0; i<m_monederos.size(); i++)
        m_monederos[i].precio();
    for (i=0; i<m_monederos.size(); i++)
        m_monederos[i].precio();
}

void Maquina::SeleccionarProducto(int producto) {
    int i;
    for (i=0; i<m_producto.size(); i++)
        if (m_producto[i].precio()==m_producto[producto].precio())
            m_producto[i].precio();
}

```





¿Y de qué más?

•De bases de datos

```

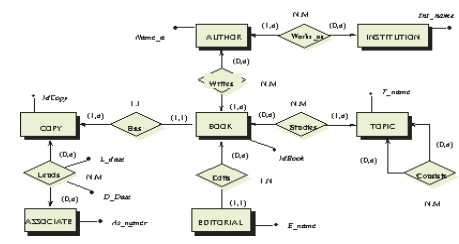
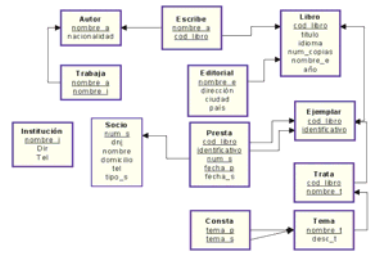
***** TABLAS *****
CREATE TABLE autor
(nombre_a nombres,
nacionalidad nacionalidades,
PRIMARY KEY (nombre_a))

CREATE TABLE trabaja
(nombre_a nombres,
nombre_i instituciones,
PRIMARY KEY (nombre_a, nombre_i),
FOREIGN KEY (nombre_a) REFERENCES autor
ON UPDATE CASCADE)

CREATE TABLE institucion
(nombre_i instituciones,
Dir lugares,
Tel telefonos,
PRIMARY KEY (nombre_i))

CREATE TABLE libro
(cod_libro códigos,
titulo título NOT NULL,
idioma idiomas NOT NULL,
num_copias número copias,
nombre_e nombres NOT NULL,
año año,
PRIMARY KEY (cod_libro),
FOREIGN KEY (nombre_e) REFERENCES editorial
ON UPDATE CASCADE)

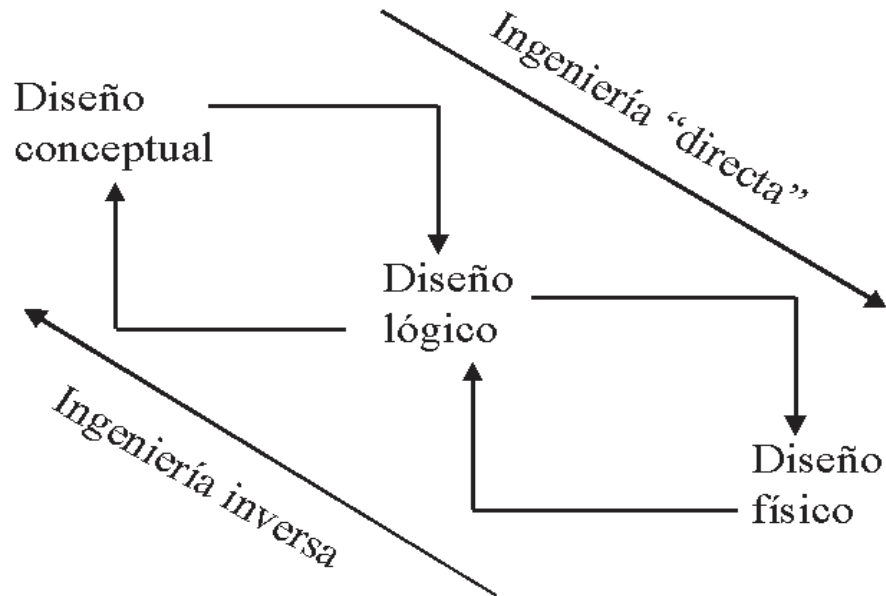
```



Ingeniería inversa de bases de datos relacionales



Ingeniería inversa de bases de datos relacionales



Ingeniería inversa de bases de datos relacionales

Comparación de métodos (Pedro de Jesús y Sousa, 1999)

- Conocimiento semántico de:
 - Atributos
 - Consistencia de nombres
- Datos:
 - Utilización de los datos
 - Existencia de errores en los datos
- Código
 - Utilización del código
 - Existencia de errores en el código
- Claves candidatas
- Claves ajenas (externas)
- Dependencias funcionales de atributos no claves o 3FN
- Dependencias de inclusión no basadas en claves
- Casos en los que se requiere entradas proporcionadas por personas



Resumen de métodos.

Método	Entradas	Salidas	Precondiciones
Chiang et al.	Datos Relaciones Claves primarias	EER	3FN Consistencia de nombres Ausencia de errores en PK
Johannesson	Relaciones Dependencias funcionales Dependencias de inclusión	Par (L, IC)	3FN
Markowitz y Makowsk	Relaciones Dependencias clave Restricciones de integridad referencial	EER	Relaciones en FN de Boyce-Codd
Navathe y Awong	Relaciones	EER	Relaciones en 3FN o FN de Boyce-Codd Consistencia en los nombres de los atributos Ausencia de ambigüedades u homónimos en FK Especificación de todas las claves candidatas
Petit et al	Relaciones con restric. de unicidad no nulas Datos Código	EER	Ninguna
Premerlani y Blaha	Relaciones Datos	Modelo de clases OMT	Ninguna
Signore et al	Relaciones Código	ER	Ninguna



Método de Hainaut et al. (I)

- Fase 1) Extracción de estructuras
 - Considerar cada fichero una posible tabla
 - Considerar campo del fichero como un posible campo de la tabla
 - Identificar un conjunto de campos susceptibles de formar la clave primaria
 - Determinar las claves externas
 - "Filtrar" las tablas (despreciar, p. ej., aquellos ficheros sin clave principal)
 - Buscar y encontrar generalizaciones
 - Encontrar asociaciones

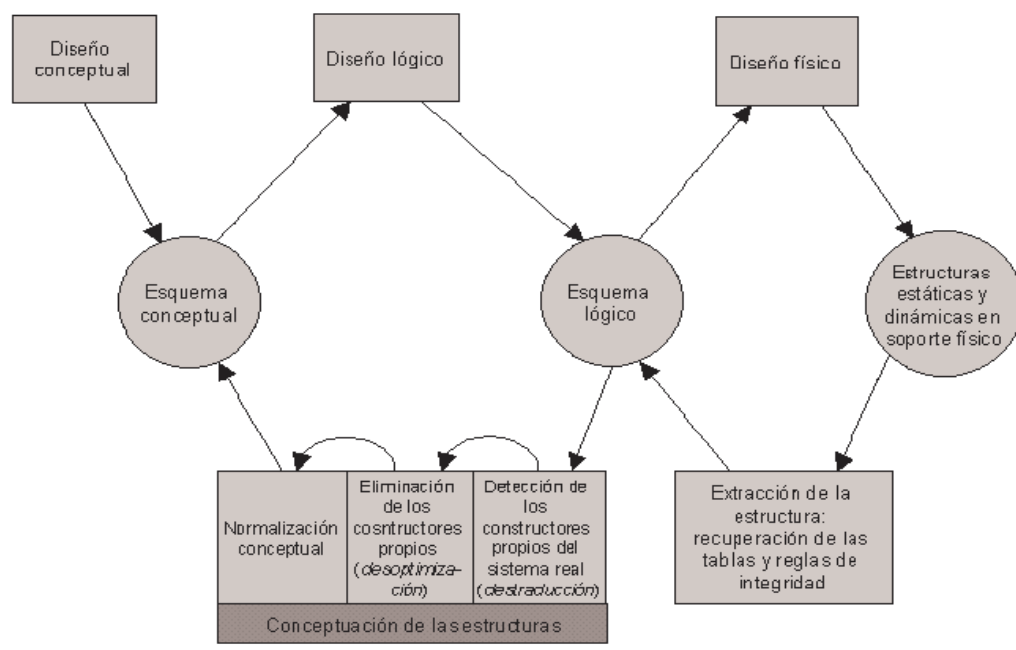


Método de Hainaut et al. (II)

- Fase 2) Conceptualización de las estructuras
 - Detectar constructores propios del sistema real
 - Reemplazarlos por constructores independientes
 - Detectar y eliminar constructores no semánticos
 - Normalización conceptual (se obtienen estructuras de alto nivel transformadas en la fase anterior)



Método de Hainaut et al. (III)





Detección de clones



Detección de Clones

- Causas para la existencia de clones:
 - Copiar y pegar por comodidad
 - Estilos de codificación
 - Operaciones sobre T.A.D.'s
 - Mejora del rendimiento
 - "Accidente"

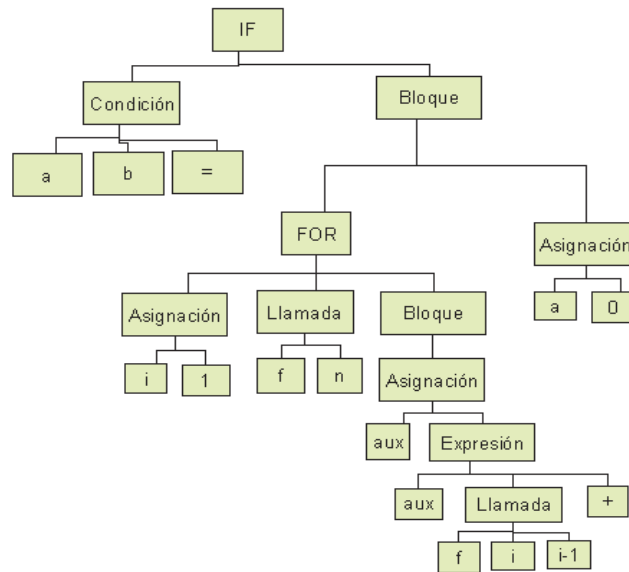


Detección de Clones

- Se pueden emplear Árboles de Sintaxis Abstracta (ASAs).

...
...
...
...
...
...
...

```
IF a=b THEN
  FOR i=1 TO f(n)
    aux= aux +f(i, i-1)
  NEXT i
  a = 0
END IF
```



Detección de Clones

- Consiste en buscar subárboles iguales o parecidos lo más grandes posible:

Clones= \emptyset

Para cada $s \in$ Subárboles

Si $\text{Hash}(s) \geq \text{PesoMínimo}$ entonces

$\text{TablaHash} = \text{TablaHash} \cup \{s\}$

Para cada $(s1, s2)$ que están en la misma entrada de TablaHash

Si $\text{Similitud}(s1, s2) \geq \text{UmbralDeSimilitud}$ entonces

Para cada $s \in$ Subárboles($s1$)

Si $s \in$ Clones entonces

$\text{Clones} = \text{Clones} - \{s\}$

Para cada $s \in$ Subárboles($s2$)

Si $s \in$ Clones entonces

$\text{Clones} = \text{Clones} - \{s\}$

$\text{Clones} = \text{Clones} \cup \{s1\} \cup \{s2\}$



Ingeniería inversa de programas

- ¿Qué solemos hacer?
 - Buscamos el programa principal
 - Despreciamos inicializaciones de variables, etc.
 - Inspeccionamos la primera rutina llamada, y la examinamos si es importante
 - Inspeccionamos las rutinas llamadas por la primera rutina del programa principal, y examinamos aquéllas que nos parecen importantes
 - ...



Ingeniería inversa de programas

- Recopilamos esas rutinas "importantes", que se llaman *componentes funcionales*
- Asignamos significado a cada componente funcional (c.f.):
 - Explicamos qué hace cada c.f. en el conjunto del sistema
 - Explicamos qué hace el sistema a partir de los diferentes cc.ff.



Ingeniería inversa de programas

- Los módulos suelen estar ocupados por c.f.
- Suele haber cc.ff. cerca de grandes zonas de comentarios (líneas de asteriscos, p. ej.)
- Los identificadores suelen ser largos y formados por palabras entendibles



Ingeniería inversa de programas

- Un componente es *funcional* cuando su ausencia...
 - ...impide seriamente el funcionamiento de la aplicación
 - ...dificulta la legibilidad del código
 - ...impide la comprensión de todo o de otro c.f.
 - ...hace caer a niveles muy bajos la calidad, fiabilidad, mantenibilidad, etc.



Encapsulación de código en funciones



Encapsulación de código en funciones

- El usuario selecciona un bloque de código y la herramienta decide si éste es sintácticamente completo.
- En caso afirmativo, coloca el código en una nueva función y reemplaza el bloque original por una llamada.
- La función creada puede pasar a ser miembro de una clase.



Encapsulación de código en funciones

- Tratamiento de las variables del bloque:
 - Genera una vble. local por cada variable del bloque que no posea información para el resto del bloque.
 - Las vbles. globales permanecen como globales.
 - Se pasan por valor las vbles. locales que se usan posteriormente sin haber cambiado su valor.
 - El resto, se pasan por referencia.



Encapsulación de código en funciones

```
void f(char c)
{
    int i, count, len;
    char str[max];

    cin >> str;
    len=strlen(str);

    count=0;
    for (i=0; i<=len; i++)
        if (str[i]==c) {
            count++;
            str[i]='\n';
        }

    cout << str << count <<
    "\n";
}
```

```
void f(char c)
{
    int i, count, len;
    char str[max];

    cin >> str;
    len=strlen(str);
    newfun(count, len, str, c);
    cout << str << count << "\n";
}

newfun(int &count, int len, char
*str, char c)
{
    int i;
    count=0;
    for (i=0; i<=len; i++)
        if (str[i]==c) {
            count++;
            str[i]='\n';
        }
}
```



Generación de clases

- Reescritura de estructuras como clases:

```
struct point
{
    int x, y;
    struct point *next;
};
```

```
class point
{
public:
    int x, y;
    struct point *next;
};
```



Ingeniería inversa de interfaces de usuario



Ingeniería inversa de interfaces de usuario

Muchos programas gozan de gran fiabilidad, capacidad de procesamiento, etc., pero sus diseñadores han olvidado la comodidad y facilidad de uso del usuario final



“Respetemos la lógica interior, pero adaptemos su aspecto exterior”.



Ingeniería inversa de interfaces de usuario

- Consejos
 - Recopilación de la documentación disponible
 - Entrevistas a los diferentes usuarios
 - Dirección
 - Diseñadores y equipo de mantenimiento
 - Usuarios finales
 - Uso del sistema por el propio equipo de mantenimiento.
 - Uso de contadores