Mantenimiento y Evolución del Software

M.I. Capel

ETS Ingenierías Informática
y Telecommunicación
Universidad de Granada
Email: manuelcapel@ugr.es

Desarrollo de Software

Índice

Índice

Índice

Índice

Mantenimiento

Definición

"Cualquier trabajo hecho para cambiar el sistema después de ponerlo en operación"

- El software no se degrada ni necesita de un mantenimiento ""físico" periódico, como otras obras de ingeniería
- Sin embargo, el software está en continua evolución y el proceso de su mantenimiento puede ser muy difícil

Mantenimiento

Definición

"Cualquier trabajo hecho para cambiar el sistema después de ponerlo en operación"

- El software no se degrada ni necesita de un mantenimiento ""físico" periódico, como otras obras de ingeniería
- Sin embargo, el software está en continua evolución y el proceso de su mantenimiento puede ser muy difícil

Evolución de un sistema software

Aspectos a tratar

- Las funciones diarias del sistema
- Las modificaciones del sistema
- Perfeccionamiento de su funcionalidad
- Conservación de los niveles de desempeño del sistema

Mantenimiento de sistemas *hard* y *soft*

Sistemas Software

Se diseñan e implementan para que se incorporen cambios a lo largo de todo su ciclo de vida, a diferencia de los sistemas hardware–exclusivos

Mutabilidad de los sistemas

- Alta si los requerimientos del sistema son muy dependientes del contexto de ejecución del sistema
- Dependencia con el tipo de sistema: S, P, o E

Diferencias entre mantenimiento y desarrollo de software

Control de los Cambios

- Los cambios afectan a etapas del ciclo anteriores y posteriores a la actual
- Modularización de componentes de código
- Trazabilidad de los requerimientos hacia/desde las pruebas
- Aplicación de los principios de la IS
- El costo del ciclo de vida de un sistema se incrementa con el paso del tiempo
- Comportamiento evolutivo de los sistemas software

Características del mantenimiento del software

Dificultades

- Un difícil mantenimiento tiene un impacto importante en los costes reales del software
- Novedad del software
- Plazo de vida del sistema
- Dependencia de entornos de ejecución cambiantes
- Calidad del diseño y del código

El esfuerzo de mantenimiento de un sistema software se puede modelar empíricamente

Características del mantenimiento del software

Dificultades

- Un difícil mantenimiento tiene un impacto importante en los costes reales del software
- Novedad del software
- Plazo de vida del sistema
- Dependencia de entornos de ejecución cambiantes
- Calidad del diseño y del código

El esfuerzo de mantenimiento de un sistema software se puede modelar empíricamente

Medida y estimación del esfuerzo de mantenimiento

Facilidad de mantenimiento del software

- Se trata de un atributo *externo* del software
- Resultado de varios factores: calidad del código, especificación, diseño, documentación y plan de pruebas

Medidas

- Predicen la probabilidad de que un sistema sea fácil de mantener
- Se deberían proporcionar al cliente junto con el software

Medida y estimación del esfuerzo de mantenimiento

Facilidad de mantenimiento del software

- Se trata de un atributo *externo* del software
- Resultado de varios factores: calidad del código, especificación, diseño, documentación y plan de pruebas

Medidas

- Predicen la probabilidad de que un sistema sea fácil de mantener
- Se deberían proporcionar al cliente junto con el software

Gestión de Configuración del Sistema

Seguimiento de los cambios

- Cuanto más complejo es un sistema, los cambios producen mayor impacto en más componentes
- La *gestión de la configuración* (GC) de un sistema se convierte en una actividad crítica del mantenimiento del software

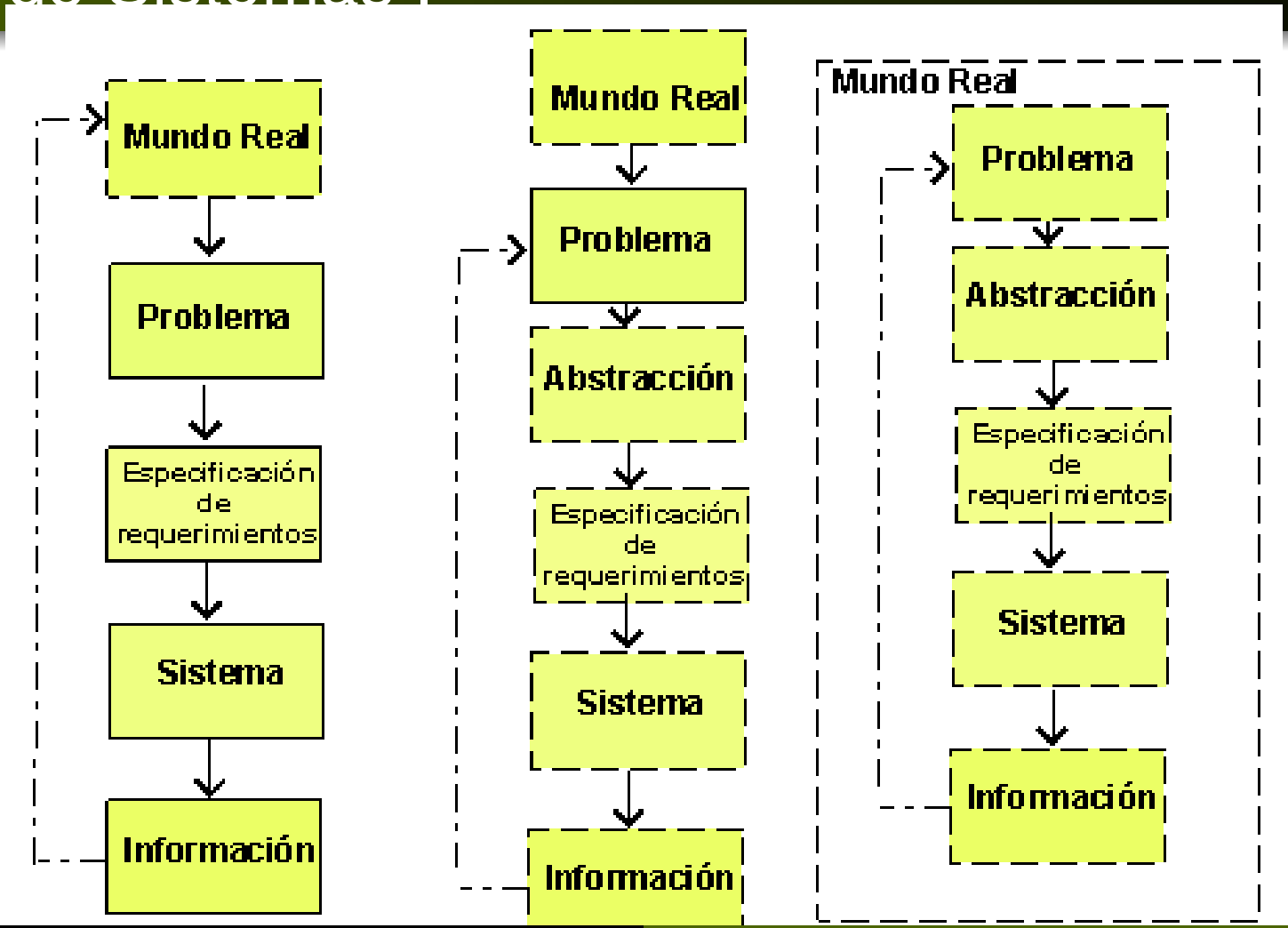
Técnicas y herramientas de mantenimiento software

- Técnicas específicas:
  - Control de versiones
  - Análisis de impacto
  - Uso de herramientas–software

Costes del Mantenimiento de Sistemas Software

Periodo	Desarrollo	Mantenimiento
años setenta	60%	40%
años ochenta	40%	60%
actualidad	20%	80%

Tipos de Sistemas I



Tipos de Sistemas II

Predisposición a los cambios de los sistemas

- *Sistemas-S*:
  - Improbabilidad de cambios en todas las fases del ciclo
    -Manipulación de matrices
- *Sistemas-P*:
  - Solución aproximada al problema que pretende resolver el sistema
    -Programa para jugar al ajedrez
- *Sistemas-E*:
  - Naturaleza altamente mutable, que cambia con rapidez
  - Se han de incluir las mutaciones en el propio sistema para acomodar los frecuentes cambios
    -Software que predice cómo funciona la Economía

Cambio en las actividades del proceso de desarrollo

Actividad que inicia cambios	**Artefactos a cambiar**
Análisis de requerimientos	Especificación de requerimientos
Diseño de sistemas	Especificación de diseño arquitectónico Especificación de diseño técnico
Diseño de programas	Especificación del diseño de programa
Implementación de programas	Código de programas Documentación de programas
Pruebas unitarias	Planificación de las pruebas Scripts con las pruebas(casos,suites,etc.)
Entrega del sistema	Documentación del usuario Documentación del operador Guía del usuario Guía del programador Clases de entrenamiento

Tabla: Actividades y artefactos que cambian

Control de costos asociados a los cambios

- Costos del *ciclo de vida*:
  - Tamaño del sistema
  - Recursos que gestiona el sistema
  - Complejidad del sistema
  - Comprensión del código
- Fácil instrumentación de los cambios si se siguen los principios de la IS

Leyes de comportamiento evolutivo

Observación

Los sistemas software no evolucionan caóticamente, sino que siguen un comportamiento *previsible*

Leyes de Lehman de evolución del software

- Comportamiento *evolutivo* de los sistemas software
- Ley de Continuidad del Cambio
- Ley de Complejidad Creciente
- Ley Fundamental de la Evolución de un Programa
- Ley de Conservación de la Estabilidad Orgánica
- Ley de Conservación de la Familiaridad

Procedimiento para decidir los cambios

Para decidir llevar a cambio un cambio en el sistema se ha de seguir un procedimiento sistemático de toma de decisiones

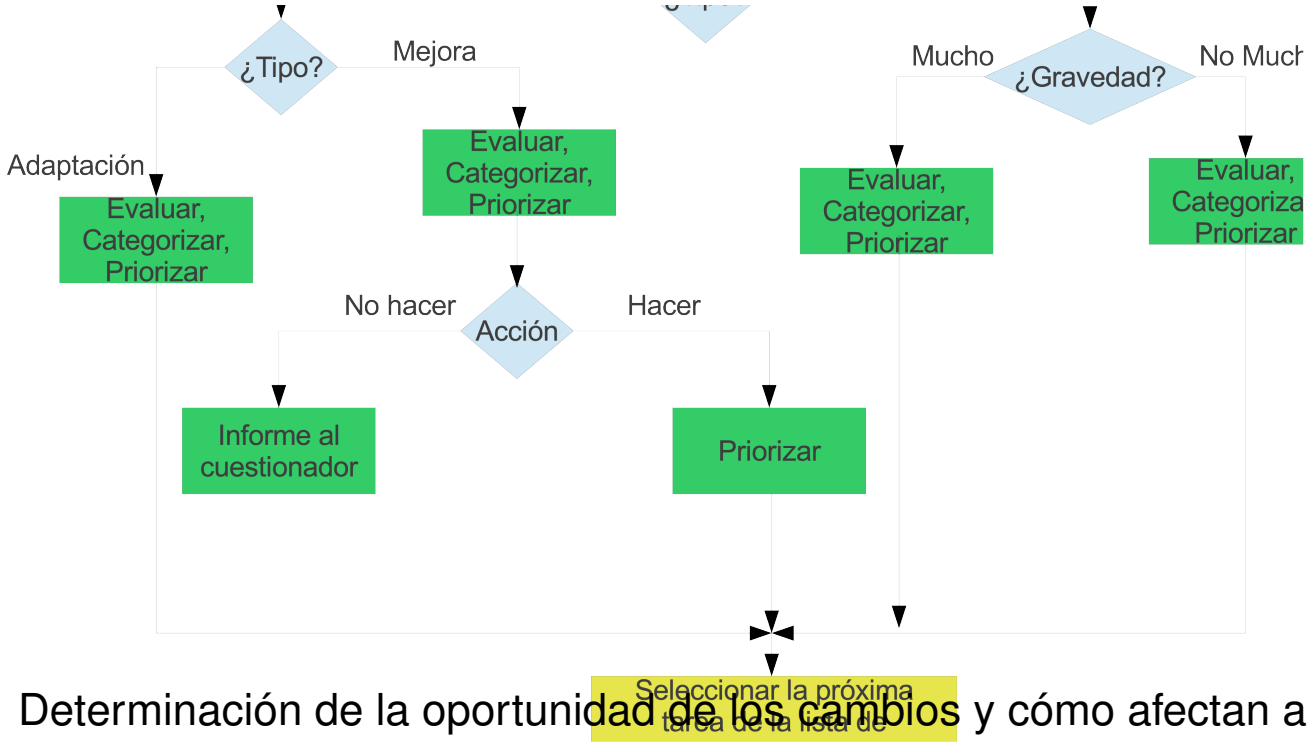


Figura: Determinación de la oportunidad de los cambios y cómo afectan a los requerimientos

El Plan de Control de Cambios

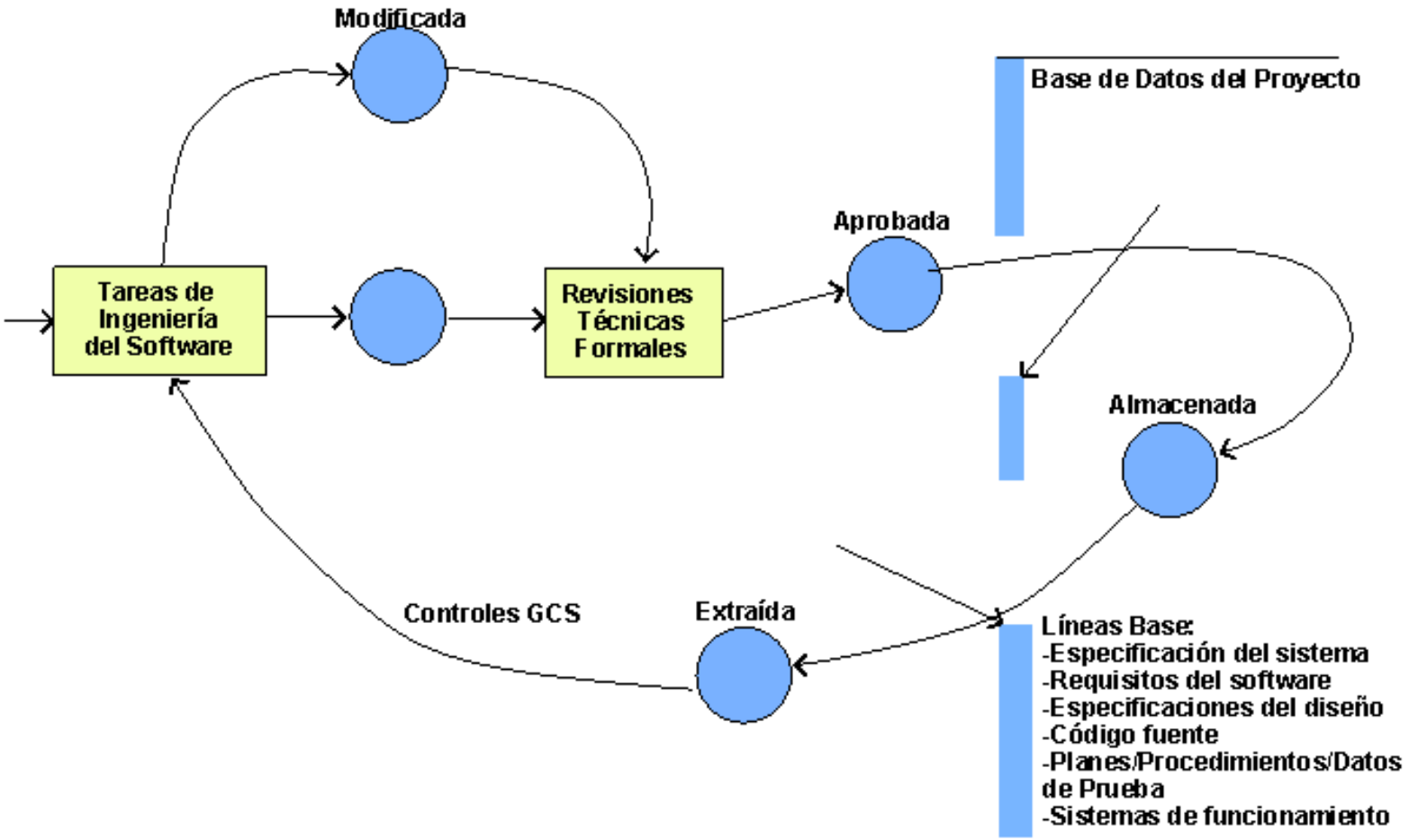
- Clase de elementos que componen el plan de control
- Quién toma la responsabilidad de los procedimientos y la creación de la *Línea Base*
- Políticas de control de cambios y versiones
- Almacenamiento de la información relevante
- Las herramientas que deberían ser usadas
- Proceso de uso de la herramienta.
- Base de datos de gestión de los cambios
- Información adicional

Línea Base I

Antecedentes

- Es un concepto que nos ayuda a controlar modificaciones en el sistema sin impedir que se lleven a cabo ningún cambio justificado
- El estándar IEEE 610.12-1990 define una línea base como:
  - Especificación o producto que se ha revisado formalmente
  - Existe acuerdo sobre el producto
  - Soporte para un desarrollo posterior de otro sistema
  - Puede cambiarse solamente a través de procedimientos establecidos

Línea Base II



La Base de Datos de Configuración

- Mantiene toda la información relevante para gestionar la *configuración* de un sistema software
- Debería permitir consultas sobre las configuraciones:
  - ¿Quién tiene una versión particular del sistema?
  - ¿Qué plataforma es requerida para una versión particular?
  - ¿Qué versiones se ven afectadas por un cambio en el componente X?
  - ¿Cuántas fallas fueron informadas desde la distribución de la versión T?
- Debería estar unida indisociablemente al software que está siendo gestionado

Gerencia de los cambios en una organización

Tareas de mantenimiento

- La aceptación de nuevos requisitos para el software implica a varios niveles de la organización
- El equipo de mantenimiento finalmente decide si el cambio puede ser asumido a un coste razonable

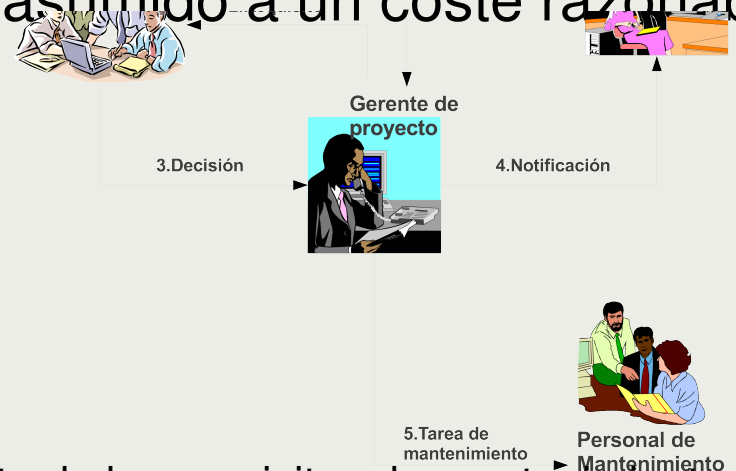


Figura: Movimiento de los requisitos de mantenimiento a través de la organización

Comité de Gestión de Cambios

1. Registra los síntomas en un formulario de solicitud de cambio.
2. El cambio propuesto se informa al comité.
3. Discutir el problema y determinar el motivo del cambio
4. Cuando se trata de un fallo del sistema, discute el origen probable del problema
5. El CCG asigna una prioridad o nivel de severidad a la solicitud
6. El analista o programador designado localiza el origen del problema o los componentes involucrados
7. El responsable de los cambios trabaja en colaboración con el *administrador del sistema*
8. El programador o analista archiva un informe de cambio

Formulario de Solicitud de Cambios

Proyecto	Número
Solicitante	Fecha
Cambio Solicitado	Urgencia, motivación
Analizador del cambio	Fecha de análisis
Componentes asociados	
Evaluación del cambio	Costos
Prioridad del cambio	
Implementación del cambio	
Esfuerzo estimado	
Fecha entrega al comité	Fecha de decisión del comité
Decisión del comité	
Implementador del cambio	Fecha del cambio
Fecha envío al QA	Decisión del QA
Fecha de envío a CM	
Comentarios	

Tabla: Formato de Solicitud de Cambio

Evolución del *Ciclo de Mantenimiento*

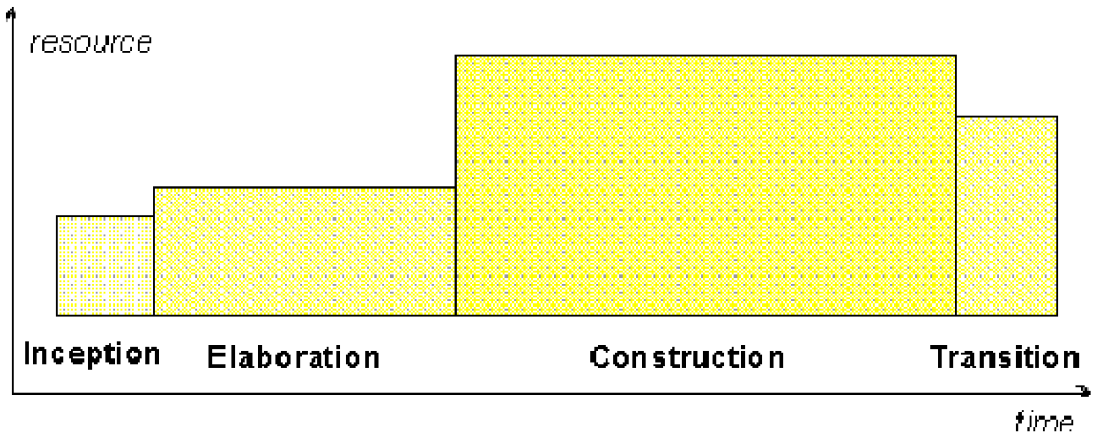


Figura: Primera versión de un sistema de software

Impacto del mantenimiento en la evolución de los sistemas software

Características mantenimiento vs. desarrollo de software

A diferencia de la actividad de *desarrollo*, en el mantenimiento de hay tener en cuenta etapas anteriores, actual y futura del ciclo del software

Actividades para controlar

- Funciones diarias
- Modificaciones
- Perfeccionar la funcionalidad
- Impedir la degradación en el desempeño

Actividades del mantenimiento

Características

- Tiene relación con todas las fases del ciclo de vida:
  - Productos anteriores de desarrollo (anteriores)
  - Relación activa con usuarios, programadores y operadores (presente)
  - Anticiparse a fallos (futuro)
- Valoración del importante papel que juegan los programadores en el mantenimiento del sistema

Evolución del *Ciclo de Mantenimiento II*

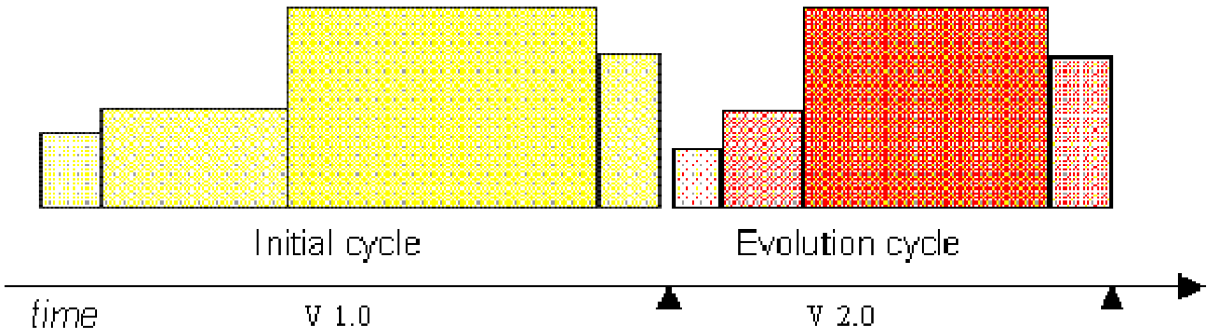


Figura: Extensión simple de un sistema de software

Condiciones

- Ninguno de los nuevos requerimientos afecta a la arquitectura software
- Fase de *Elaboración* reducida

Clasificación de los tipos de mantenimiento

Clasificación para sistemas software

- Mantenimiento correctivo
- Mantenimiento adaptativo
- Mantenimiento perfectivo
- Mantenimiento preventivo

Mantenimiento Correctivo

Características

- Objetivos
- Frecuencia
- Afectan a los requerimientos, diseño, código, pruebas y documentación
- Durabilidad de los cambios y reparaciones

Mantenimiento Adaptativo

Características

- Objetivos
- Frecuencia
- Durabilidad de los cambios y reparaciones

Mantenimiento Perfectivo

Características

- Se buscan oportunidades de mejora del sistema
- No tiene por qué estar dirigido por la detección de fallos
- Puede afectar a la documentación del sistema, a las pruebas, al diseño y a la codificación

Evolución del *Ciclo de Mantenimiento III*

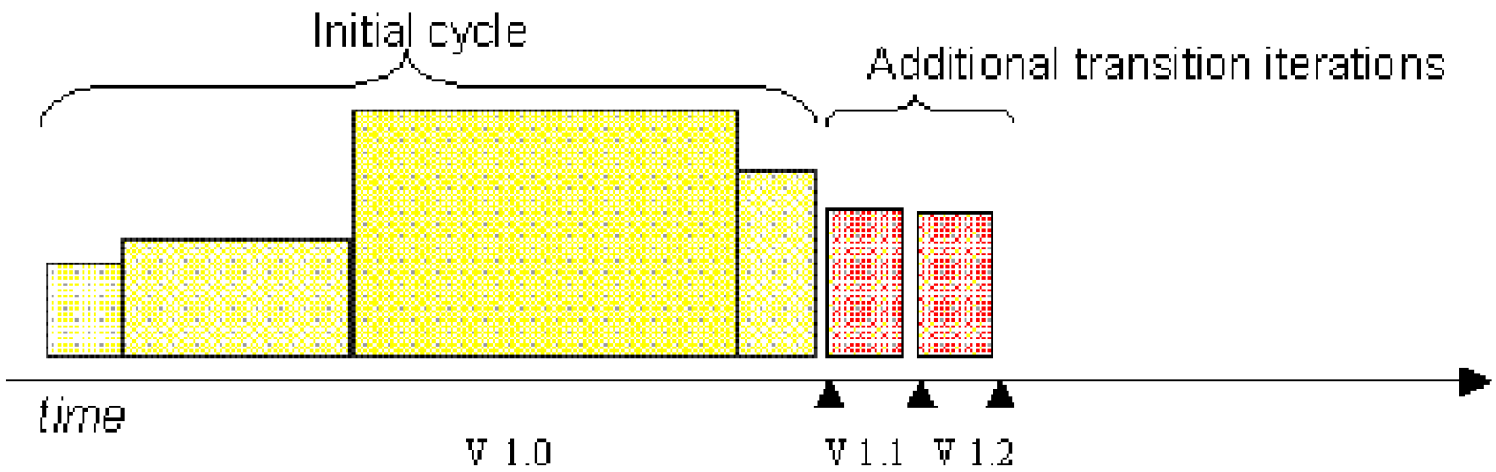


Figura: Mantenimiento perfectivo con cambios mínimos entre iteraciones

Mantenimiento Preventivo

Características

- Similar al *preventivo*, ha de involucrar también la modificación para prevenir fallos
- Frecuencia
- Anticipación

Evolución del *Ciclo de Mantenimiento IV*

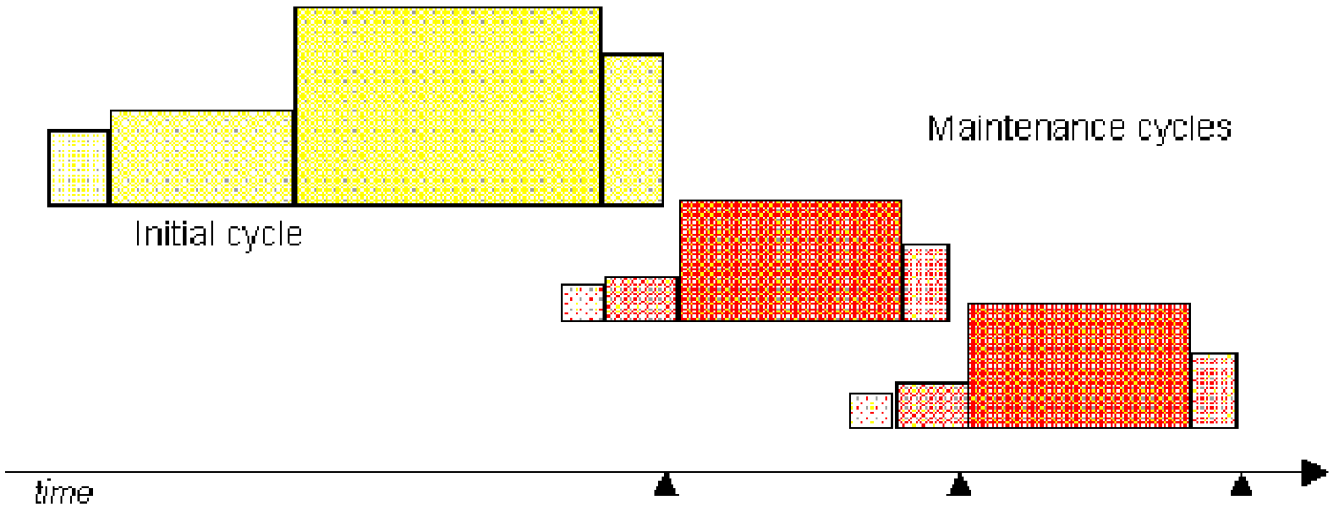


Figura: Versión del mantenimiento con ciclos concurrentes

Condiciones

- Componentes altamente cohesivos y sin acoplamiento
- Reduce los tiempos de puesta en servicio

Temporización de los tipos de mantenimiento

Antecedentes

- Varía mucho dependiendo del tipo de sistema y equipos implicados
- Lienz y Swanson (1981) realizaron un estudio de 487 organizaciones de proceso de datos

Tipo	Porcentaje
Correctivo	21 %
Adaptativo	25%
Perfectivo	50%
Preventivo	4%

Tabla: Distribución del esfuerzo de mantenimiento

Mantenimiento de Sistemas *Muy Grandes*

Estándar ISO/IEC-IEEE 12207

1. Obtención de los requerimientos de mantenimiento
2. Análisis del problema y de la modificación necesaria
3. Transformación de los requisitos detectados en cambios
4. Diseño de los cambios
5. Implementación de los cambios
6. Revisión y aceptación del mantenimiento
7. Migración

Ciclo de Mantenimiento de RUP

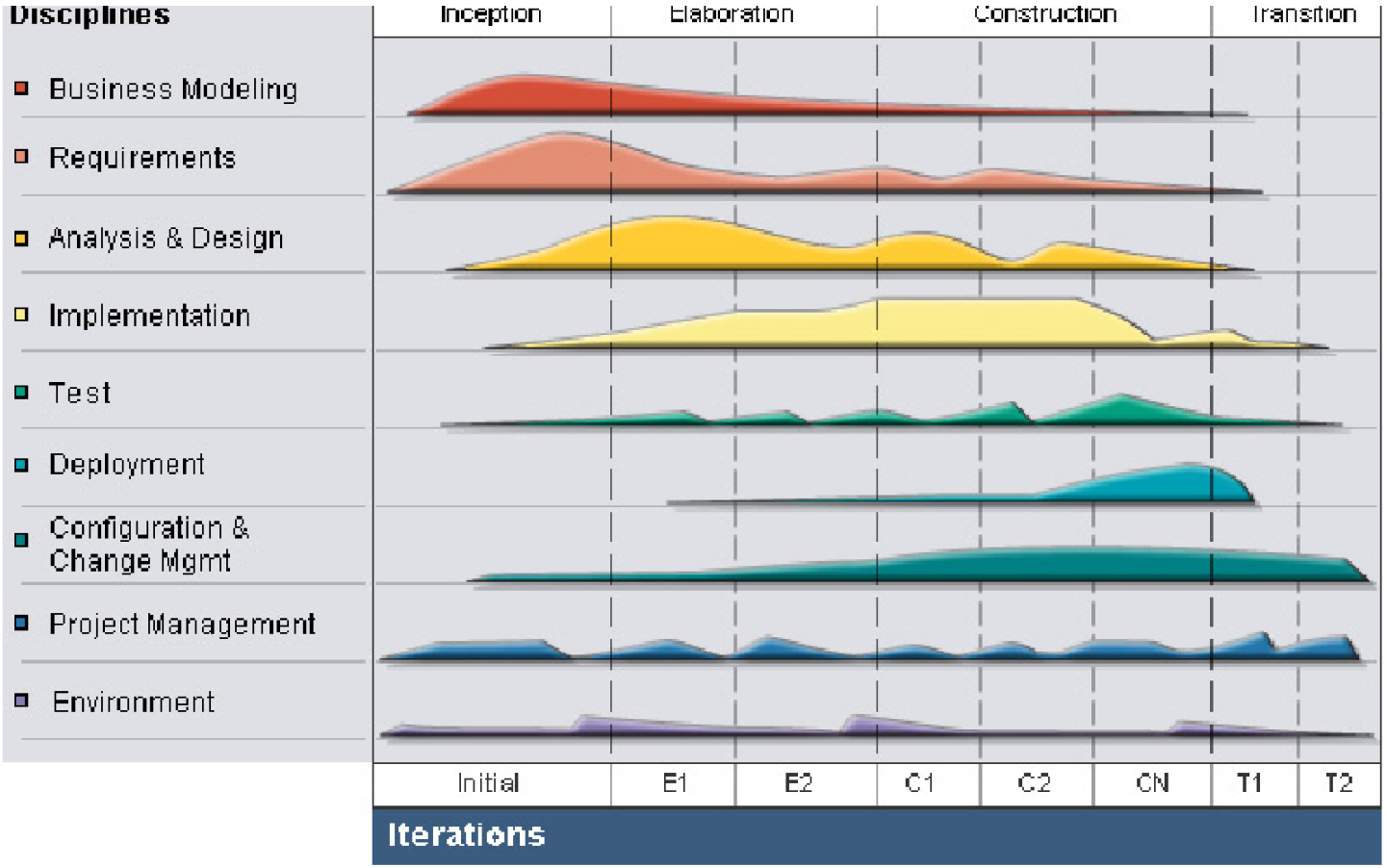
Rational Unified Process

1. Implementación
2. Pruebas
3. Despliegue del Sistema
4. Configuración y Gerencia de Cambios (CGC)
5. Gestión del Proyecto
6. Entorno de ejecución

RUP (Rational Unified Process)

- Marco de trabajo para el proceso iterativo de desarrollo de software creado por “Rational Software Corporation" (IBM, 2003)
- Se trata de un marco de trabajo adaptable, por organizaciones que se dedican al desarrollo de software, según sus necesidades
- RUP es una implementación del Proceso Unificado (1251_bestpractices_TP026B)

Ciclo de Mantenimiento de RUP II



Factores determinantes del costo del mantenimiento

- Impactan en el coste del mantenimiento:
  - *Tipo de aplicación*
  - *Novedad del sistema*
  - *Plazo de vida del Sistema*
  - *Dependencia de un ambiente cambiante y del tipo de sistema*
  - *Características del hardware*
  - *Calidad del Diseño*
  - *Calidad del Código*
  - *Calidad de la Documentación*
  - *Calidad de las Pruebas*
- Los costes de mantenimiento pueden haberse incrementado actualmente hasta el 80% del ciclo de vida

Predicción del coste de mantenimiento

El esfuerzo de mantenimiento se puede modelar para predecir su magnitud y el coste de llevarlo a cabo completamente

Modelo Predictivo de Belady-Lehman

- Deterioro de un sistema con el tiempo por las reparaciones y perfeccionamiento de su arquitectura y código
- Cuando se corrige, el sistema se vuelve más complejo:

$$M = p + K \times c - d \qquad (0)$$

- Si un sistema se desarrolla sin los principios de la IS (alta complejidad) y el código es difícil de comprender (baja "d"), los costes de mantenimiento aumentarán exponencialmente

Predicción del coste de mantenimiento II

COCOMO II

$$Tamano = ASLOC(AA + SU + 0.4DM + 0.3CM + 0.3IM)/100$$

- ASLOC: número de líneas de código fuente que han de adaptarse
- AA: esfuerzo de valoración y asimilación
- SU: cantidad de código que ha ser comprendido necesariamente
- DM: porcentaje del diseño que ha de ser modificado
- CM: porcentaje de código que ha de ser modificado
- IM: porcentaje de código externo que ha de ser integrado

Esfuerzo de comprensión del software (COCOMO)

	Muy bajo	Bajo	Normal	Alto	Muy Alto
Estructura	Cohesión muy baja Alto acoplamiento	Cohesión moderada Alto acoplamiento	Bien estructurado Con áreas débiles	Alta cohesión Bajo acoplamiento	Alta modularidad ocultación inform.
Claridad aplicación--programa	Falta correlación programa y aplicación	Alguna correlación con programa	Moderada correlación con programa	Buena correlación con programa	Coinciden aplicación -programa
Autocontenido	Código poco claro, sin documentación	Algún comentario y cabeceras	Moderada calidad documentación	Buena, útil docum. áreas débiles	Código y razón del diseño
Incremento **SU**	50	40	30	20	10

Tabla: Valoración de la comprensión del código

Esfuerzo de evaluación y asimilación (COCOMO)

Incremento de Evaluación y Asimilación (AA)	Nivel del esfuerzo de Evaluación y Asimilación (AA)
0	Ninguno
2	Búsqueda componentes y evaluación documentación
6	Bastante prueba de componentes y y evaluación de documentación
8	Prueba extendida de componentes y y evaluación de documentación

Tabla: Valoración del esfuerzo de asimilación de un software

Medida de la facilidad de mantenimiento

Facilidad de mantenimiento

- Definición: “Tiempo medio para realizar la reparación de un software"
- Factores necesarios externos a evaluar:
  - Momento en que se dá parte del problema
  - Retrasos administrativos
  - Tiempo de análisis del problema
  - Tiempo para especificar los cambios
  - Tiempo necesario para hacer los cambios
  - Tiempo para probar los cambios
  - Tiempo para documentar los cambios
- Visión *externa*, deja cosas fuera

Medida de la facilidad de mantenimiento II

Medidas aconsejadas después de los cambios

- Relación entre el tiempo total de implementación del cambio y la cantidad total de cambios realizados
- Número de problemas no resueltos
- Tiempo gastado en problemas no resueltos
- El porcentaje de cambios que introducen nuevos defectos
- El número de componentes modificados para implementar un cambio

Son dependientes del entorno

Medida del Mantenimiento III

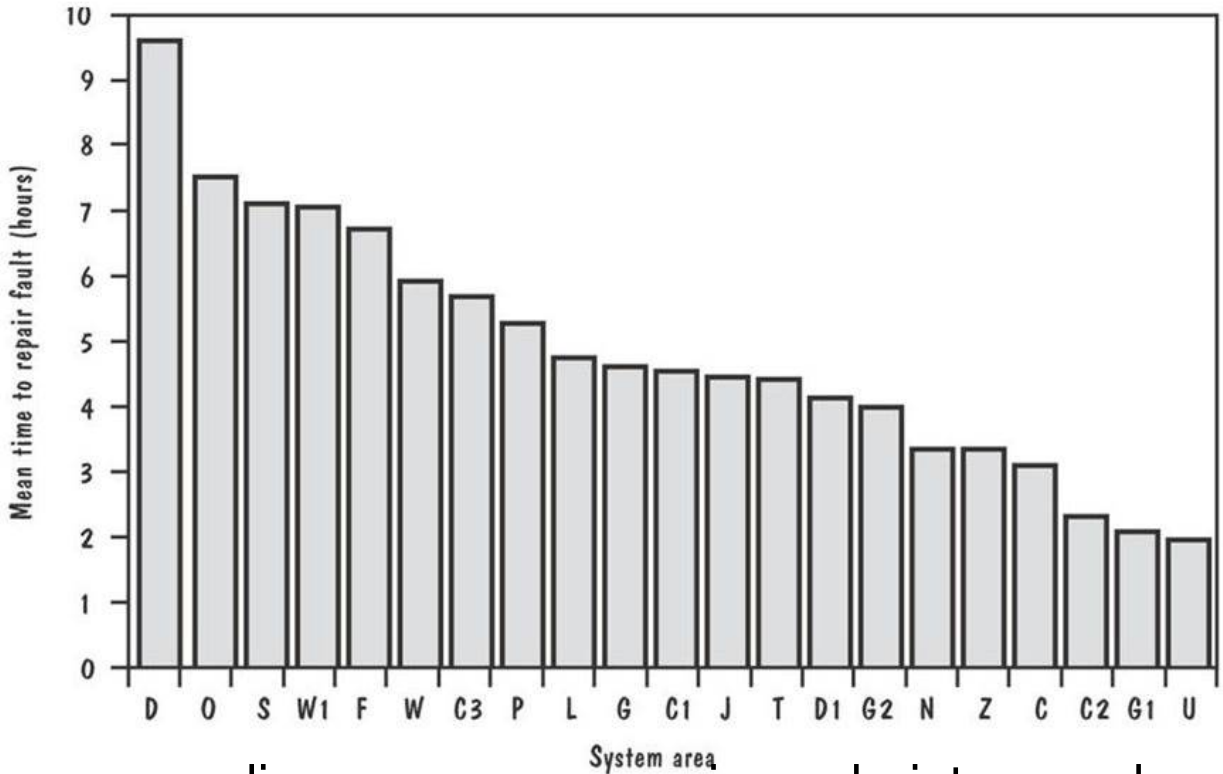


Figura: Tiempo medio para reparar varios subsistemas de un software

Medidas dependientes de la complejidad del software

Número de *McCabe* o *Número ciclomático*

- Se trata fe un atributo interno, muy importante para determinar la facilidad de mantenimiento del software:

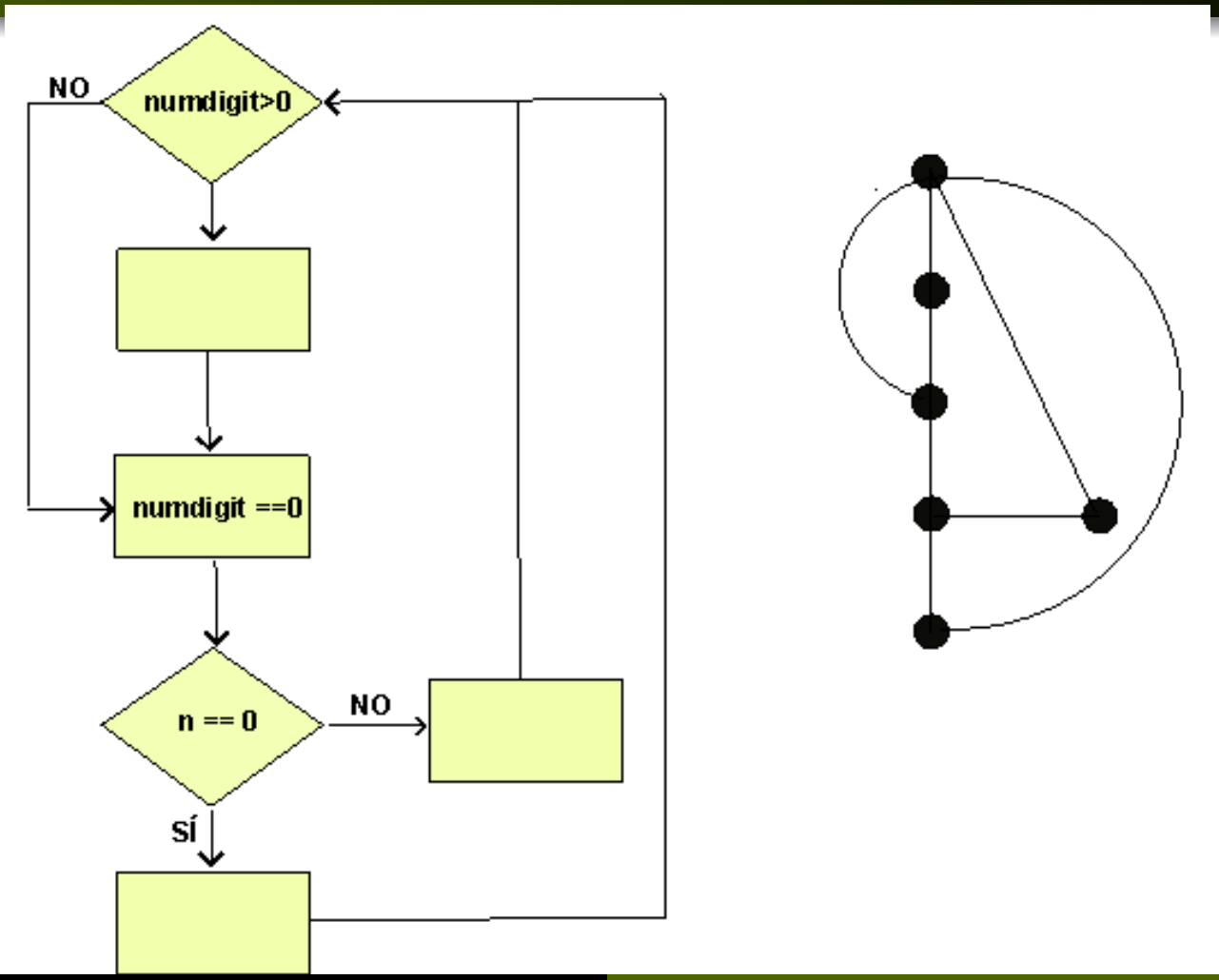
$$E - N + 2 \quad (1)$$

- Captura la complejidad estructural de un código fuente midiendo el número de caminos linealmente independientes del flujo de control
  - E: Número de arcos (*edges*) del grafo
  - N: Número de nodos del grafo

Ejemplo medida número ciclomático

```
Scoreboard::drawscore(int n)
{ while(numdigits-- > 0} {
                score[numdigits]->erase();}
        // build new score in loop, each time update
            position
        numdigits = 0;
        // if score is 0, just display ''0''
        if (n == 0) { delete score[numdigits];
                score[numdigits] = new Displayable(
                    digits[0]);
                score[numdigits]->move(Point((700-
                    numdigits*18),40));
                score[numdigits]->draw();
                numdigits++;}
while (n) { int rem = n % 10;
                delete score[numdigits];
                score[numdigits] = new Displayable(
```

Ejemplo de cálculo del número ciclomático II



Otros factores internos que dificultan el mantenimiento

- Jerarquía de herencia entre entidades sintácticas de los lenguajes de programación
- ligadura dinámica entre referencias y código actual de los métodos que se ejecutan
- relaciones de uso, delegación
- inclusión de clases y paquetes

Análisis de Impacto

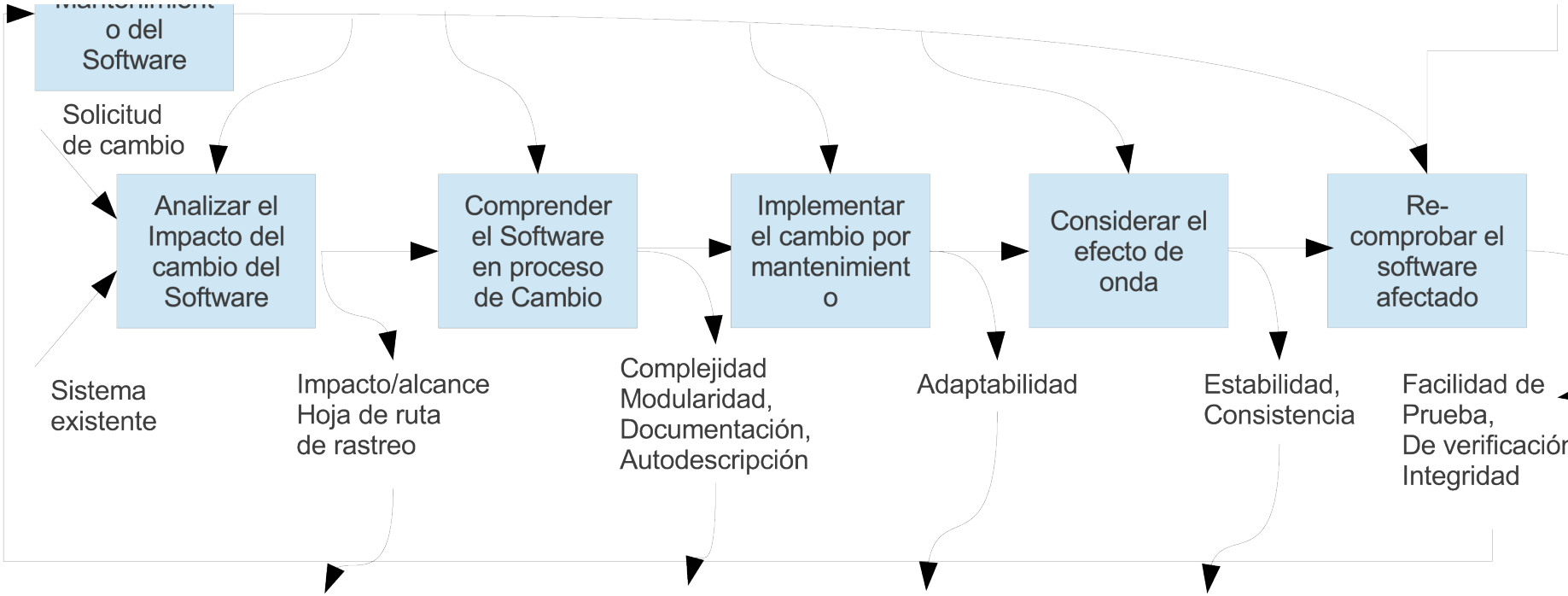
Antecedentes

- El mantenimiento de software comienza con la definición de los requerimientos del software
- Definición: “Evaluación de los riesgos asociados con el cambio de un sistema software: estimación de efectos, esfuerzo de desarrollo y cronograma"
- Ayuda a mantener bajo control el costo del mantenimiento del software
- Para el mantenimiento se han de aplicar los mismos principios de IS que para desarrollar buen software

Modelo de mantenimiento del software

- Debido a Pfleeger y Bohner, permite medir el impacto de un cambio y determinar los riesgos de llevarlo a cabo:
  - Incluye *realimentación* de subactividades
  - Medidas que proporcionan información a los gerentes
  - Determina: cuándo y cómo realizar un cambio
- Flechas etiquetadas en la parte inferior del diagrama proporcionan medidas del impacto de los cambios
- Los requerimientos, componentes de diseño, código, casos de prueba y documentación son subproductos del proceso de desarrollo del software

Modelo de Pfleeger del Mantenimiento



Análisis de Impacto II

Calidad del software

- Las calidad de los productos software (requerimientos, código, pruebas, documentación) pueda afectar a la calidad de los otros
- Se necesitan facilidades de *rastreo* (*tracing*) para comprender el juego completo de relaciones que se evalúan entre productos software durante el análisis
- Ambos tipos de rastreos son necesarios para comprender el juego completo de relaciones que se evalúan durante el análisis de impacto de los cambios

Tipos de rastreos entre subproductos

- Subproducto (workproduct): cualquier artefacto de desarrollo cuyo cambio sea significativo
- Trazabilidad horizontal: relaciones de componentes a través de colecciones de subproductos
- Trazabilidad vertical: relaciones entre partes de un subproducto

Gráfica de los rastreos entre subproductos

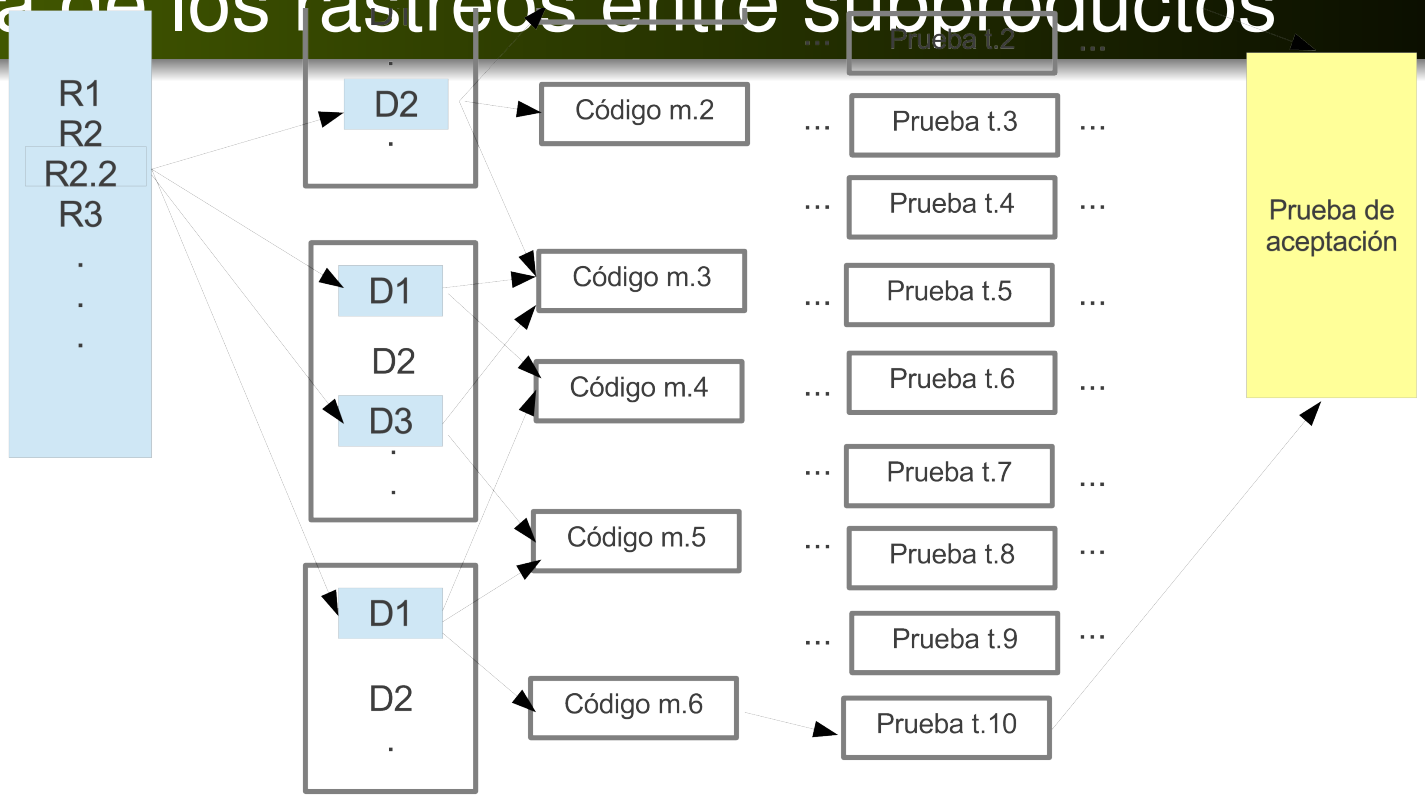


Figura: Trazabilidad Horizontal

Gráfica de los rastreos en subproductos

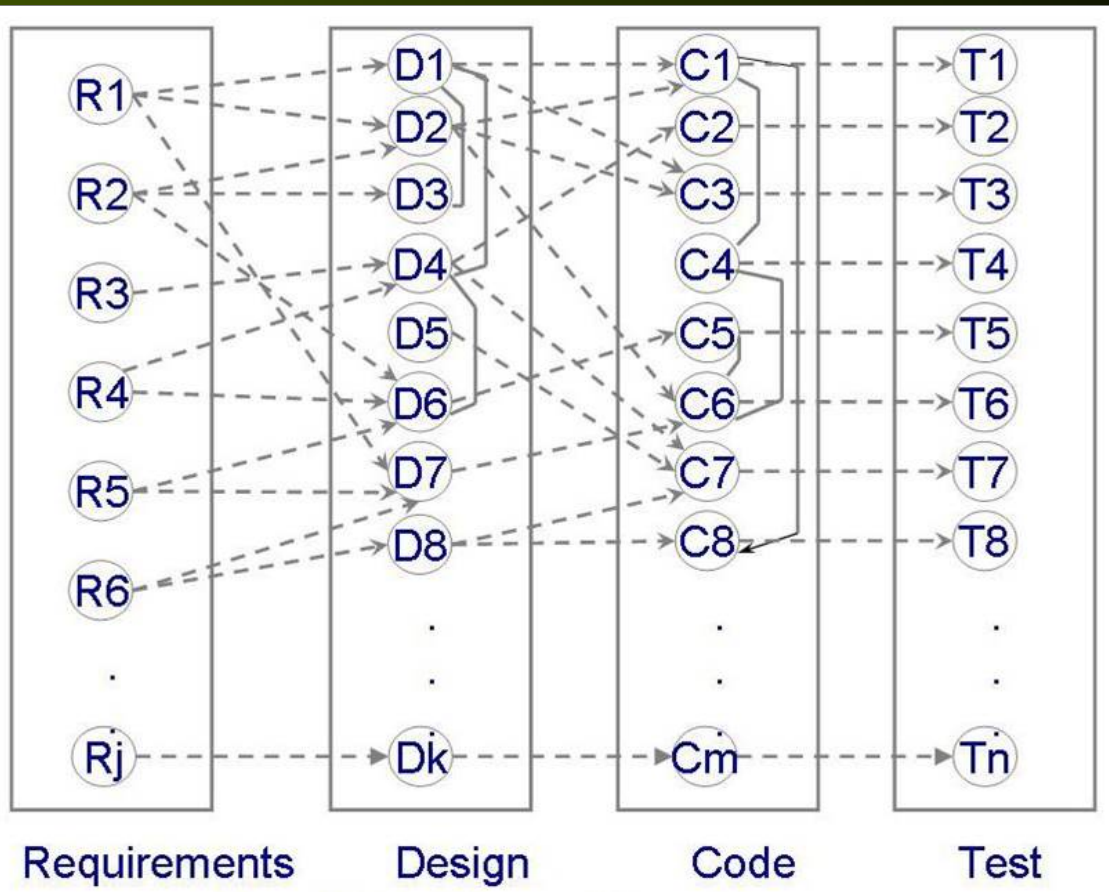


Figura: Trazabilidad Vertical

Herramientas de mantenimiento de software

Clasificación de los tipos más comunes

- Editores de texto
- Comparadores de archivos
- Compiladores y encuadernadores (linkers)
- Herramientas de depuración
- Generadores de referencias cruzadas
- Analizadores de código estático
- Repositorios de gestión de la configuración

Herramienta “tipo"

Características

- Incorpora el código fuente, código objeto , lenguaje de control, y archivos de datos necesarios para hacer funcionar un sistema
- Controla más de una versión de un sistema:
  - Se designa una única versión como la de *producción* y no se permite que ninguna la altere
- Sitúa el número de versión y la fecha del último cámbio en el listado del compilador y en el objeto producido en la compilación
- Posee facilidades de reporting, backup y recuperación, además de tres niveles de seguridad de accesos

Rejuvenecimiento del software

Tipos:

- Redocumentación: el análisis estático añade más información
- Restructuración: transformar para mejorar la estructura del código
- Ingeniería Reversa: recrear el diseño y la información de la especificación desde el código
- Re–ingeniería: Ingeniería Reversa+ cambios en la especificación y el diseño para completar el modelo lógico; después generar un nuevo sistema a partir de la especificación revisada y del diseño

Taxonomía del rejuvenecimiento del software

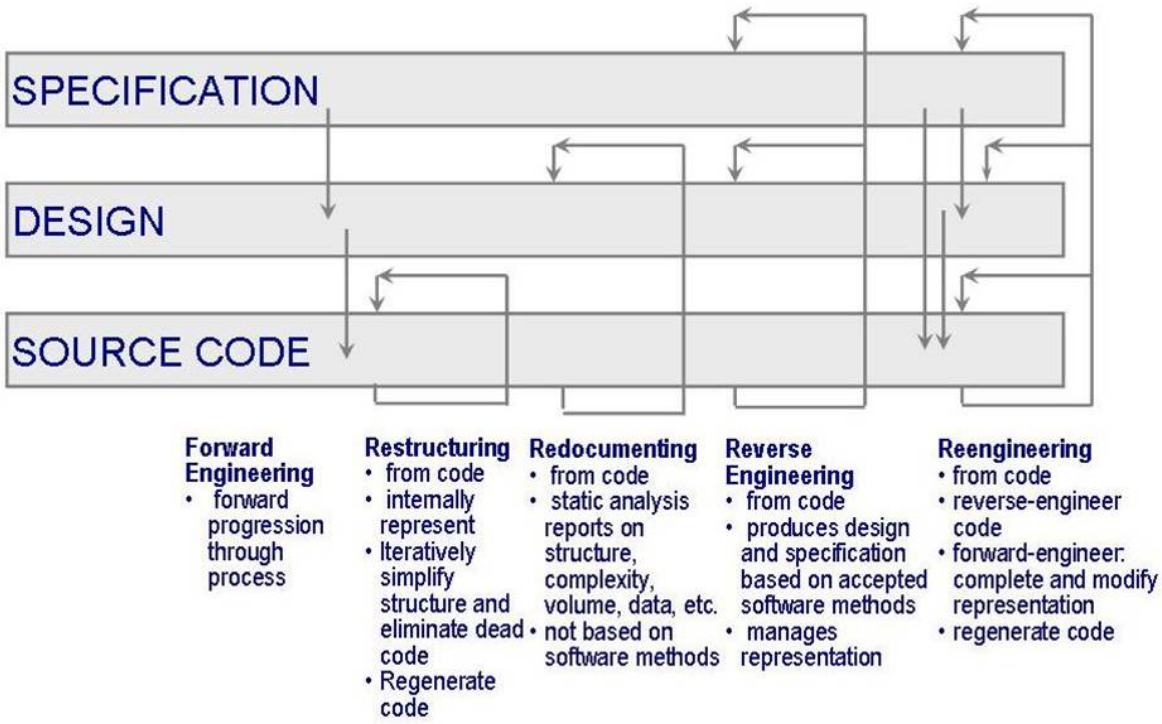


Figura: Relación gráfica entre los 4 tipos de rejuvenecimiento del software

Redocumentación de software

- Comienza enviando el código a una herramienta de análisis
- No está basada en *métodos software*
- La salida puede incluir:
  - relaciones de llamadas entre componentes
  - tablas de interfaces de datos
  - información del diccionario de datos
  - tablas de flujo de datos o diagramas
  - tablas de control de datos o diagramas
  - pseudocódigo
  - caminos de prueba
  - referencias cruzadas entre componentes y variables

Redocumentación de software II

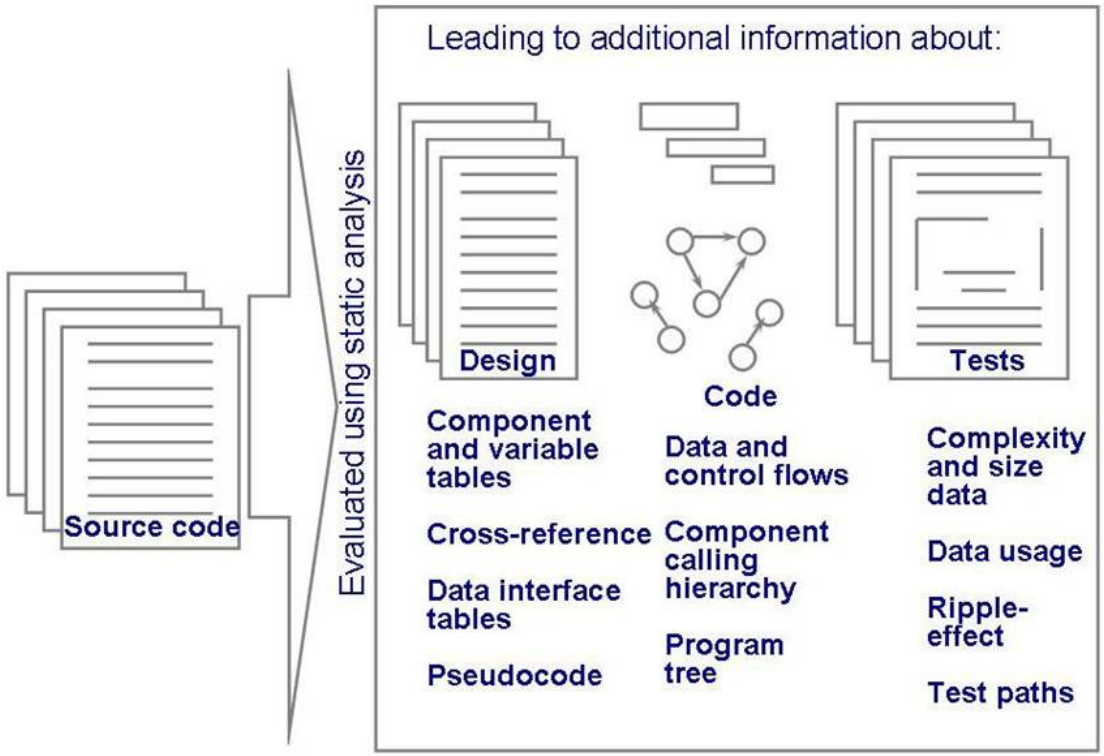


Figura: Proceso de redocumentación

Restructuración de código

- Interpretación del código fuente y representación interna del mismo
- Simplificación de la representación interna
- Regeneración del código estructurado

Actividades de restructuración

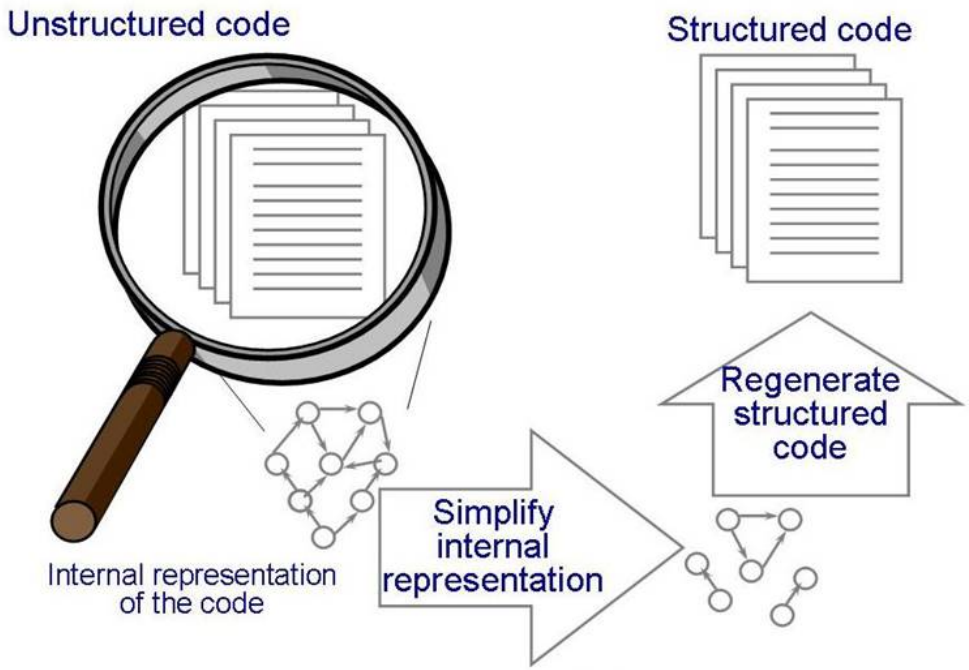


Figura: Proceso de restructuración: (1) análisis estático, (2) simplificación de la representación, (3) refinamiento para generar una versión estructurada

Ingeniería Reversa

- Se basa en métodos software aceptados
- Gestiona la representación
- Intentar recuperar información de ingeniería basándose en métodos de especificación y diseño de software
- Persisten obstáculos por superar antes de que la ingeniería se pueda utilizar en cualquier caso
  - Problema del Sistema de Tiempo Real
  - Sistema extremadamente complejo

Ingeniería Reversa II

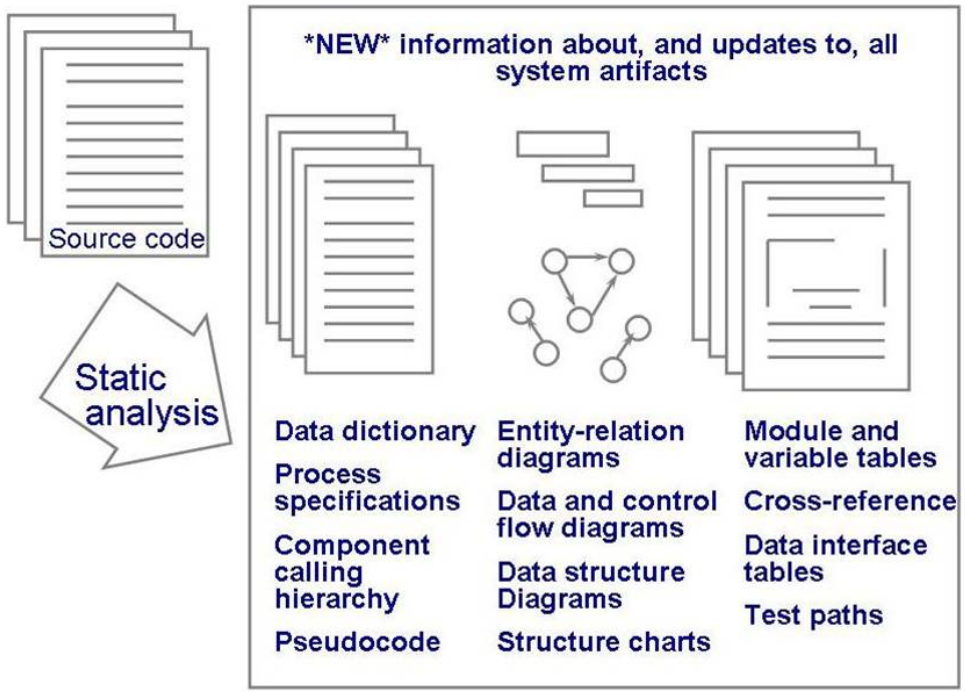


Figura: Proceso de ingeniería reversa

Reingeniería

- Se trata de una extensión de la ingeniería reversa
  - produce nuevo código–software sin cambiar la función del sistema completo
  - completa y modifica la representación
- Pasos de Reingeniería:
  - Al sistema se le aplica ingeniería reversa
  - El sistema software es corregido o completado
  - Se genera el nuevo sistema

Reingeniería II

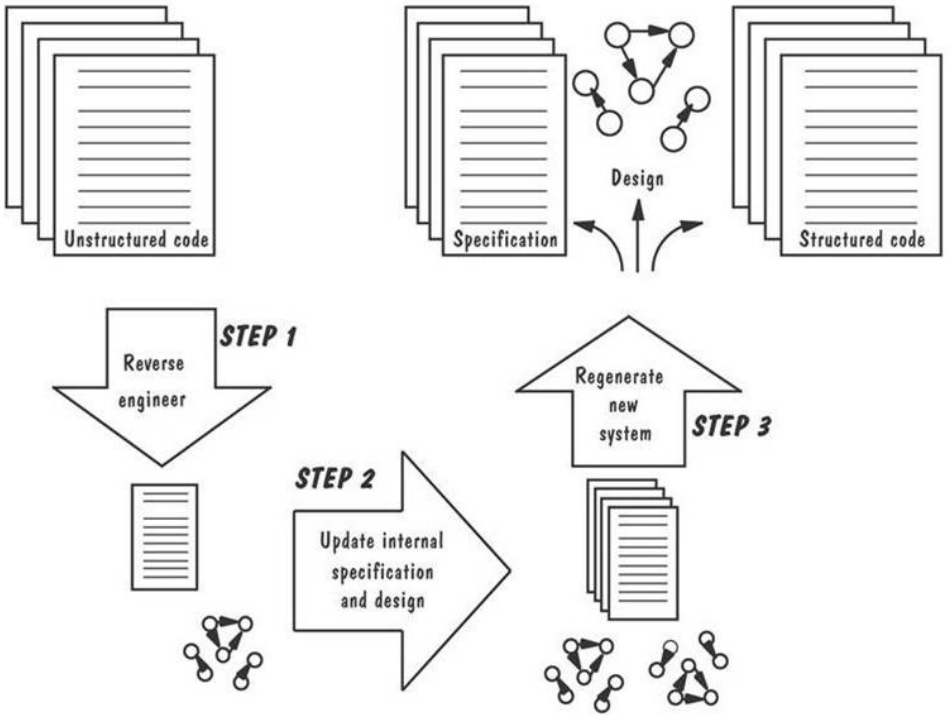


Figura: Proceso de reingeniería

Gerencia de la Configuración de Software

Definición de IEEE (2004)

Es un soporte al proceso del ciclo de vida del software que beneficia a la gestión del proyecto, el desarrollo del producto y las actividades de mantenimiento, además de estar íntimamente ligado a las actividades de *aseguramiento de la calidad del software*.

IEEE (Institute of Electrical and Electronic Engineers) han desarrollado varios estándares para seguir en el proceso de desarrollo, prueba y evolución del software

Gerencia de la Configuración de Software

Definición de IEEE (2004)

Es un soporte al proceso del ciclo de vida del software que beneficia a la gestión del proyecto, el desarrollo del producto y las actividades de mantenimiento, además de estar íntimamente ligado a las actividades de *aseguramiento de la calidad del software*.

IEEE (Institute of Electrical and Electronic Engineers) han desarrollado varios estándares para seguir en el proceso de desarrollo, prueba y evolución del software

Gerencia de Configuración II

Basada en estándares

- La GC siempre debería estar basada en un conjunto de estándares que son aplicados dentro de la organización
- Los citados estándares deberían definir:
  - Cómo se identifican los elementos,
  - Cómo se controlan los cambios y
  - Cómo se manejan las nuevas versiones del sistema

Estándares relacionados con la GC

Estándar	Descripción
1042–1987	Guía IEEE para Gestión de Configuración de Software
828–2005	Planes de Gestión de Configuración de Software
1008–1987	Unidad de Prueba de Software
1012–2004	Planes para Validación y Verificación de Software
1028–2008	Revisión y Auditoría de Software
1004–1993	Clasificación de Anomalías de Software
1059–1993	Guía de Planes de Verificación y Validación de Software
829–1998	Documentación de Pruebas de Software

Tabla: Estándares de IEEE

Estándares relacionados con la GC II

Estándar	Descripción
10007–2003	Gestión de calidad de sistemas (Guías de GC)
15846–1998	Tecnología de la Información– Proceso de Ciclo de Vida (GC)

Tabla: Estándares ISO Software Engineering

Estandarización de las prácticas para gestión de servicios en IT

ITIL

- “Information Technology Infrastructure Library" , conjunto de prácticas estandarizadas que alinean servicios–IT con la necesidades de un negocio
- Publicaciones ITIL (2011)(ver http://www.itil-officialsite.com), cada una cubre una etapa del ciclo de vida de gestión de servicios
- Conforme con estándar ISO/IEC 20000, internacional de referencia para servicios–IT

Estandarización de las prácticas para gestión de servicios en IT (2)

CMM

- “Capability Maturity Model", modelo de desarrollo de software creado a partir de datos de organizaciones que mantienen contratos con el DoD (USA)
- El modelo es la base de la actividades desarrolladas por Software Engineering Institute (SEI), creado por Carnegie–Mellon University
- *Madurez* es el grado de formalización y optimización de procesos implicados en el desarrollo de software
- Métricas de gestión de los resultados para conseguir la optimización de los procesos aludidos

Marcos de trabajo I

áreas de proceso	Actividades	ITIL	RUP	CMM	IEEE
Gerencia y Planificación	1.1.Entender el contexto organizacional	X	X	X	X
	1.2.Preparar el plan de GCS	X	X	X	X
	1.2.1.Definir restricciones Guías y organizacional	X	X	X	X
	1.2.2.Establecer procedimientos y políticas	X	X	X	X
	1.2.3.Establecer roles y responsabilidades	X	X	X	X
	1.2.4.Establecer tiempos y cronogramas	X	X	X	X
	1.2.5.Selección de herramientas	X	X	X	X
	1.2.6.Control de proveedores	X	X	—	X
	1.3.Revisar resultados de auditorias	X	X	X	X
	1.4.Realizar mediciones para mejorar	X	X	X	X

Marcos de trabajo II

áreas de proceso	Actividades	ITIL	RUP	CMM	IEEE
Identificación y Almacenamiento	2.1.Seleccionar elementos a ser controlados	X	X	X	X
	2.2.Identificar versiones	X	X	X	X
	2.3.Establ. líneas base	X	X	X	X
	2.4.Definir Almacenamiento	X	X	X	X
Control de Cambios, versiones	3.1.Requisitos de cambios	X	X	X	X
	3.2.Aprobar cambios	X	X	X	X
	3.3.Implementar cambios	X	X	X	—
Estados	4.1.Informe del estado	X	X	X	X
	4.2 Reportes diversos	X	X	—	X
Auditoría	5.1.Verificar si se cumplen procedimientos y políticas	X	X	X	X
	5.2.Verificar el Almacenamiento	X	X	X	—
	5.3.Verificación líneas base	X	X	X	X
	5.4.Revisión funcionamiento herramientas	X	X	X	X

Gestión de Versiones

Antecedentes

- El administrador del sistema ha de poder identificar siempre la *versión operativa actual* del sistema y el *número de revisión de cada componente* en uso
- Garantizar que todos los procedimientos que afectan a los cambios se aplican adecuadamente:
  - Se asigna un número a cada versión activa del sistema
  - Se asigna un número de revisión a cada componente que resulte cambiado
  - Se guardan registros de cada versión, estado de componente e histórico de cambios en el sistema

Herramientas CASE para control de cambios I

Gestión de Cambios:

- Editores de formularios para soportar los formatos de solicitud de cambio.
- Sistemas basados en *flujo de trabajos* (WF) para definir quién hace qué y automatizar la transferencia de información.
- Base de Datos de cambio que gestionan las propuestas de cambio y está enlazadas con un sistema de gerencia de versiones.

Herramientas CASE para control de cambios II

Identificación de versiones y distribuciones

- Los sistemas asignan identificadores automáticamente cuando se libera una nueva versión del sistema.
- Gestión del Almacenamiento. Los sistemas almacenan las diferencias entre las versiones más que todo el código de la nueva versión.
- Registro de la Historia de Cambios. Registra las razones para la creación de una nueva versión.
- Desarrollo Independiente. Sólo se puede permitir una versión a la vez para el cambio, por lo que se realiza un trabajo paralelo en diferentes versiones.

Para ampliar


Black (2007).
Pragmatic Software Testing.
Wiley.


Everett and Raymond (2007).
Software Testing.
Wiley.


Lewis (2004).
Software Testing and Continuous Quality Improvement.
Auerbach.


Perry (2005).
Effective Methods for Software Testing.
Wiley.


Pressman, R. (2010).
Software engineering: a practitioners approach.
McGraw-Hill.


Spiller (2007).
Softwre Testing Process: Test Management.
Rocky Nook.