Ingeniería del Software II

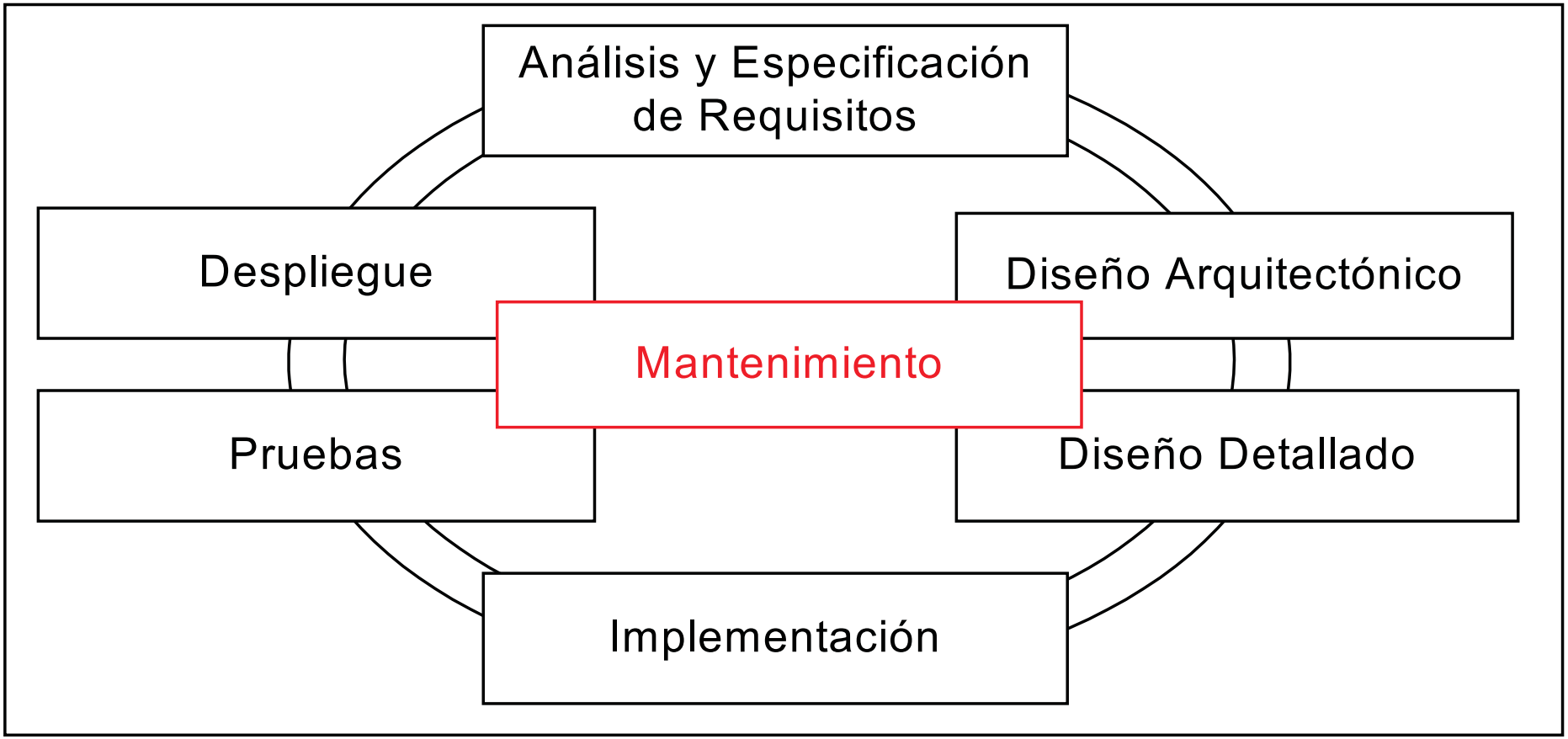
Tema 08. Mantenimiento de Sistemas Software

Pablo Sánchez Barreiro

DPTO. DE MATEMÁTICAS, ESTADÍSTICA Y COMPUTACIÓN

p.sanchez@unican.es

Mantenimiento Software



Importancia del Mantenimiento Software

- Identificado como una de las causas de la *crisis del software* [7].

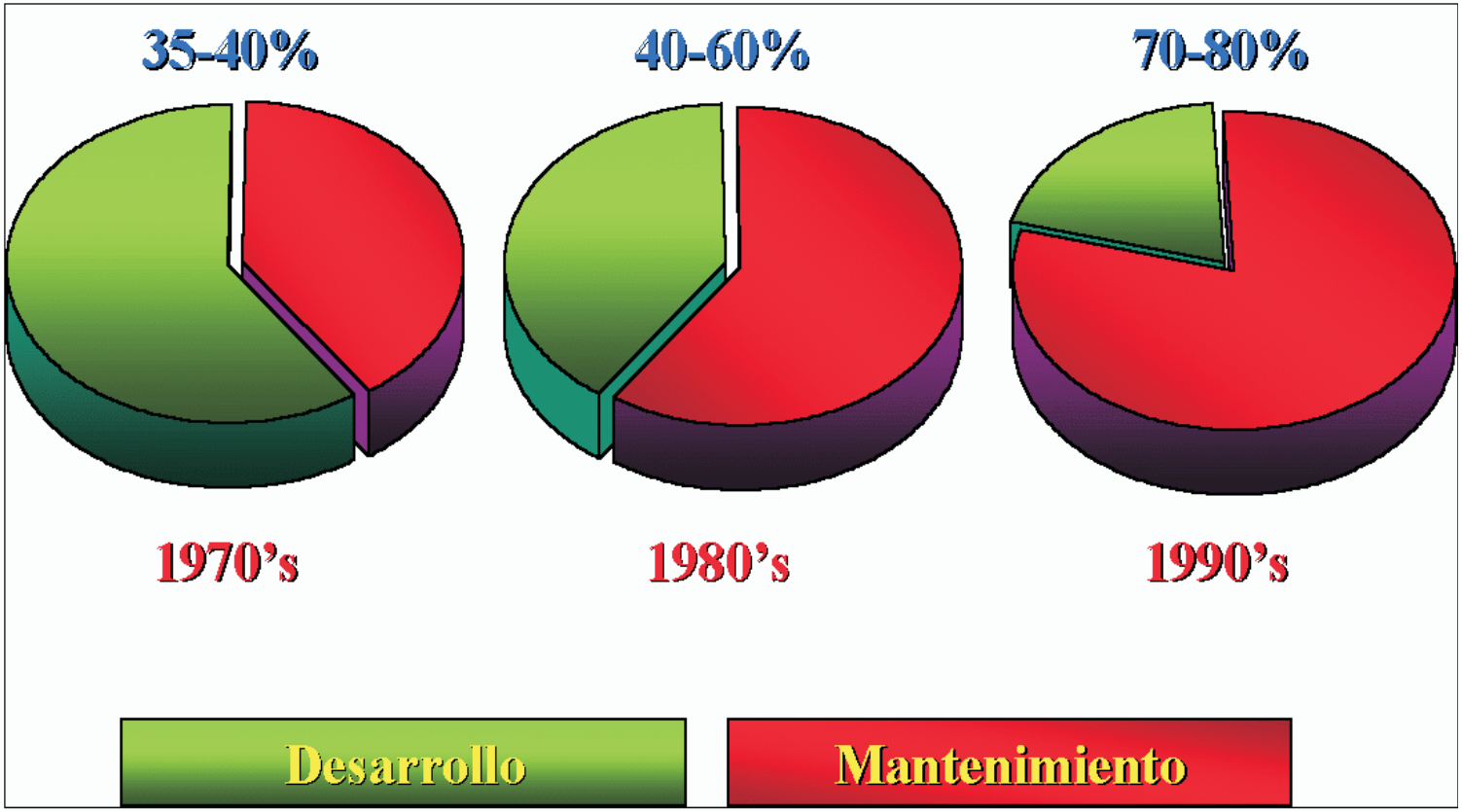
Mantenimento software vs sistemas clásicos

- Permiten pequeñas adaptaciones, pero su modificación es muy costosa.
- Suele implicar la construcción de un nuevo sistema.

Mantenimento software vs sistemas clásicos

- El coste material de un producto software es despreciable.
- Un producto software es altamente modificable. Hasta el punto de poder llegar a ser un producto completamente nuevo (ej. Linux).

Costes históricos mantenimiento software

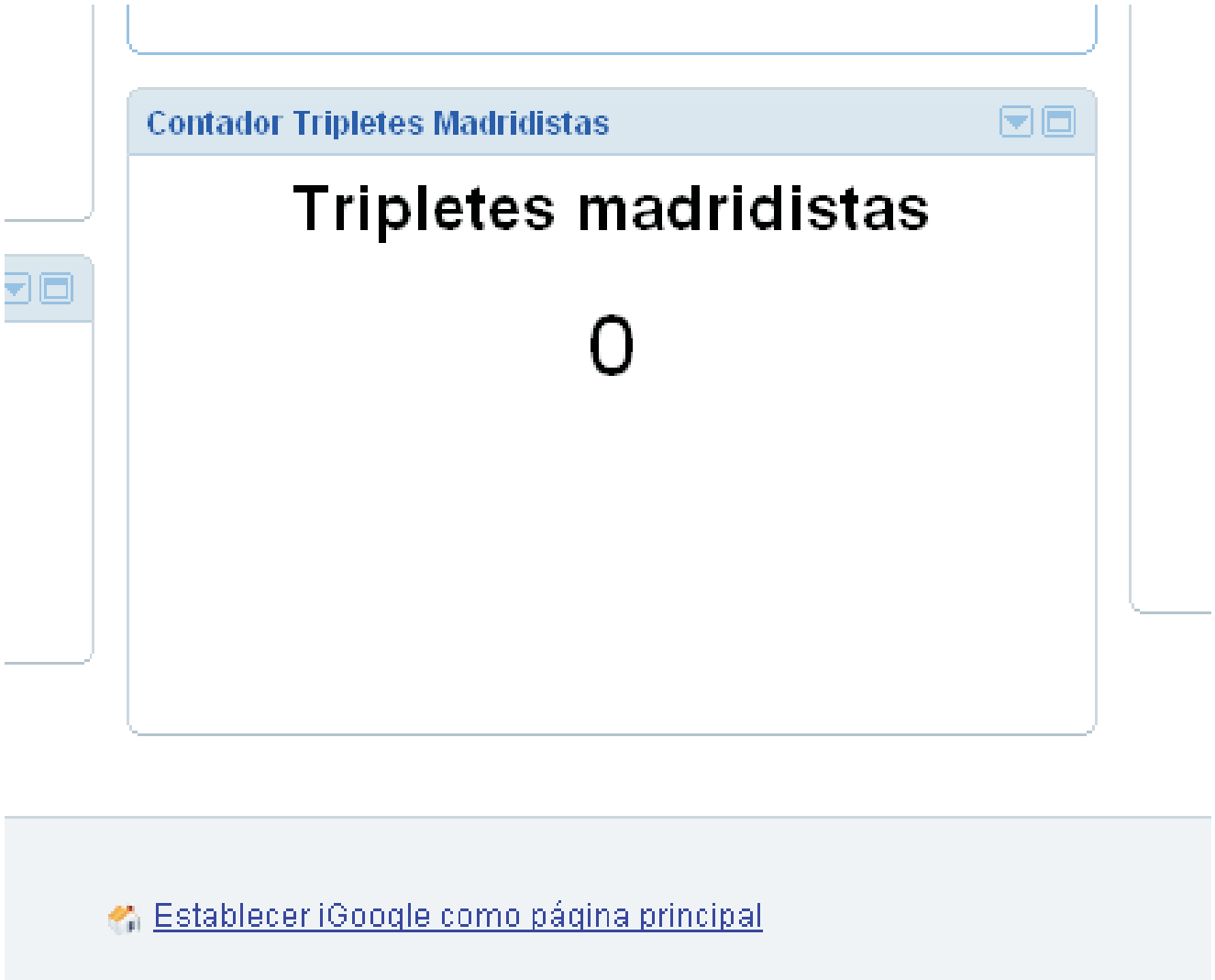


- Listas de bugs y foros de soporte en productos comerciales .
- Actualizaciones de productos comerciales (ej. Windows Update, Acrobat Update)

Importancia mantenimiento software

- Adaptación al euro - Efecto 2000.
- Pantallas táctiles móviles.
- Sistemas de tarificación telefonía móvil.
- Negocios de software libre basados en consultoría y mantenimiento (ej. openArchitecureWare `http://www.openarchitectureware.org/`).
- Actualmente, facilidad de mantenimiento, adaptabilidad y evolución son las características más importantes de la mayoría de los sistemas software.

En Ingeniería del Software no hay verdades universales



Este no es el objetivo del tema

Esto tampoco es el objetivo del tema

Objetivos del tema

Objetivos

Aprender a analizar, planificar, ejecutar y gestionar acciones de mantenimiento software.

Definiciones mantenimiento software

Estándar IEEE 1219

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 [9].

Estándar ISO/IEC 14764

Conjunto de actividades destinadas a proporcionar soporte económicamente rentable para un determinado producto software. Estas actividades se realizan tanto antes de la entrega del producto como después de la entrega del mismo. Las actividades previas a la entrega incluyen las actividades destinadas a planificar, anticipar y preparar actividades de mantenimiento posteriores. Las actividades posteriores a la entrega incluyen modificaciones del producto software, formación y asistencia al usuario [8].

Definiciones mantenimiento software

Mantenibilidad (Maintainability)

Capacidad de un producto software de ser modificado. Estas modificaciones incluyen correcciones, mejoras, o adaptaciones a cambios en el entorno, los requisitos o las especificaciones funcionales [8].

Efecto dominó (ripple effect)

Un determinado cambio en un producto software se dice que genera un *efecto dominó* cuando a consecuencia del cambio debemos realizar cambios adicionales en el sistema.

Estabilidad de un diseño software

Capacidad de resistencia al efecto dominó que tendrá un sistema software derivado de dicho diseño cuando es modificado [13].

Definiciones mantenimiento software

Sistema Heredado (Legacy Systems)

Un *sistema heredado* es un método, tecnología, computador o aplicación antiguo que continúa en uso porque aún satisface las necesidades de los usuarios, aún existiendo nuevas tecnologías o métodos más eficientes disponibles (ej. aplicaciones COBOL en banca, Amadeus).

Tipos de acciones de mantenimiento software

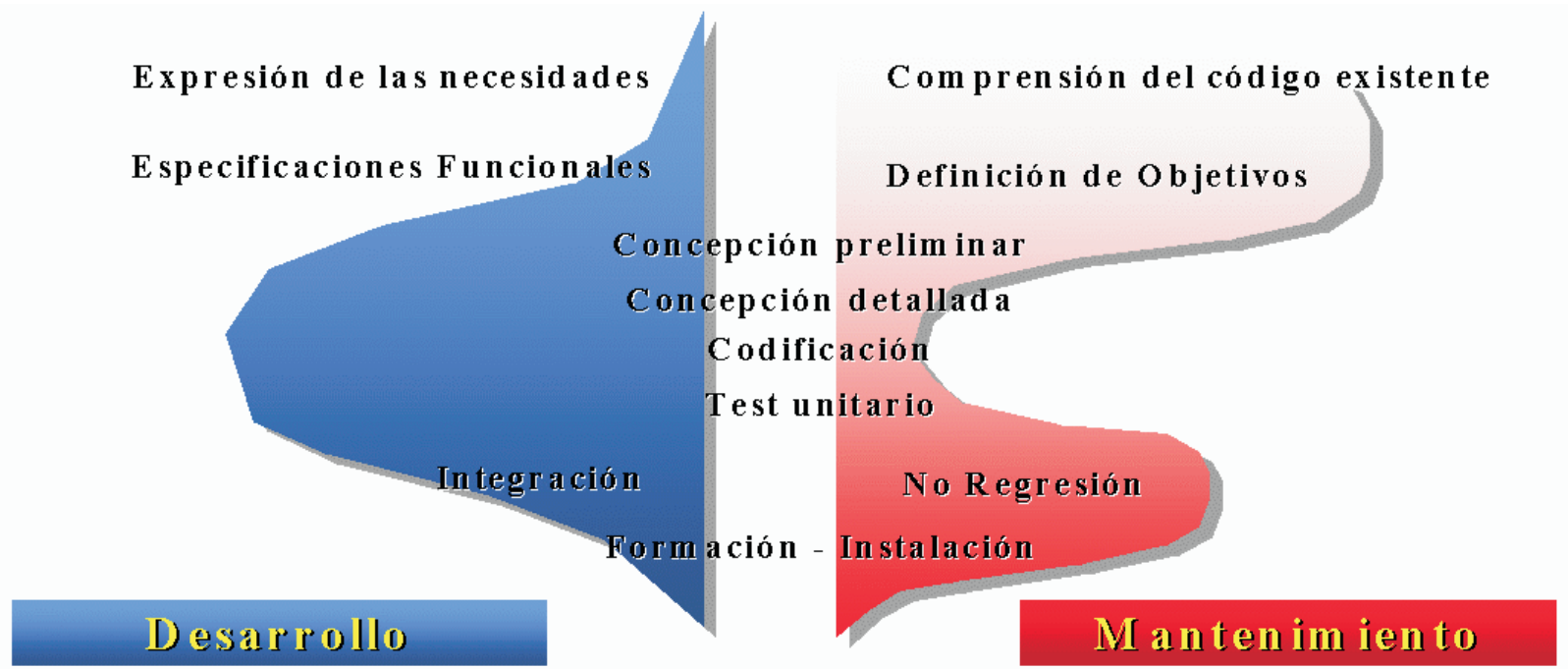
1. **Correctivo**: localiza y corrige defectos en un programa tras su entrega (ej. IVA al 15 %, agujeros de seguridad). Puede ser urgente o no urgente.
2. **Adaptativo**: Modificación para adaptarse a un cambio en el entorno (ej. Euro, pantallas táctiles).
3. **Perfectivo**: Modificación para detectar y corregir fallos latentes antes de que se conviertan en carencias [8]. Modificación para modificar o añadir nuevas funcionalidades [11]. (Ej. firma digital en banca online).
4. **Preventivo**: Modificación para detectar y corregir fallos latentes antes de que se conviertan en fallos operacionales [8]. Mejorar las propiedades del software [11]. (Ej. recodificar para aplicar patrones de diseño).

Tipos de cambios en productos software

1. **Anticipado**: Un cambio se dice que es anticipado cuando ha sido previsto durante el desarrollo del sistema software y se han adoptado decisiones de diseño que permiten acomodar la variación (ej. uso de constantes, patrón estrategia, sistema de plugins de Eclipse).
2. **No anticipado**: Todo aquel que no es anticipado. *There is not silver bullet* [4].

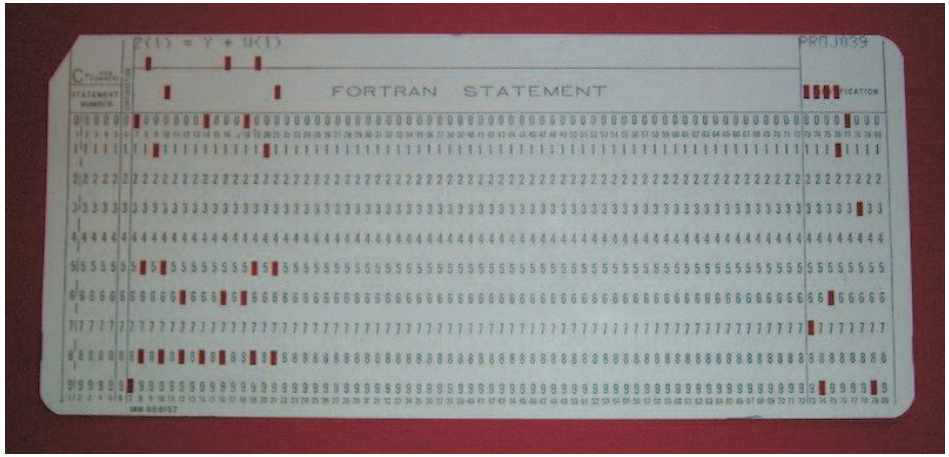
Principales Retos Mantenimiento Software

1. *El efecto iceberg* (o el efecto usillos/reforma).
2. No es lo mismo producir que mantener:
   - Diferentes medidas de productividad entre desarrollo y mantenimiento:40 LDC desarrollado por 1 LDC mantenido [11].



Necesidades en la fase de desarrollo vs fase de mantenimiento

Código heredado (legacy code)



- Desarrollado con tecnologías y técnicas "anticuadas".
- No hay documentación.
- Si la hay, está en notación Benito & Manolo, que ya no trabajan en la empresa.
- Reescribirlo entero no es factible (ej. Amadeus).
- El sistema *no* tiene porque estar bien diseñado, programado, ni haber sido desarrollado siguiendo un proceso de ingeniería.

Cambios en sistemas heredados/indebidamente documentados

Degradación calidad producto software

```
class Bill {

   float amountToBill;

   float calculateTotalCharge() {
      return amountToBill + (amountToBill * (16.0/100.0);
   } // calculateTotalCharge()

} //Bill
```

Degradación calidad producto software

```
class Bill {

   float amountToBill;

   float calculateTotalCharge(int clientType) {
      if (clientType == 0) {
          return amountToBill + (amountToBill * (16.0/100.0);
      } else {
          return amountToBill + (amountToBill * (6.0/100.0);
      }
   } // calculateTotalCharge()

} //Bill
```

Degradación calidad producto software

```
class Bill {

   float amountToBill;

   float calculateTotalCharge(int clientType, int productType) {
      if ((clientType == 0) && (productType == 0)) {
        return amountToBill + (amountToBill * (16.0/100.0);
      } else if ((clientType == 0) && (productType == 1)) {
        return amountToBill + (amountToBill * (4.5/100.0);
      } else if ((clientType == 1) && (productType == 0)) {
        return amountToBill + (amountToBill * (12.0/100.0);
      } else if ((clientType == 1) && (productType == 1)) {
        return amountToBill + (amountToBill * (4.5/100.0);
      }
   } // calculateTotalCharge()
} //Bill
```

Añadiendo iva al total mensual

```
class Bill {

   float amountToBill;

   float calculateTotalCharge(int clientType, int productType) {

      float total, iva;

      if ((clientType == 0) && (productType == 0)) {
        iva = amountToBill * (16.0/100.0);
      } else if ((clientType == 0) && (productType == 1)) {
        iva = amountToBill * (4.5/100.0);
      } else if ((clientType == 1) && (productType == 0)) {
        iva = amountToBill * (12.0/100.0);
      } else if ((clientType == 1) && (productType == 1)) {
        iva = amountToBill * (4.5/100.0);
      }
      total = amountToBill + iva;
      IvaLogger.addIva(iva);
      return total;
  } // calculateTotalCharge()
} //Bill
```

Principales Problemas Mantenimiento Sw (Resumen)

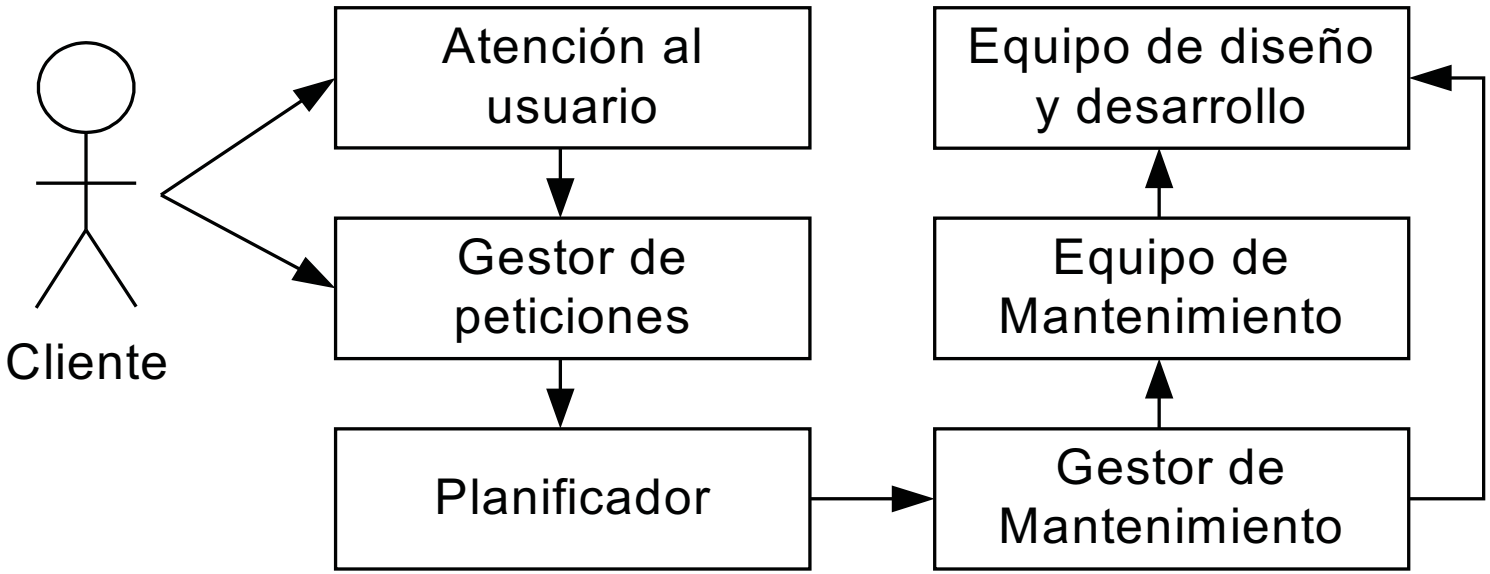
1. Efectos dominó y efecto iceberg.
2. Cambios ad-hoc, ausencia metodológica del cambio.
3. Ausencia de documentación adecuada (decisiones de diseño).
4. Degradación calidad del producto.

Solución para el mantenimiento software

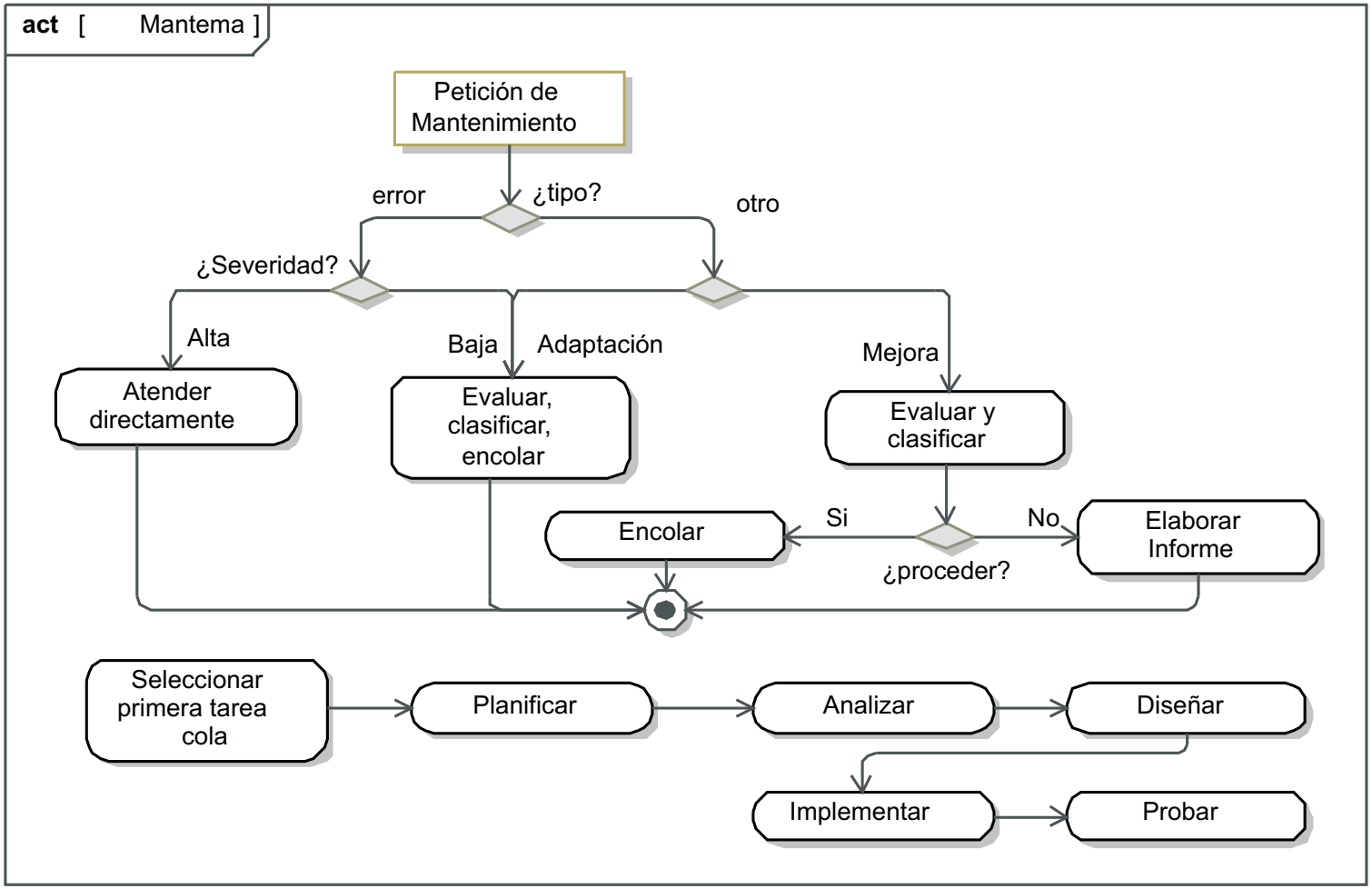
- Establecer procedimientos claramente definidos y estandarizados para el mantenimiento software, que se basen en técnicas y herramientas para el mantenimiento claramente definidas y validadas.
- Asignarle los recursos adecuados, tanto físicos y económicos como humanos,
- Usar técnicas para control de calidad, tanto sobre el producto como sobre el proceso.

Ejemplo: S3M (`http://www.s3m.ca/`).

Organigrama del equipo humano

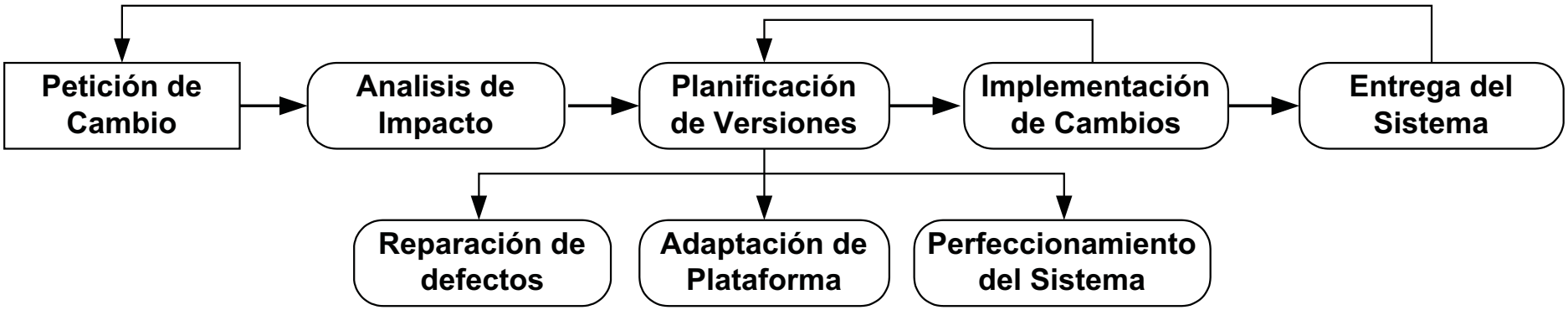


MANTEMA [12]

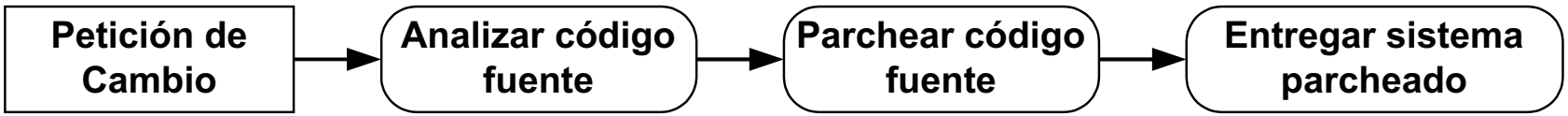

act [Mantema]
Petición de Mantenimiento
¿tipo?
error
otro
¿Severidad?
Alta
Baja
Adaptación
Mejora
Atender directamente
Evaluar, clasificar, encolar
Evaluar y clasificar
Encolar
Si
No
¿proceder?
Elaborar Informe
Seleccionar primera tarea cola
Planificar
Analizar
Diseñar
Implementar
Probar

Lowel Jay Arthur [1]

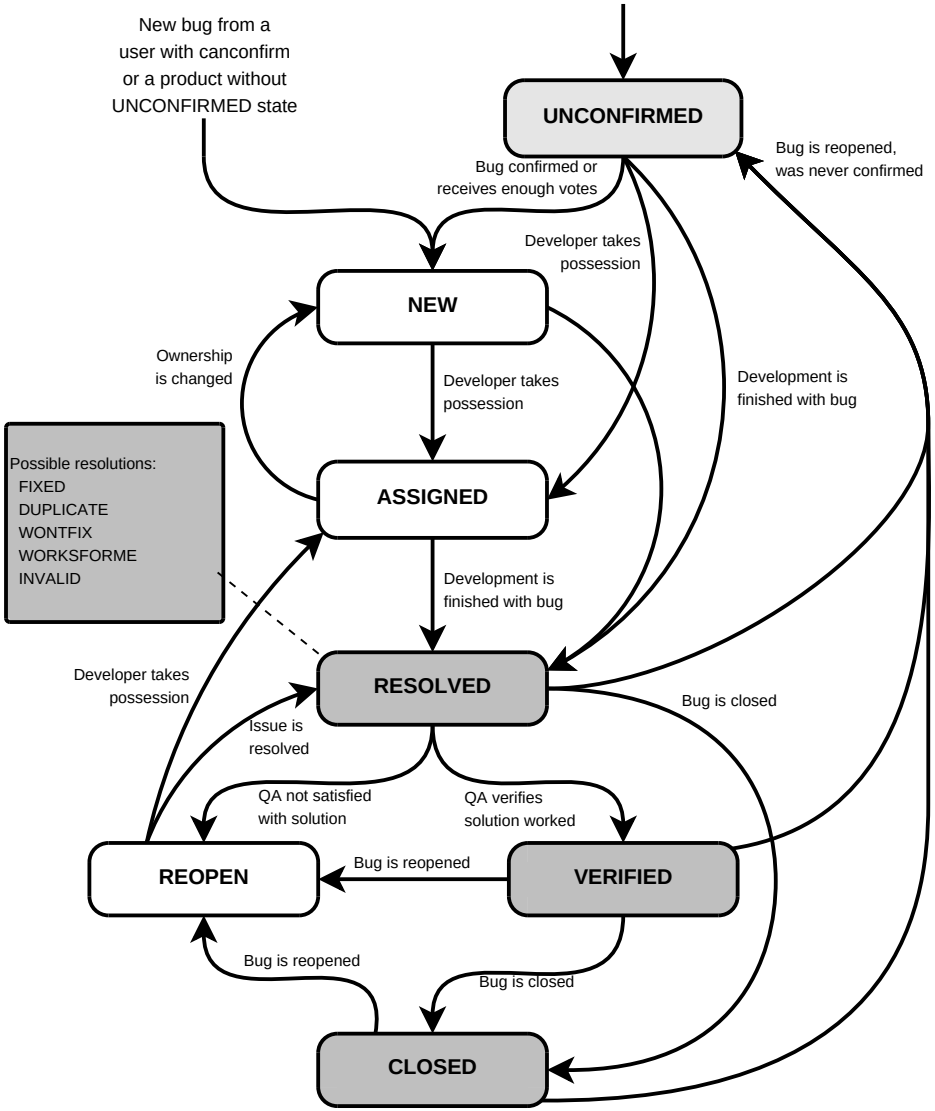
Proceso de evolución de un sistema



Proceso de cambios de urgencia



Gestión de petición de cambio en Bugzilla



Plantillas para documentación cambios software

Tomada de Basili et al [2]

1. Descripción del cambio.
   1. Localización.
   2. Subsistemas afectados.
   3. Módulos afectados.
   4. Entradas/salidas afectadas.
   5. Tamaño.
   6. Líneas de código añadidas, modificadas y eliminadas.
   7. Módulos examinados, añadidos, modificados y eliminados.
   8. Tipo del cambio (Correctivo — Perfectivo — Preventivo — Adaptativo)
2. Descripción del proceso de cambio.
   1. Esfuerzo dedicado.
   2. Experiencia del personal de mantenimiento
   3. Tiempo del personal de mantenimiento trabajando en el sistema.
   4. Tiempo del personal trabajando en el dominio.
   5. Artefactos relacionados con el cambio.
3. Descripción del problema.
   1. Descripción del error.
      - ★ Causa y origen del error.
      - ★ Momento del proceso en que se produjo el error.
   2. Dificultad.
      - ★ Causas que dificultaron la modificación.
      - ★ Actividad más difícil relacionada con la modificación.
   3. Cantidad de esfuerzo desperdiciado.
   4. Decisiones que se podrían haber tomado para disminuir la dificultad de los errores.

Plantillas para documentación cambios software

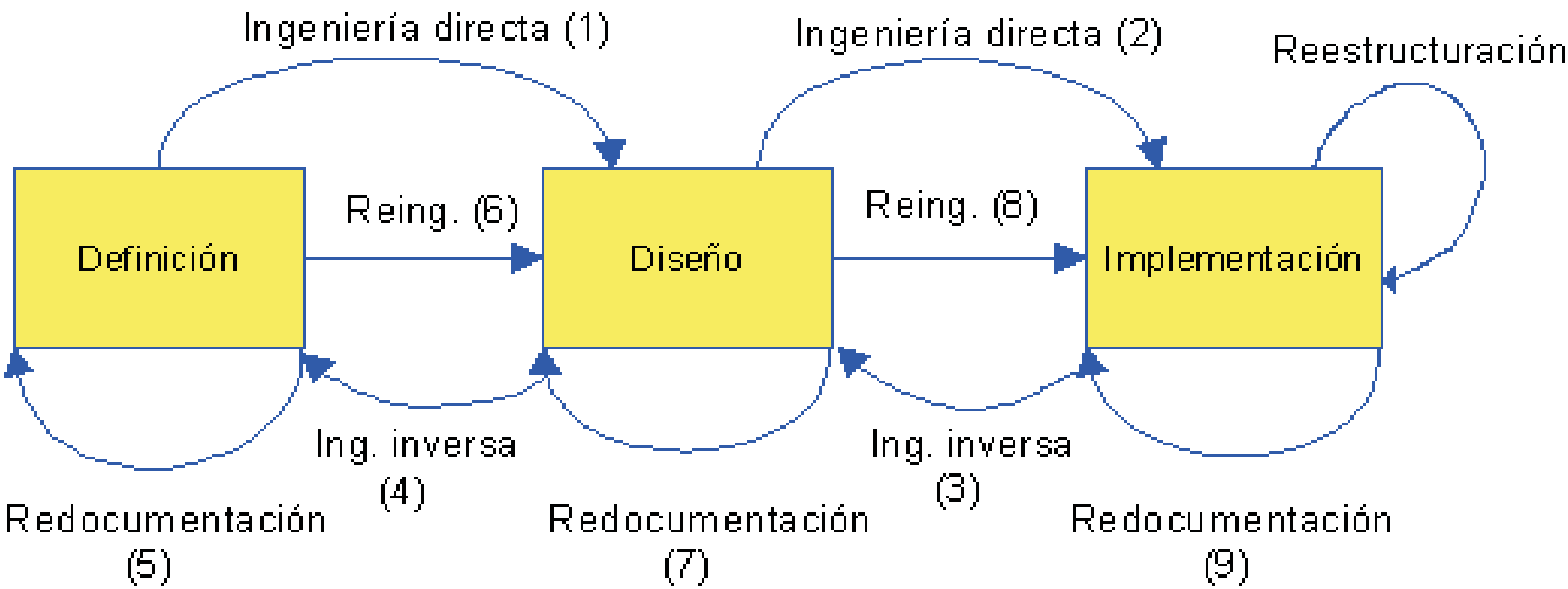
- Plantilla aleatoria encontrada en internet (moodle).
- Plantilla modelo de bugzilla:

  `https://landfill.bugzilla.org/bugzilla-3.6-branch/show_bug.cgi?id=1`

Soluciones técnicas para el problema de mantenimiento

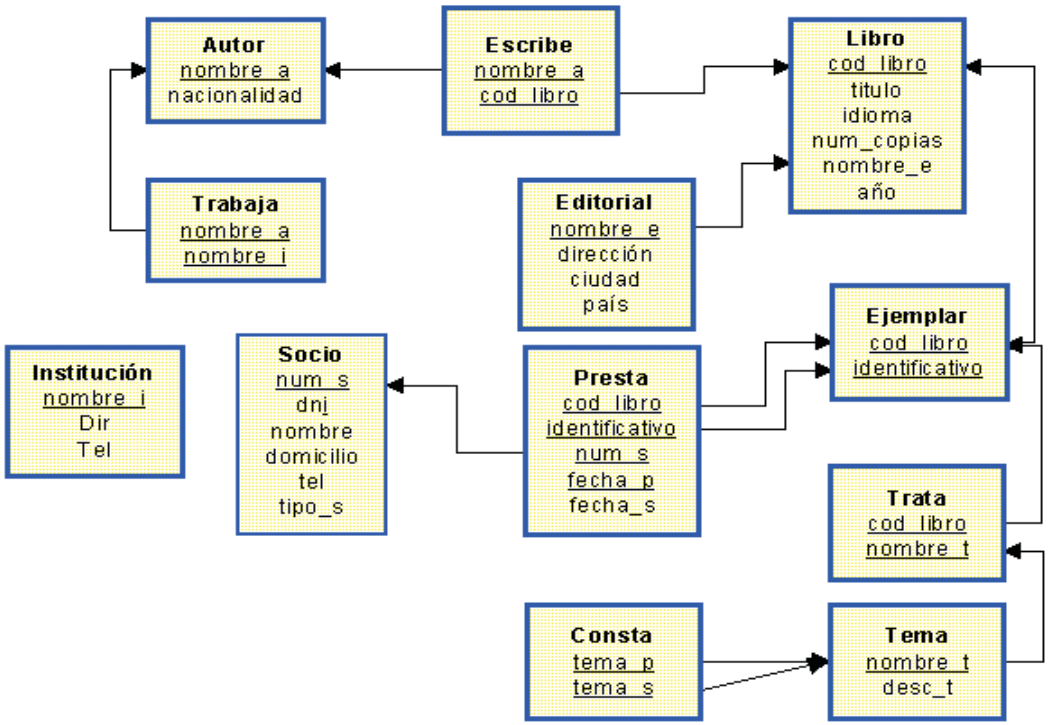
1. **Ingeniería Inversa**: Reconstruir el proceso de ingeniería de un producto a partir de ciertos artefactos de dicho producto.
2. **Reingeniería:** Examen y modificación de un sistema para reconstruirlo en una nueva forma. Puede precisar de un proceso de ingeniería inversa para reconstruir el proceso de ingeniería del producto.
3. **Reestructuración:** Modificación del software para hacerlo más fácil de entender y cambiar o menos susceptible de incluir errores en cambios posteriores.
4. **Transformaciones:** Manipulación, por medio de transformaciones automáticas ejecutadas por un computador, del código o modelo de un sistema para añadirle/modificarle/eliminarle elementos. (ej. pasar un sistema a distribuido).

Soluciones técnicas para el problema de mantenimiento

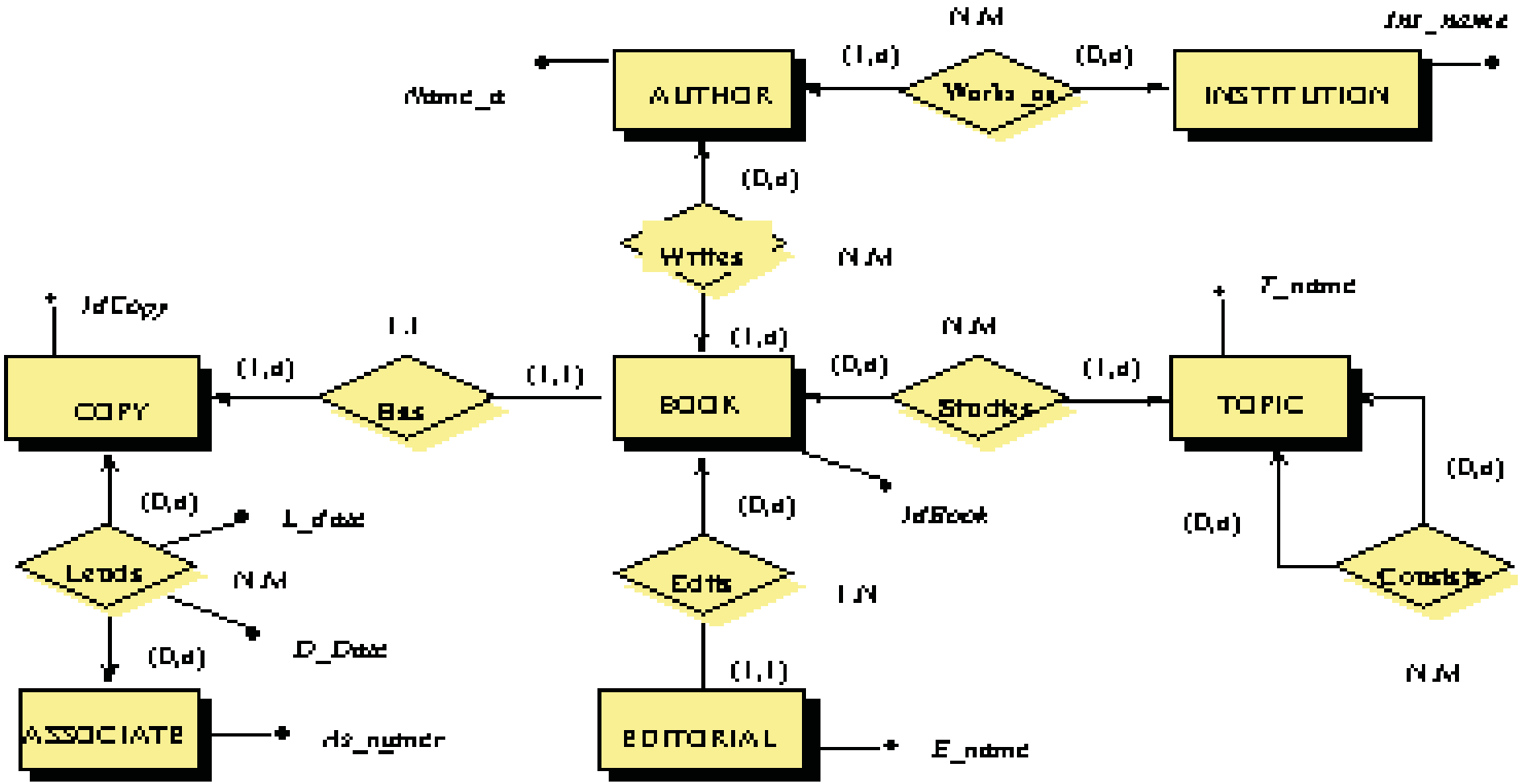


¿Qué es Ingeniería Inversa?

```
********** 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,
 título             título NOT NULL,
 idioma   idiomas NOT NULL,
 num_copias         número_ejemplar NOT NULL,
 nombre_e           nombres NOT NULL,
 año                año,
PRIMARY KEY (cod_libro),
FOREIGN KEY (nombre_e) REFERENCES editorial
          ON UPDATE CASCADE.
```



¿Qué es Ingeniería Inversa?

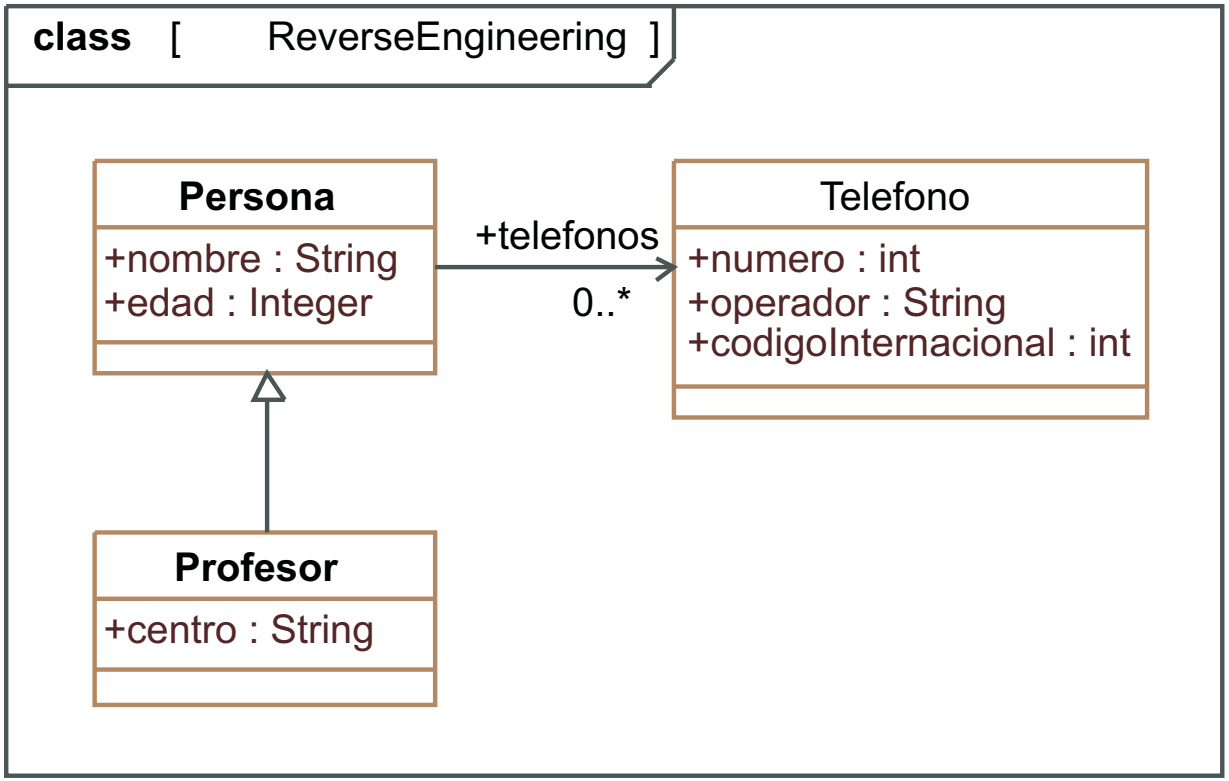


Ingeniería Inversa de código Java a UML

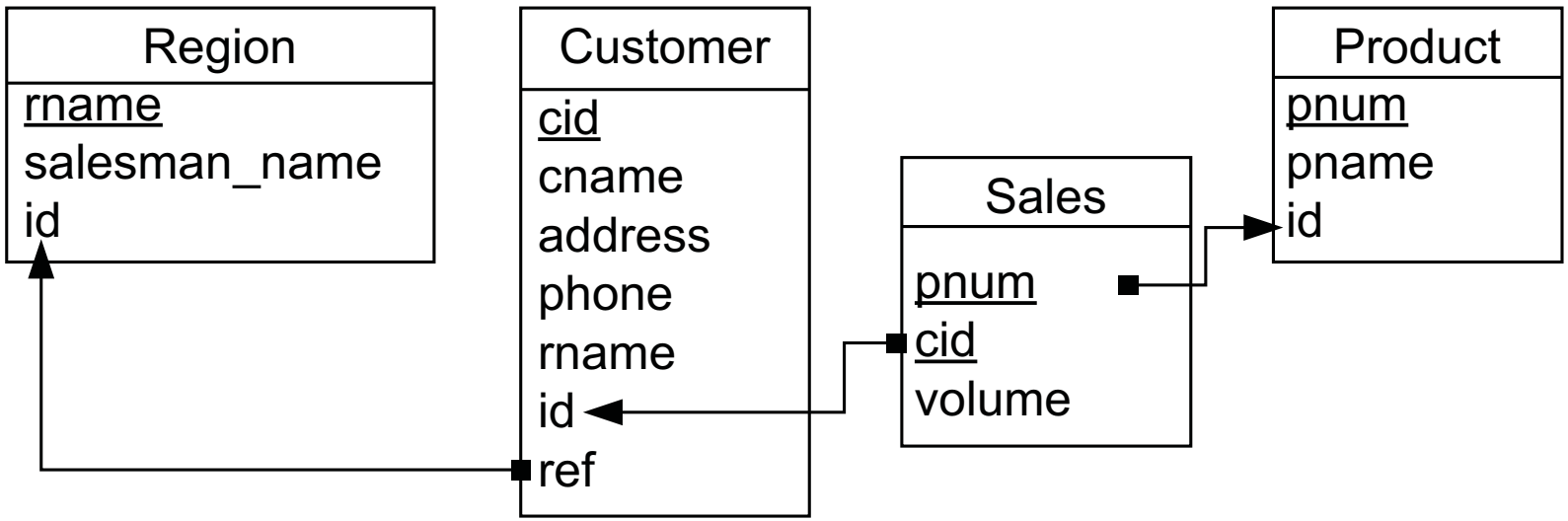
```
class Persona {
    String nombre;
    int edad;
    Set<Telefono> telefonos;

    String getName() {
       ...
    }

    ...
}

class Profesor extends Persona {
    String centro;

}
```

```
class Telefono {
    int numero;
     String operador;
     int codigoInternacional;

}
```

Ingeniería Inversa de código Java a UML



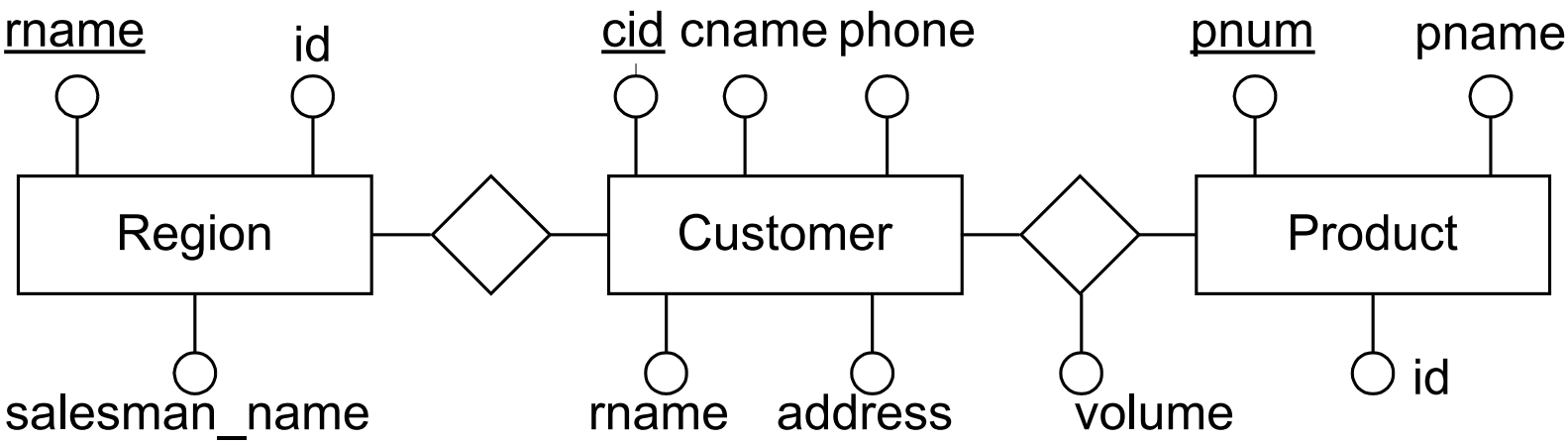
Ingeniería Inversa de bases de datos relacionales



Ingeniería Inversa de bases de datos relacionales

1. Por cada tabla sin referencias a otras tablas, creamos una entidad.
2. Para las entidades creadas, copiamos atributos y marcamos claves primarias. Marcamos las tablas como *procesadas*.
3. Por cada tabla con referencias externas, creamos una entidad.
4. Para las entidades creadas, copiamos atributos y marcamos claves primarias.
5. Por cada clave externa, creamos una relación entre la entidad creada y la referenciada.
6. Para dicha relación, el extremo de tabla referenciada, tendrá cardinalidad 1, salvo restricción expresa.
7. Para dicha relación, el extremo de la entidad creada, tendrá cardinalidad $0..*$, salvo restricción expresa.
8. Si una entidad tiene: (1) su clave primaria formada por la unión de las claves primarias de tablas con las cuales se relaciona; y (2) los extremos de las relaciones con las otras entidades tiene como cardinalidad superior 1, convertir dicha entidad en una relación n a m entre entidades.
9. Copiar los atributos de la entidad a atributos de la relación.

Ingeniería Inversa de bases de datos relacionales



Reestructuración: Refactorings

Refactorización

Proceso de cambio de un sistema software de forma que su comprtamiento externo no se vea afectado pero que mejora su estructura interna (ej. atributos relacionados a clase).

Malos olores (bad smells)

Indicios sobre potenciales problemas en el código (ej. código replicado, mismo método en varias subclases).

Esquema de una refactorización

1. Nombre
2. Síntomas
3. Causas
4. Refactorizaciones propuestas
5. Beneficios esperados
6. Efectos colaterales y contraindicaciones

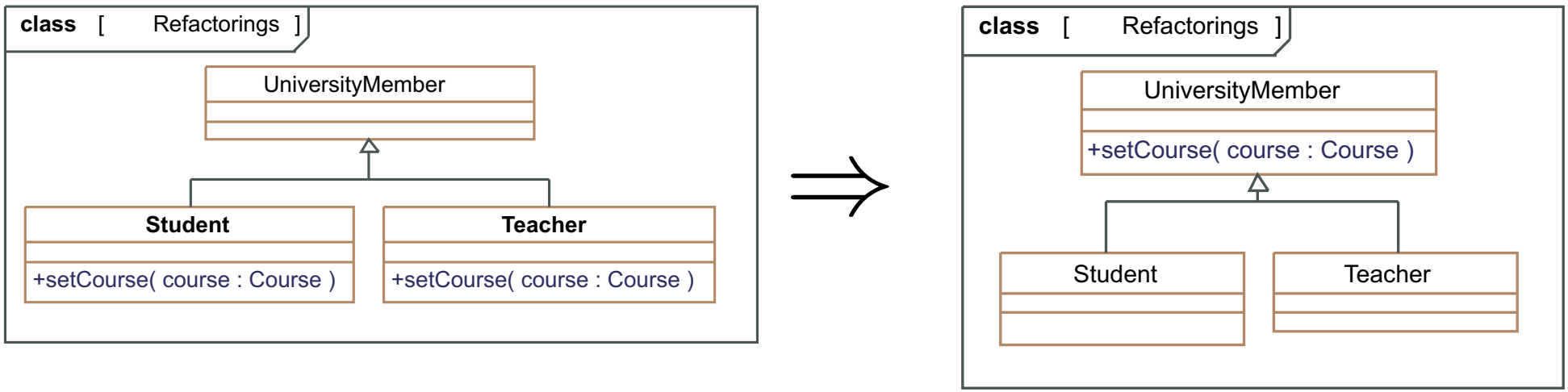
Refactorizaciones más Populares [6]

1. Pull Up Method.
2. Move Method.
3. Add Parameter.
4. Move Field.
5. Rename Method.
6. Rename Field.

Pull Up Method

Descripción

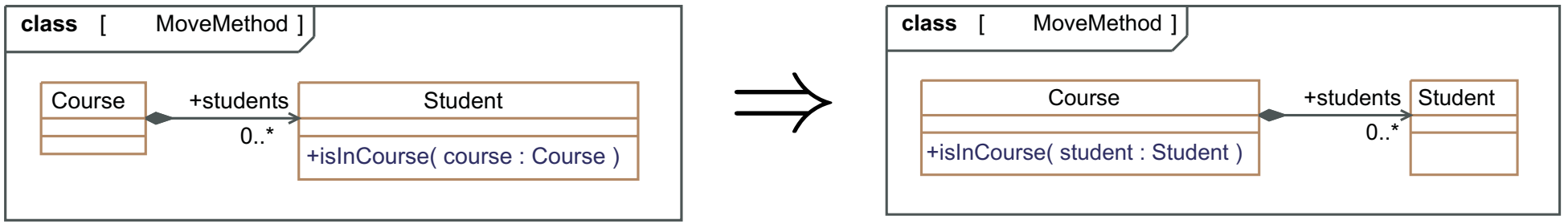
Existen métodos *cuasi*-idénticos en las subclases



Move Method

Descripción

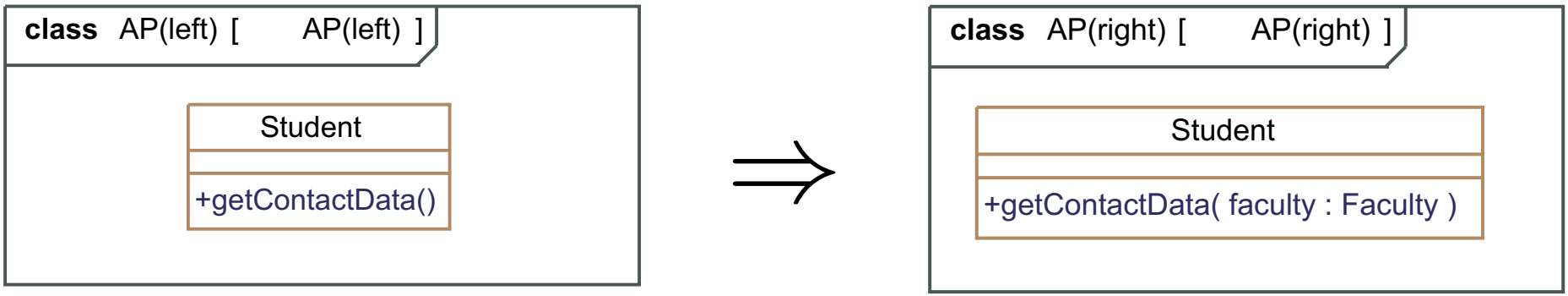
Un método de una clase A es más usado en una clase B que en la clase donde está definido.



Add Parameter

Descripción

A method needs more information from its caller.

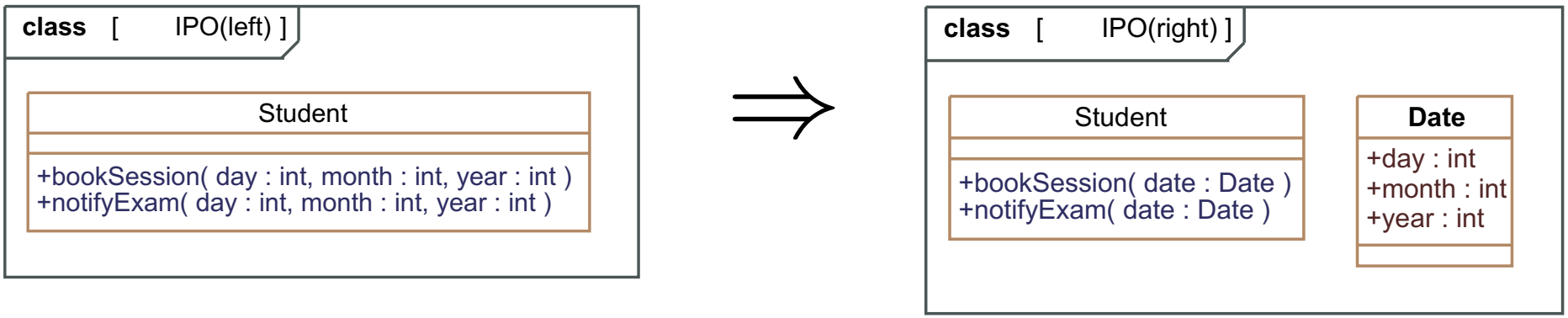


¡ **OJO** ! puede crear listas interminables de argumentos

Introduce Parameter Object

Descripción

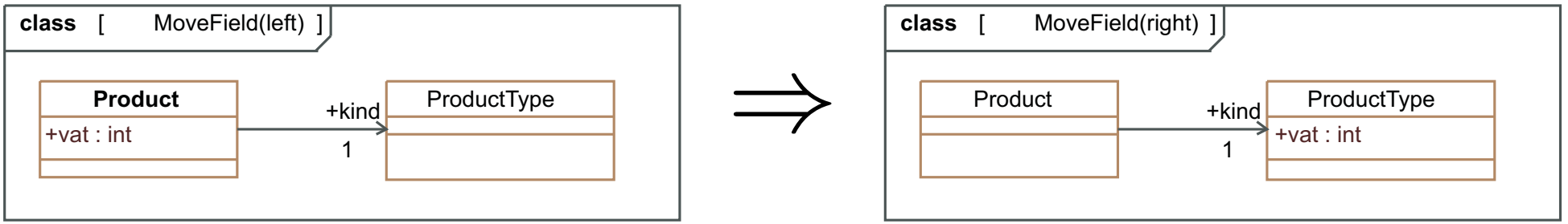
Existen grupos de parámetros que están naturalmente relacionados



Move Field

Descripción

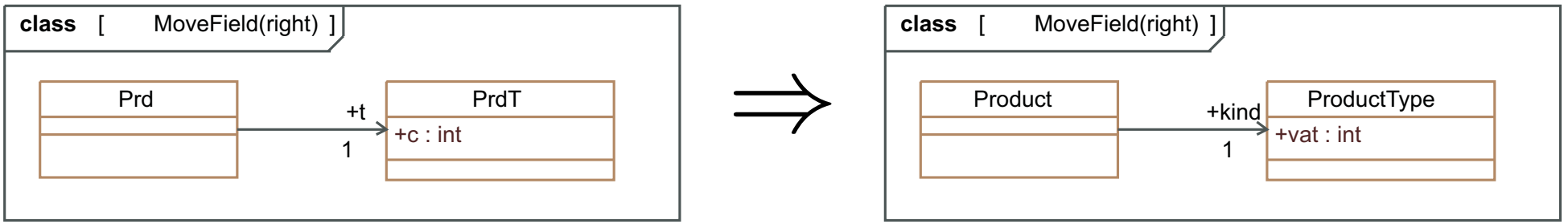
Un atributo de una clase A es más usado en una clase B que en la clase donde está definido.



Rename Field/Method

Descripción

El nombre de un atributo o método no es significativo



Replace Magic Number with Symbolic Constant

Descripción

Tenemos una constante numérica con un significado bien definido

```
double calculateTotalCharge() {
   return this.totalAmount +
         (this.totalAmount*16.0/100.0);
}
```

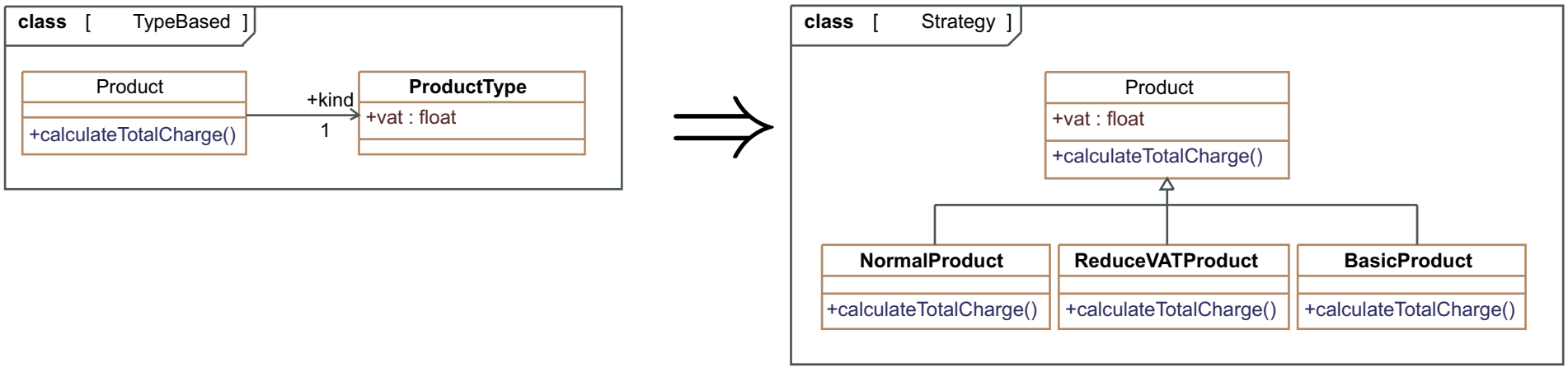
refactors to

```
static final double NORMAL_VAT = 16.0;

double calculateTotalCharge() {
   return this.totalAmount +
         (this.totalAmount*NORMAL_VAT/100.0);
}
```

Replace Type Code with State/Strategy

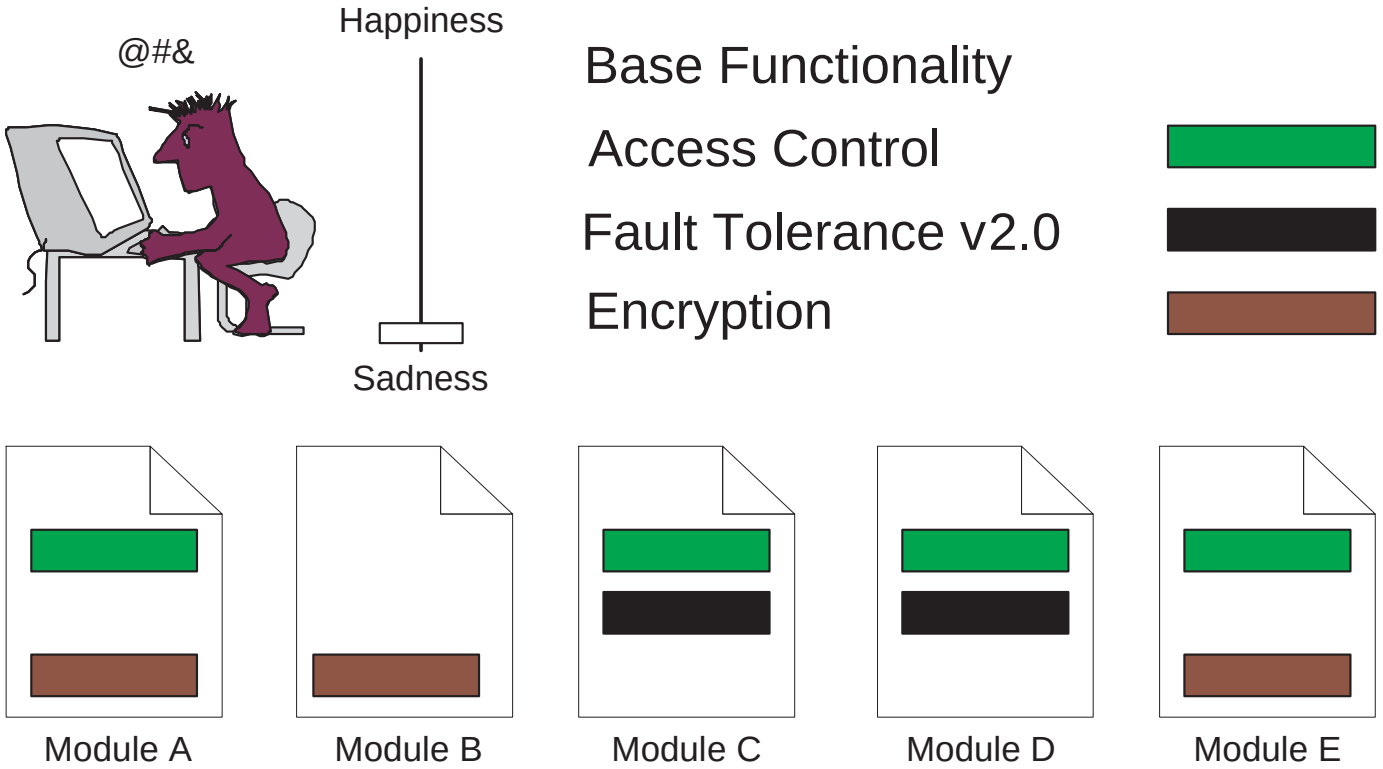
Descripción

Una clase tiene un atributo que indica tipo y que afecta al comportamiento de la clase



Detección de código replicado

Problema del código replicado



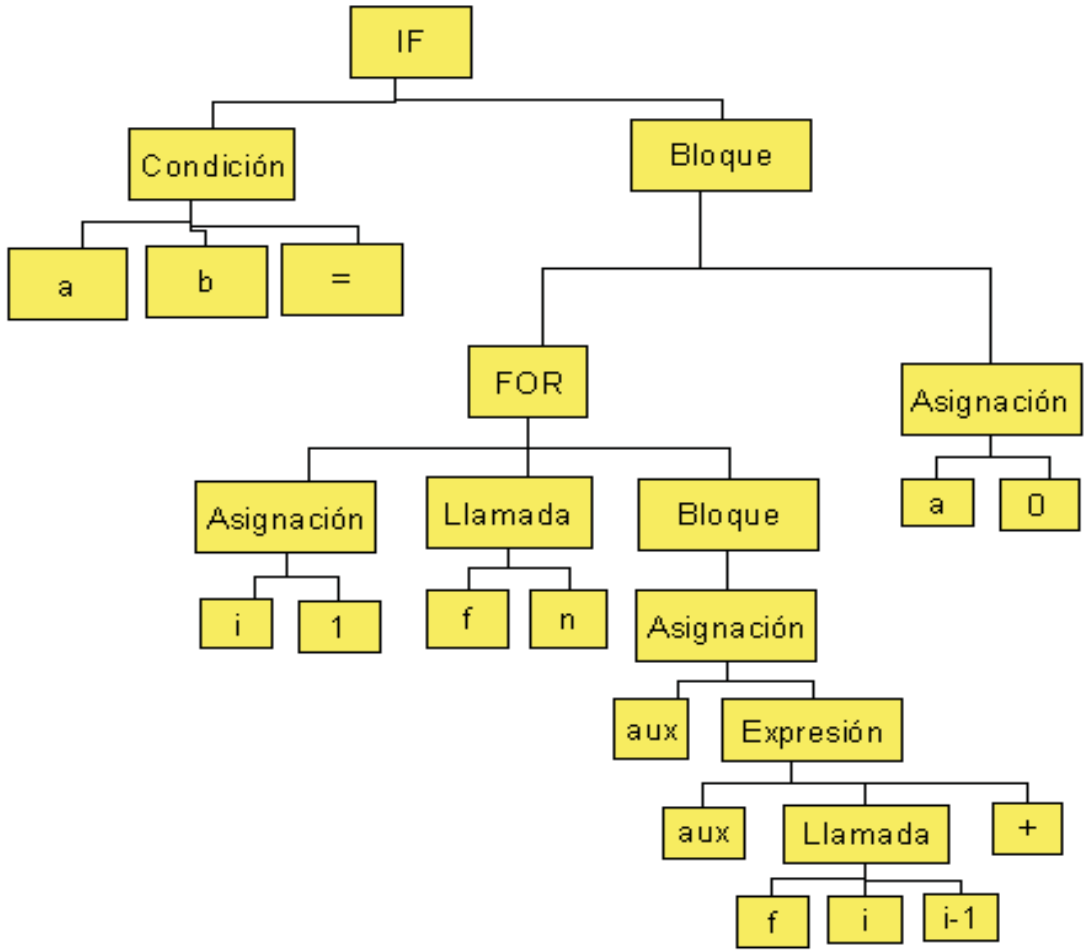
Detección de código replicado (Baxter et al [3])

Fragmento 1 en módulo A	Fragmento 2 en módulo B

`void Anadir(int e, lista l) {`
`int i:=1; boolean enc:=false;`
`while (not enc) {`
`if l[i].ocupado then`
`i++;`
`else enc=true;`
`}`
`l.info:=e;`
`l.ocupado:=true;`
`}`
	`void Insertar(lista l, int info)`
	`{`
	`int cont; boolean enc;`
	`Elemento e;`
	`cont:=1;`
	`enc:=false;`
	`while (enc==false) {`
	`e:=l[cont];`
	`if (not e.ocupado) then {`
	`e.ocupado:=true;`
	`e.info:=info;`
	`l[cont]:=e;`
	`enc:=true;`
	`} else cont:=cont+1;`
	`}`
	`}`

Detección de código replicado (Baxter et al [3])

```
  ...
if a=b then
   for i=1 to f(n)
     aux = aux + f(i, i-1)
   end for
   a=0
end if
  ....
```



Detección de código replicado (Baxter et al [3])

```
Clones=∅
Para cada s ∈ Subárboles
    Si s ≤ PesoMínimo entonces
      TablaHash = TablaHash ∪ {s}
Para cada (s1, s2) que están en la misma entrada de TablaHash
    Si Similitud(s1,s2)≥UmbralDeSimilitud entonces
      Para cada s ∈ Subárboles(s1)
            Si s ∈ Clones entonces
                  Clones=Clones - {s}
      Para cada s ∈ Subárboles(s2)
            Si s ∈ Clones entonces
                  Clones=Clones - {s}
      Clones=Clones ∪ {s1} ∪ {s2}
```

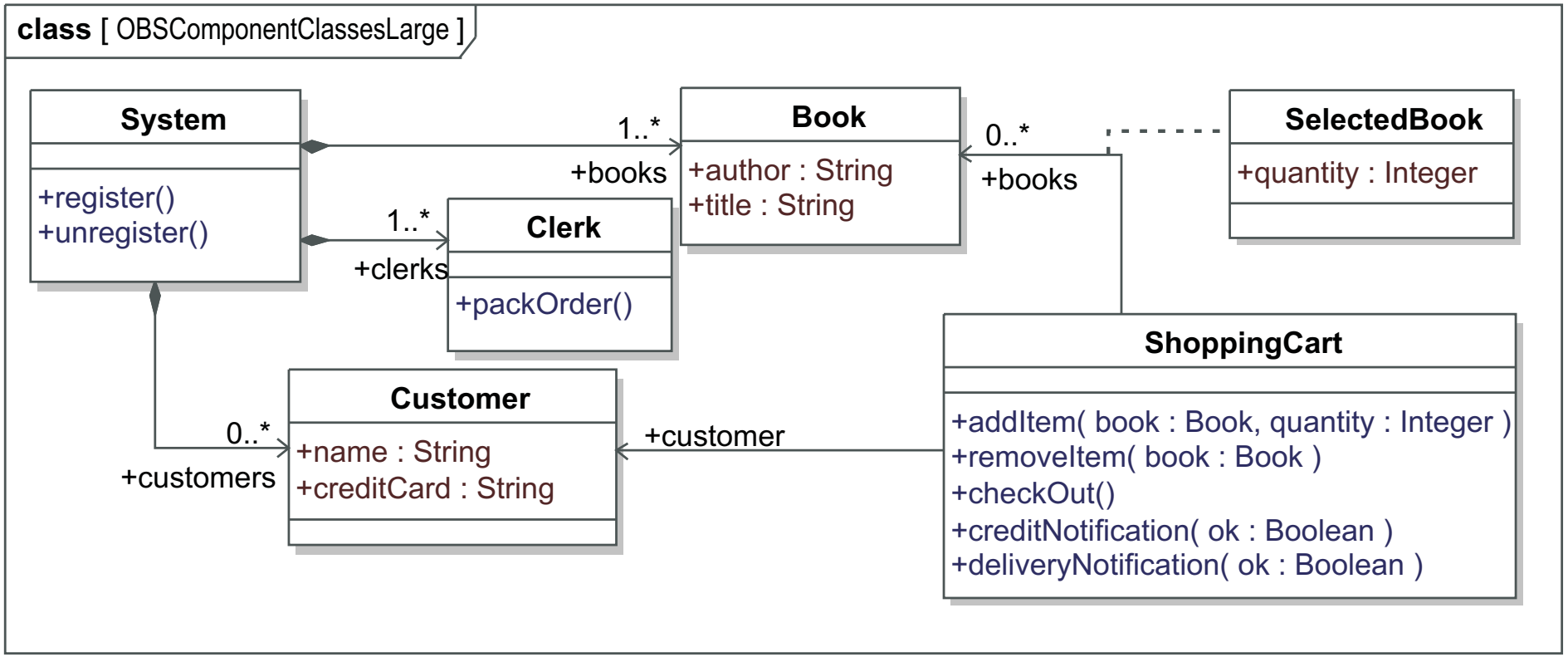
Métricas para diseños software orientados a objetos [5]

Weigthed Method per Class (WMC)	$WMC = \sum_{i=1}^{n} c_i$
Depth of Inheritance Tree (DIT)	maximum lenght to the root of the tree
Number of Children (NOC)	number of inmediate subclasses of a class
Coupling Between Objects (CBO)	number of tother classes used by a class
Response For a Class (RFC)	$RFC = M + \bigcup_{i=1}^{n} R_i$
Lack of Cohesion in Methods (LCOM)	$LCOM = P - Q$

Regla informal

Un sistema será más mantenible cuanto mayor cohesión tenga y menor sea su acoplamiento [10].

Métricas para diseños software orientados a objetos [5]



Conclusiones

1. Importancia del mantenimiento software y diferencia con sistemas clásicos.
2. Defínción de mantenimiento software. Tipos de mantenimiento y cambio.
3. Importancia y problemas de los sistemas heredados.
4. Organización y gestión del mantenimiento.
5. Concepto y técnicas de Ingeniería Inversa.
6. Concepto y técnicas de Refactorización.
7. Métricas para mantenimiento.

Referencias I

Lowell Jay Arthur.
Software Evolution: A Software Maintenance Challenge.
John Wiley & Sons, February 1988.

Victor Basili, Lionel Briand, Steven Condon, Yong-Mi Kim, Walcélio L. Melo, and Jon D. Valett.
Understanding and Predicting the Process of Software Maintenance Release.
In *Proc. of the 18th Int. Conference on Software Engineering (ICSE)*, pages 464–474, Berlin, Germany, March 1996.

Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier.
Clone Detection Using Abstract Syntax Trees.
In *Proc. of the Int. Conference on Software Maintenance (ICSM)*, pages 368–377, Bethesda (Maryland, USA), November 1998.

Referencias II

Frederick P. Brooks.
No Silver Bullet - Essence and Accidents of Software Engineering.
IEEE Computer, 20(4):10–19, April 1987.

Shyam R. Chidamber and Chris F. Kemerer.
A Metrics Suite for Object Oriented Design.
IEEE Transactions on Software Engineering, 20(6):476–493, June 1994.

Steve Counsell, Youssef Hassoun, George Loizou, and Rajaa Najjar.
Common Refactorings, a Dependency Graph and Some Code Smells: an Empirical Study of Java OSS.
In Guilherme Horta Travassos, José Carlos Maldonado, and Claes Wohlin, editors, *Proc. of the Int. Symposium on Empirical Software Engineering (ISESE)*, pages 288–296, Rio de Janeiro (Brazil), September 2006.

Referencias III

Edsger W. Dijkstra.
The humble programmer.
Communications of the ACM, 15(10):859–866, October 1972.

ISO/IEC/IEEE.
Software engineering – software life cycle processes – maintenance.
Technical Report International Standard ISO/IEC 14764:2006, IEEE Std 14764-2006, September 2006.

IEEE (The Institute of Electrical and Electronics Engineers).
Standard for software maintenance.
Technical Report IEEE Std 1219-1993, December 1992.

David L. Parnas.
On the Criteria to be Used in Decomposing Systems into Modules.
Communications of the ACM, 15(12):1053–1058, 1972.

Referencias IV


Mario Piattini, José Villalba, Francisco Ruiz, Teresa Bastanchury, Macario Polo, Miguel Angel Martínez, and César Nistal.
Mantenimiento del Software. Modelos, Técnicas y Métodos para la Gestión del Cambio.


Macario Polo, Mario Piattini, Francisco Ruiz, and Coral Calero.
MANTEMA: A Complete Rigorous Methodology for Supporting Maintenance Based On The ISO/IEC 12207 Standard.
In *Proc. of the 3rd European Conference on Software Maintenance and Reengineering (CSMR)*, pages 178–181, Amsterdam (The Netherlands), March 1999.


Stephen S. Yau and James S. Collofello.
Design Stability Measures for Software Maintenance.
IEEE Transactions on Software Engineering, 11(9):849–856, October 1985.