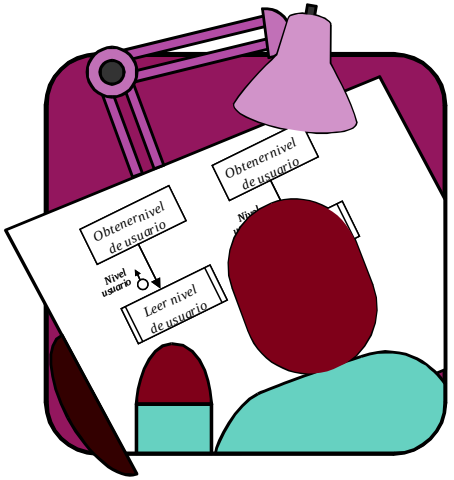
Ingeniería del Software

Tema 5: Principios del diseño del software

Dr. Francisco José García Peñalvo
(fgarcia@usal.es)
Miguel Ángel Conde González
(mconde@usal.es)
Sergio Bravo Martín
(ser@usal.es)



3º I.T.I.S.
Fecha de última modificación: 16-10-2008

Universidad de Salamanca – Departamento de Informática y Automática

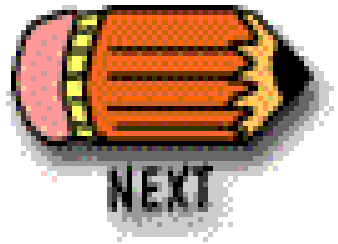
Resumen

Resumen	El diseño y la construcción del software está soportada por varios principios fundamentales. Estos principios favorecen que los objetivos de calidad del software se alcancen con mayor facilidad. En este tema se estudiarán los principios y técnicas que permiten construir arquitecturas software correctas. Primeramente se introducirá la fase de diseño y el proceso de diseño, para posteriormente centrarse en los principios y conceptos fundamentales del diseño del software, haciendo un especial hincapié en todos aquéllos que permitan alcanzar un diseño modular eficaz, basado en módulos altamente cohesionados, con bajo acoplamiento y construidos sobre la base de la ocultación de la información
Descriptores	Diseño de software; Diseño arquitectónico; Diseño de datos; Diseño de procedimientos; Diseño de la interfaz; Abstracción; Refinamiento sucesivo; Modularidad; Arquitectura del software; Jerarquía de control; Partición estructural; Estructura de datos; Procedimiento del software; Ocultación de la información; Diseño modular; Módulo; Independencia modular; Cohesión; Acoplamiento
Bibliografía	[Meyer, 1999] Capítulo 3 [Piattini et al., 2004] Capítulo 8 [Pressman, 2006] Capítulos 9, 10, 11 y 12 [Sommerville, 2005] Capítulos 11 y 16

Esquema

- Introducción
- Proceso de diseño del software
- Principios y conceptos del diseño del software
- Aportaciones principales del tema
- Ejercicios
- Lecturas complementarias
- Referencias

1. Introducción

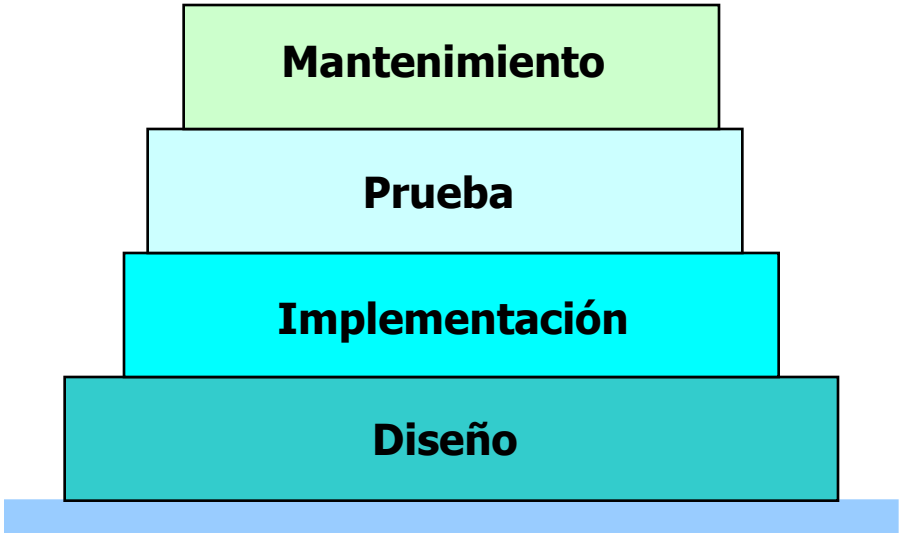


Concepto de diseño

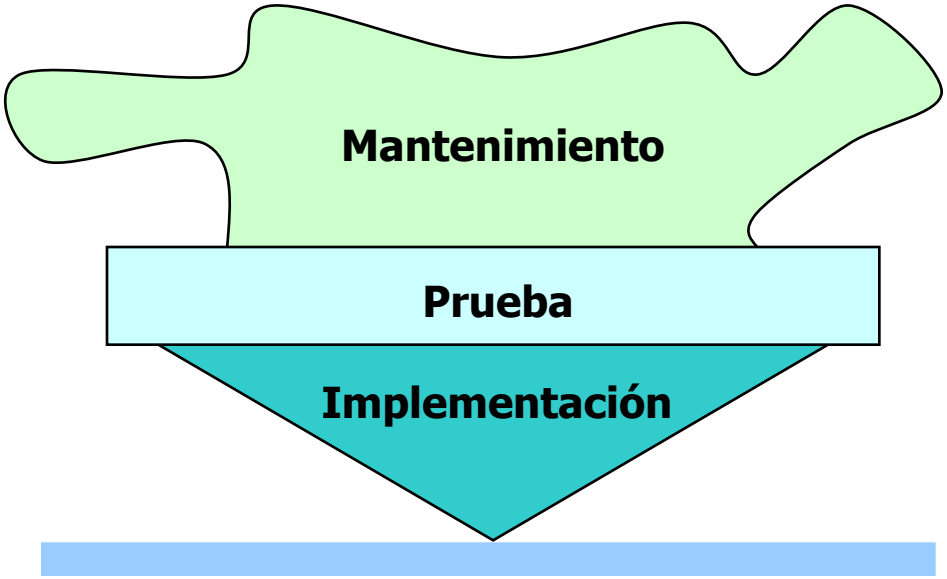
> Proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física [Taylor, 1959]

- Proceso común en la actividad humana
- Intuitivamente es el proceso que se trata de formular y evaluar una solución para un problema dado
- En el caso del diseño de un sistema software será la búsqueda de soluciones que se ajusten a los requisitos del usuario
- Actividad necesaria para conseguir un software bien acabado

Importancia del diseño en el ciclo de vida de un producto



Con diseño



Sin diseño

Diseño como actividad creativa

- El diseño combina
  - Creatividad
  - Intuición
  - Experiencia

  Del ingeniero del software
  - Guías
  - Métodos
  - Heurísticas

  Del proceso de diseño
  - Criterios de calidad
  - Proceso iterativo

  Diseño Final

Definición de diseño de software (i)

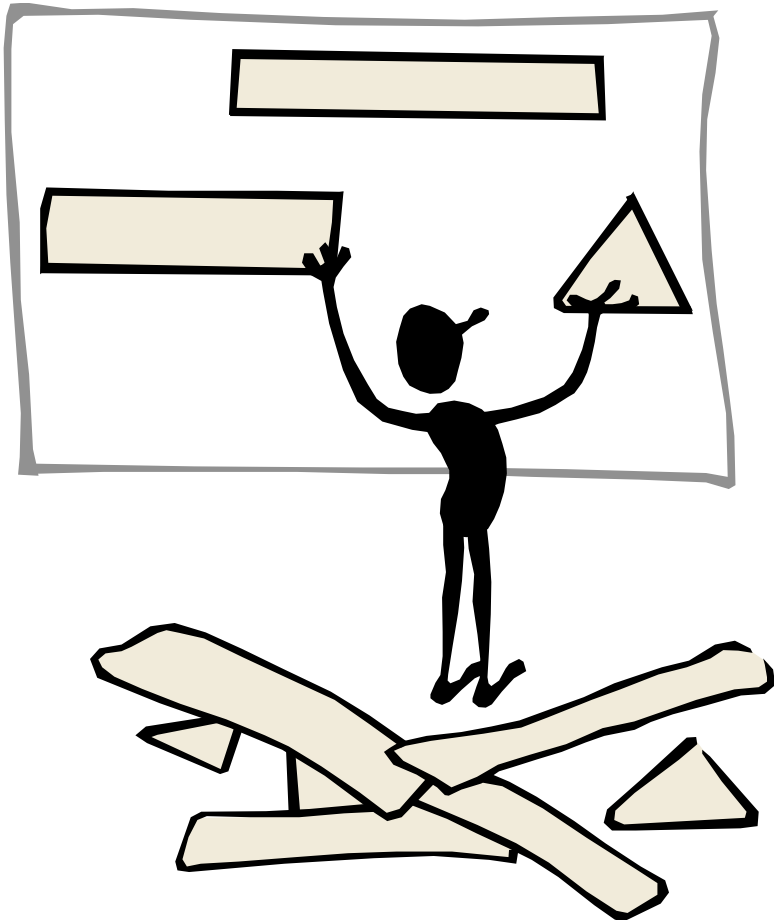
- Es el proceso de definición de la arquitectura software: componentes módulos, interfaces, procedimientos de prueba y datos de un sistema que se crean para satisfacer unos requisitos especificados [AECC, 1986]
- En un sentido, el diseño es la representación de un objeto que está siendo creado. Un diseño es una información de base que describe aspectos de este objeto, y el proceso de diseño puede ser visto como una elaboración sucesiva de representaciones, tales como añadir más información, puntos de retorno y explorar alternativas [Webster, 1988]
- Es la práctica de tomar una especificación del comportamiento observable externamente y añadir los detalles necesarios para la implementación actual del sistema computacional, incluyendo detalles sobre la interacción de los usuarios, la gestión de tareas y la gestión de datos [Coad y Yourdon, 1991]

Definición de diseño de software (ii)

- Es un proceso de invención y selección de programas que cumplan los objetivos de un sistema software. La entrada incluye el entendimiento de los requisitos, las restricciones de entorno y los criterios de diseño. La salida del proceso de diseño está compuesta de una arquitectura de diseño que muestra como las piezas están interrelacionadas, de especificaciones de cualquier pieza nueva y de las definiciones de cualquier dato nuevo [Stevens, 1991]
- El diseño de software es el proceso de definir la arquitectura, componentes, interfaces y otras características de un sistema o componente; el resultado de ese proceso IEEE-Std. 610.12 [IEEE, 1999]
- El diseño del software es una descripción de la estructura del software que se va a implementar, los datos que son parte del sistema, las interfaces entre los componentes del sistema y, algunas veces, los algoritmos utilizados [Sommerville, 2005]

Evolución del diseño de software

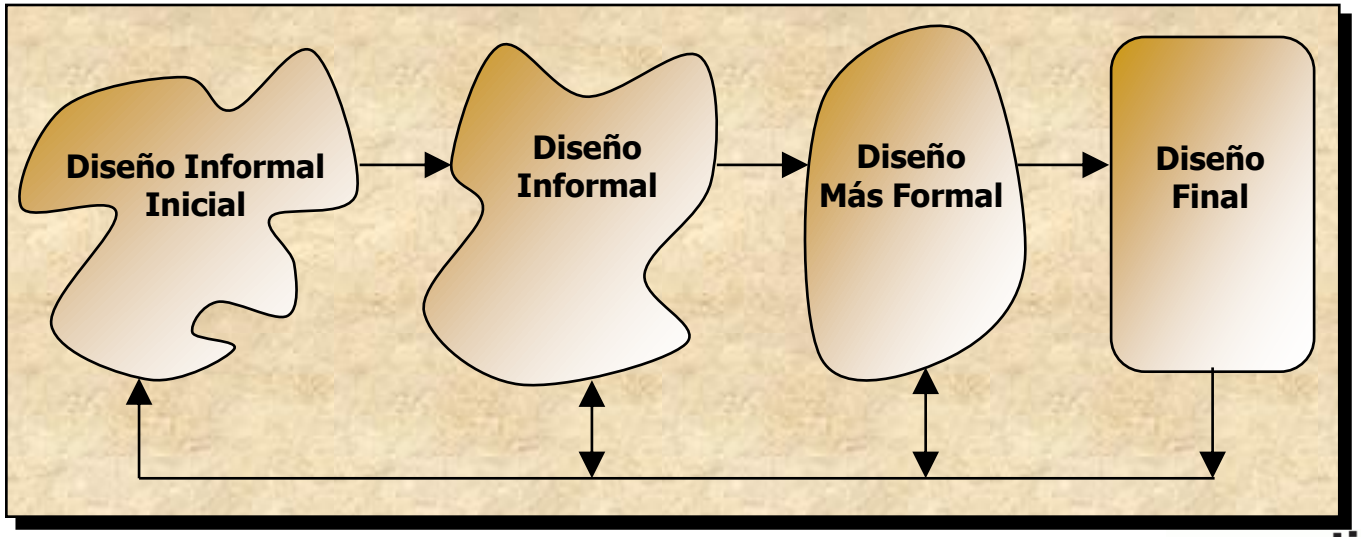
- El diseño de software disciplina que evoluciona
- Primeros años de la década de los 70s
  - Programación modular [Dennis, 1973]
  - Refinamiento descendente [Wirth, 1971]
  - Evolución hacia la programación estructurada [Dahl et al., 1972]
- Mediados de los 70s
  - Transformaciones de los flujos de datos [Stevens et al., 1974]
  - Transformaciones de la estructura de datos [Warnier, 1974], [Jackson, 1975]
- Finales de los 80s, década de los 90s
  - Diseño Orientado a Objeto (DOO) [Wirfs-Brock et al., 1990], [Gamma et al., 1995], [Buschmann et al., 1996]
- Las diferentes tendencias en diseño han dado lugar a métodos de diseño



2. Proceso de diseño del software

El proceso de diseño

- El diseño es un proceso de resolución de problemas cuyo objetivo es encontrar y describir una forma
  - Para implementar los **requisitos funcionales** del sistema
  - Respetando las restricciones impuestas por los **requisitos no funcionales**
    - Incluyendo las presupuestarias
  - Ajustándose a los principios generales de **calidad**
- El proceso de diseño es, por tanto, un proceso iterativo, mediante el cual se va a realizar una traducción de los requisitos en una representación del software



Diseño como toma de decisiones

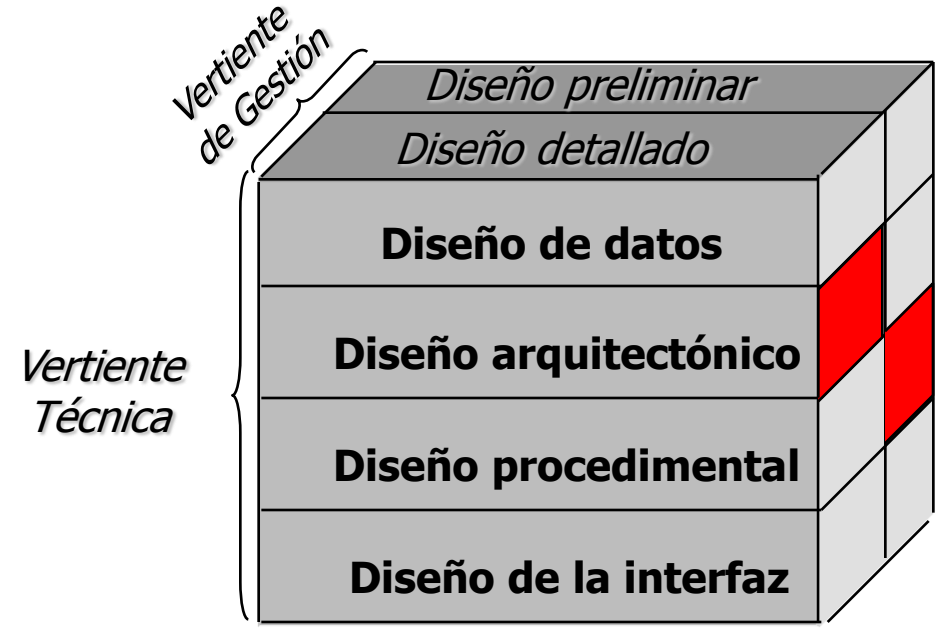
- El diseñador se enfrenta a una colección de problemas de diseño
  - Cada problema tiene normalmente varias soluciones alternativas
    - Opciones de diseño
  - El diseñador toma una decisión de diseño para resolver cada problema
    - Este proceso implica elegir la mejor opción entre las alternativas
- Para tomar las decisiones de diseño el ingeniero software utiliza el conocimiento que tiene de
  - Los requisitos
  - El diseño realizado hasta el momento
  - La tecnología disponible
  - Los principios de diseño y de las "buenas prácticas"
  - Lo que ha funcionado bien en situaciones anteriores

Objetivos de la fase de diseño

- Descomponer el sistema en subsistemas
  - Identificar la **arquitectura software**
- Determinar las relaciones entre componentes
  - Identificar las dependencias entre componentes y determinar los mecanismos de comunicación entre componentes
- Especificar las interfaces entre los componentes
  - Interfaces bien definidas para facilitar la prueba y comunicación entre los componentes
- Describir la funcionalidad de los componentes

Actividades del proceso de diseño (i)

- Según Roger S. Pressman (1992)
  - Diseño preliminar
    - También llamado *diseño de sistema, diseño arquitectónico* o *diseño de alto nivel*
    - Identificar los módulos en los que puede dividirse atendiendo a motivos de conveniencia de implementación
  - Diseño detallado
    - Se centra en la lógica interna de dichos módulos
    - Se ocupa del refinamiento de la representación arquitectónica que lleva a una estructura de datos detallada y a las representaciones algorítmicas del software
  - Hay una vertiente técnica y una vertiente de gestión en el diseño



Actividades del proceso de diseño (ii)

- **Diseño arquitectónico**
  - Define la relación entre los elementos estructurales principales del software, los patrones de diseño que se pueden utilizar para lograr los requisitos que se han definido para el sistema, y las restricciones que afectan a la manera en que se pueden aplicar los patrones de diseño arquitectónicos [Shaw y Garlan, 1996]
- **Diseño de datos**
  - Transforma el modelo del dominio de información creado en el análisis en las estructuras de datos necesarias para la implementación del software [Pressman, 2006]
  - Influencia de la estructura de datos en la estructura del programa y en la complejidad de los procedimientos
  - Ocultación de la información y Abstracción
  - Datos bien diseñados conducen a
    - Mejor estructura del programa
    - Modularidad efectiva
    - Reducción de la complejidad procedimental

Actividades del proceso de diseño (iii)

- Principios de Wasserman (1996) para el diseño de datos
  - Los principios sistemáticos del análisis aplicados a la función y al comportamiento también deben aplicarse a los datos
  - Deben identificarse todas las estructuras de datos y las operaciones que se han de realizar sobre cada una de ellas
  - Debe establecerse y usarse un diccionario de datos para definir el diseño de los datos y del programa
  - Deben posponerse las decisiones de datos de bajo nivel hasta el diseño detallado
  - La representación de una estructura de datos sólo debe ser conocida por los módulos que hagan uso directo de los datos contenidos en la estructura
  - Se debe desarrollar una biblioteca de estructuras de datos útiles y de las operaciones que se les pueden aplicar
  - El diseño del software y el lenguaje de programación deben soportar la especificación y la realización de tipos abstractos de datos

Actividades del proceso de diseño (iv)

- **Diseño a nivel de componentes (diseño procedimental)**
  - Transforma los elementos estructurales de la arquitectura del software en una descripción procedimental de los componentes del software
  - Diseño de algoritmos
- **Diseño de la interfaz**
  - Diseño de interfaces hombre-máquina para facilitar al usuario la utilización del sistema
  - Propósito
    - **Recoger de los usuarios la información del sistema y ponerla a disposición de otros usuarios**
      - La interfaz cubre las entradas y las salidas
  - Se debe tener en cuenta la psicología del usuario
    - Sobrecarga de la información
    - Complejidad de la tarea
    - Grado de control del sistema permitido al usuario
  - Ergonomía
    - Estudio de datos biológicos y tecnológicos aplicados a problemas de mutua adaptación entre el hombre y la máquina [RAE, 2001]

Actividades del proceso de diseño (v)

- Por su parte D. E. Webster (1988) y L. A. Belady (1990) hablan de
  - Diseño de flujo ascendente (*Upstream design*)
    - Es adaptable y abstracto y tiende a corresponderse con las fases de análisis de requisitos y de diseño preliminar
  - Diseño de flujo descendente (*Downstream design*)
    - Se refiere a los módulos, codificación y documentación, correspondiéndose con las fases de diseño detallado e implementación

3. Principios y conceptos del diseño del software

Introducción

> "El comienzo de la sabiduría de un programador de computadoras está en reconocer la diferencia entre obtener un programa que funcione y obtener uno que funcione correctamente"
>
> M. A. Jackson (1975)

- ¿Qué criterios se pueden usar para partir el software en componentes individuales?
- ¿Cómo se separan los detalles de una función o de la estructura de los datos de la representación conceptual del software?
- ¿Existen criterios uniformes que definen la calidad técnica de un diseño de programas?

Principios del diseño del software

- Los principios básicos de diseño hacen posible que el ingeniero del software navegue por el proceso de diseño [Pressman, 2002]
  - En el proceso de diseño no deberá utilizarse "orejeras"
  - El diseño deberá poderse rastrear hasta el modelo de análisis
  - El diseño no deberá inventar nada que ya esté inventado
  - El diseño deberá minimizar la distancia intelectual entre el software y el problema, como si de misma vida real se tratara
  - El diseño deberá presentar uniformidad e integración
  - El diseño deberá estructurarse para admitir cambios
  - El diseño deberá estructurarse para degradarse poco a poco, incluso cuando se enfrenta con datos, sucesos o condiciones de operación aberrantes
  - El diseño no es escribir código y escribir código no es diseñar
  - El diseño deberá evaluarse en función de la calidad mientras que se va creando, no después de terminarlo
  - El diseño deberá revisarse para minimizar los errores conceptuales (semánticos)

Conceptos del diseño

- Los conceptos fundamentales del diseño del software
  - Proporcionan el marco de trabajo necesario para conseguir que se haga correctamente
  - Favorecen la gestión de la complejidad de los sistemas software y la consecución de los factores de calidad que estos sistemas han de exhibir
- Los conceptos del diseño a tratar son [Pressman, 2002]
  - Abstracción
  - Refinamiento sucesivo (descomposición)
  - Ocultación de la información
  - Modularidad
  - Arquitectura del software
  - Jerarquía de control
  - División estructural
  - Estructura de datos
  - Procedimiento de software

Abstracción (i)

- Definición
  - Separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción [RAE, 2001]
  - La representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes [Graham, 1994]
  - La noción psicológica de abstracción permite concentrarse en un problema a un nivel de generalización independiente de los detalles de nivel inferior; la abstracción también permite trabajar con conceptos y términos que son familiares en el entorno del problema sin tener que transformarlos en una estructura poco familiar [Wasserman, 1983]

Abstracción (ii)

- Los diseños han de ocultar o diferir los detalles de implementación
- Las abstracciones permiten comprender la esencia de los subsistemas sin tener que conocer detalles innecesarios
- Las decisiones de diseño susceptibles de cambio deben ocultarse detrás de interfaces abstractas
- Los módulos se han de diseñar de forma que la información interna del módulo sea inaccesible a otros módulos que no la necesitan
  - Una solución modular implica niveles de abstracción
- Ventajas
  - Define y refuerza las restricciones de acceso
  - Facilita el mantenimiento y la evolución de los sistemas software
  - Reduce los efectos laterales
  - Limita el impacto global de las decisiones de diseño locales
  - Favorece la **encapsulación**, uno de los elementos de un buen diseño

Abstracción (iii)

- Descripción de una función de un programa en el nivel de detalle adecuado
- Cada paso en el proceso del software es un refinamiento del nivel de abstracción de la solución software (abstracciones funcionales y abstracciones de datos)
- Refinamiento y modularidad son conceptos cercanos al concepto de abstracción
- Abstracción de datos
  - Colección identificada de datos que describe un objeto de datos
- Abstracción procedimental
  - Secuencia identificada de instrucciones que tiene una función especificada y limitada
- Abstracción de control
  - Mecanismo de control de programa sin especificar los datos internos

Abstracción (iv)

- Principios
  - Especialización
    - Forma común de la herencia, donde el objeto derivado tiene propiedades más precisas que el objeto base
  - Descomposición
    - Principio de separación de una abstracción en sus componentes
  - Instanciación
    - Es el proceso de crear instancias de una clase
  - Individualización
    - El agrupamiento de objetos similares para propósitos comunes
    - Un objeto de una clase dada tiene un propósito o rol dado

Refinamiento sucesivo (i)

- Estrategia de diseño descendente
- El diseño se refina con una jerarquía de detalles creciente
- Concepto muy ligado a la abstracción
  - El refinamiento es un concepto complementario a la abstracción
- Es el procedimiento por el que se va pasando de los niveles superiores de abstracción a los niveles inferiores, es decir, la manera en que se va añadiendo información de un nivel a otro [Wirth, 1971]
  - En cada paso, una o varias instrucciones del programa dado se descomponen en instrucciones más detalladas
  - Refinamiento paralelo de funciones y datos
  - Cada refinamiento implica decisiones de diseño

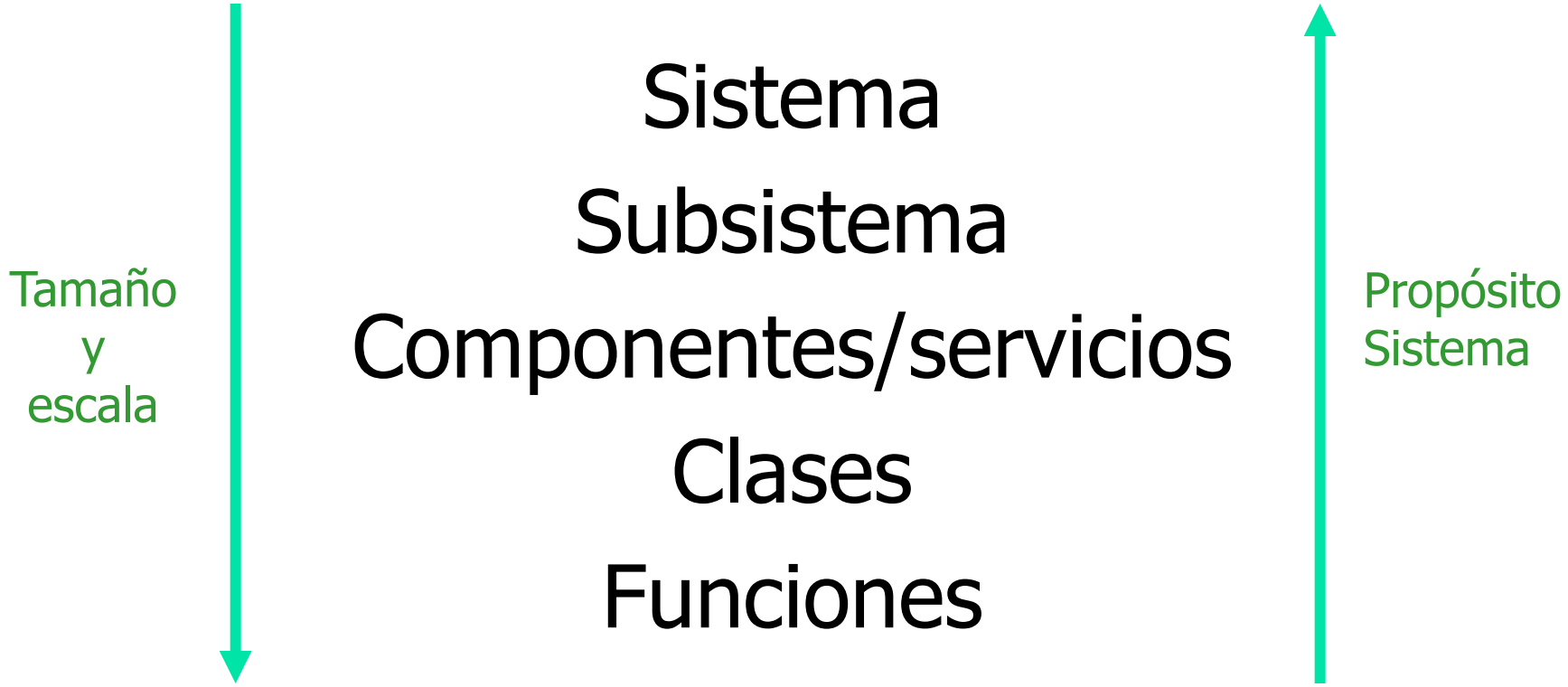
Refinamiento sucesivo (ii)

- Similitud con el procedimiento de partición y refinamiento del análisis de requisitos
  - La diferencia está en el nivel de detalle, no en el enfoque
- El refinamiento es un procedimiento de elaboración
  - Función descrita a un nivel conceptual
  - Refinamientos sucesivos que incorporan más detalles
- Gestiona la complejidad dividiendo problemas grandes en problemas pequeños
  - Permite que personas diferentes puedan trabajar con cada parte
  - Permite la especialización de los ingenieros del software
  - Cada componente individual es más pequeño y, por tanto, su comprensión es más sencilla
  - Las partes se pueden remplazar o cambiar sin tener que remplazar o cambiar de forma generalizada el resto de las partes

Refinamiento sucesivo (iii)

- Proceso
  1. Seleccionar un problema (inicialmente el sistema completo)
  2. Determinar los componentes de este problema o parte del problema mediante un paradigma de diseño
     1. Funcional, estructurado, orientado al objeto...
  3. Describir las interacciones entre los componentes
  4. Repetir los pasos 1-3 hasta que se verifique el criterio de finalización
- Objetivos
  - Cada pieza en el mismo nivel de detalle
  - Cada pieza resoluble independientemente
  - Se pueden combinar piezas para resolver el problema original

Refinamiento sucesivo (iv)



Ocultación de la información (i)

- Los módulos se caracterizan por las decisiones de diseño que (cada uno) oculta al resto [Parnas, 1972]
  - Los módulos deberán especificarse y diseñarse de manera que la información (procedimientos y datos) que está dentro de un módulo sea inaccesible a otros módulos que no necesiten esa información
- El ocultamiento de la información es un buen medio para conseguir abstracción
- Restricciones de acceso
  - Detalle procedimental dentro del módulo
  - Estructura de datos local empleada por el módulo
- El diseñador de cada módulo debe seleccionar un subconjunto de las propiedades del módulo como información oficial del módulo, para ponerlas a disposición de los autores de módulos o de módulos clientes

Ocultación de la información (ii)

- Las decisiones de diseño sujetas a cambio deben ocultarse detrás de interfaces abstractas
  - Las aplicaciones software han de comunicarse únicamente a través de interfaces bien definidas
  - La interfaz de un módulo debe revelar lo menos posible de su funcionamiento interno
  - Intercambio de módulos mientras que se mantengan intactas las interfaces
  - Ventajas a la hora de hacer cambios
- Ocultación significa que se puede conseguir una modularidad efectiva definiendo un conjunto de módulos independientes que se comunican intercambiando la información necesaria para realizar la función software

Modularidad (i)

- Es una partición lógica del diseño software que permite al software complejo ser manejable para propósitos de implementación y mantenimiento
- Es el atributo individual del software que permite a un programa ser intelectualmente manejable [Myers, 1978]
- Es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados [Booch, 1994]
- Propuesta de descomposición funcional del sistema
  - Sistema dividido en subsistemas
  - Subsistemas son repetidamente divididos hasta que son intelectual y técnicamente manejables como unidades individuales

Modularidad (ii)

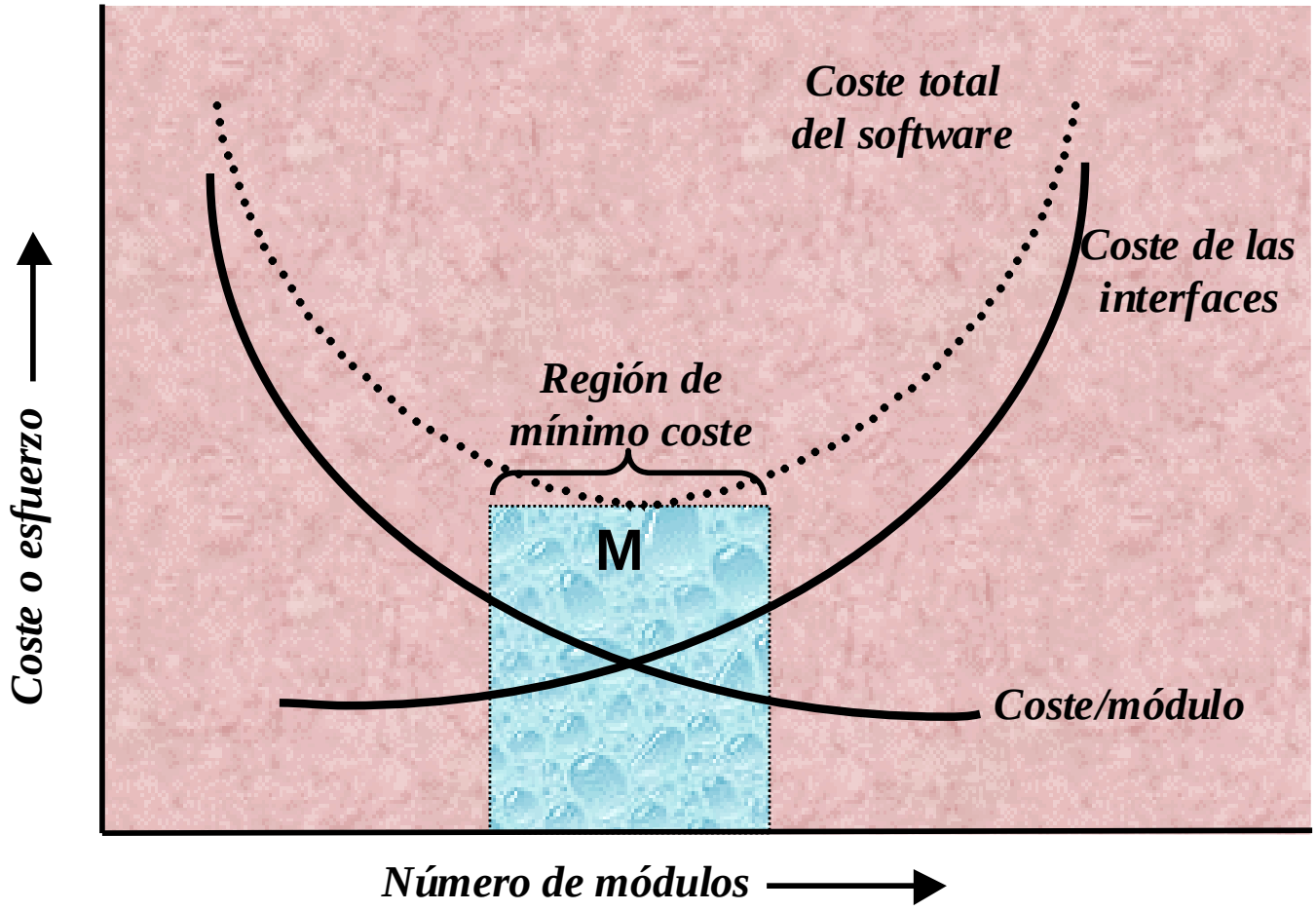
- La modularidad facilita
  - Los factores de calidad del software
    - Extensibilidad: interfaces abstractas, bien definidas
    - Reusabilidad: bajo acoplamiento, alta cohesión
    - Portabilidad: oculta las dependencias máquina
  - La calidad de los diseños software
    - Mejora la separación de aspectos
    - Permite reducir la complejidad del sistema global mediante arquitecturas software descentralizadas
    - Incrementa la escalabilidad mediante el soporte al desarrollo independiente y concurrente por varias personas

Modularidad (iii)

- Un módulo es cualquier parte o subsistema de un sistema mayor
  - Componente bien definido de un sistema software
  - Autónomo, auto-contenido, cohesión lógica
- Se distingue entre
  - Módulos internos: se descomponen en otros
  - Módulos hojas: no se descomponen más
- Un sistema es modular si está compuesto de módulos bien definidos, conceptualmente simples e independientes, que interactúan a través de interfaces bien definidas

Modularidad (iv)

- Modularidad y costes de software [Pressman, 2006]



Modularidad (v)

- Los sistemas modulares son deseables por las siguientes razones
  - Son fáciles de entender y explicar porque se puede estudiar cada módulo por separado, y además estos módulos tienen bien definidas las interdependencias
  - Al ser más fáciles de entender y explicar, son fáciles de documentar
  - Son más fáciles de programar porque grupos independientes pueden trabajar en módulos diferentes con poca comunicación
  - Son más fáciles de probar porque pueden ser probados por separado, y después integrados y probados juntos
  - Cuando los módulos son realmente independientes, son más fáciles de mantener. Se pueden hacer cambios en algunos módulos sin afectar al resto del sistema
- La modularidad mejora la calidad del diseño
  - Reduce la complejidad
  - Aumenta la claridad
  - Facilita implementación, depuración, pruebas, documentación y mantenimiento del software

Modularidad (vi)

- Criterios de evaluación de la modularidad [Meyer, 1997]
  - Capacidad de la descomposición modular
  - Capacidad de empleo de componentes modulares
  - Capacidad de comprensión modular
  - Continuidad modular
  - Protección modular
- Reglas de evaluación de la modularidad [Meyer, 1997]
  - Correspondencia directa
  - Pocas interfaces
  - Interfaces pequeñas
  - Interfaces explícitas
  - Ocultación de la información

Modularidad (vii)

- Descomposición
  - ¿Los componentes grandes se pueden descomponer en componentes pequeños?
  - Un método de construcción de software satisface el principio de capacidad de descomposición modular si ayuda en la tarea de descomponer un problema software en un número pequeño de subproblemas menos complejos, conectados por una estructura simple y lo suficientemente independiente para permitir trabajar de forma separada en cada uno de ellos
    - Ejemplo: Diseño descendente
    - Contraejemplo: Módulo de inicialización que lo inicializa todo para todos
- Composición
  - ¿Los componentes grandes se componen de componentes pequeños?
  - Un método satisface la capacidad de empleo de componentes modulares si favorece la producción de elementos software que pueden ser libremente combinados con el resto para producir nuevos sistemas, posiblemente en un entorno algo diferente del cual fueron inicialmente desarrollados
    - En diferentes entornos -> componentes
    - Ejemplo: Bibliotecas matemáticas; Órdenes y tuberías UNIX
    - Contraejemplo: Preprocesadores (problemas de compatibilidad)

Modularidad (viii)

- Comprensión
  - ¿Los componentes son comprensibles individualmente?
  - Un método favorece la capacidad de comprensión modular si ayuda a producir software, en el cual un lector humano pueda entender cada módulo sin tener que conocer el resto de los módulos, o, en el peor de los casos, teniendo que examinar sólo un conjunto reducido del resto de los módulos
    - Importancia para el mantenimiento
    - Independencia del nivel de abstracción del módulo
    - La información relevante de un módulo debe aparecer en el texto del módulo
  - Contraejemplo: Dependencias secuenciales (A|B|C)

Modularidad (ix)

- Continuidad
  - ¿Pequeños cambios en la especificación afectan a un número limitado y localizado de componentes?
  - Los resultados de un pequeño cambio en la especificación
    - Solamente cambia un número pequeño de módulos
    - No afecta a la arquitectura
  - Unido a la extensibilidad del software
    - Ejemplos: Constante simbólicas; Principio uniforme de acceso
    - Contraejemplo: Arrays estáticos
- Protección
  - ¿Están los efectos de las anomalías de ejecución confinados a un número pequeño de componentes relacionados?
  - Se cumple el principio de protección modular cuando si se da una condición anormal en tiempo de ejecución dentro de un módulo, permanecerá confinada dentro de dicho módulo, o en el peor de los casos se propagará sólo a unos pocos módulos cercanos
    - Ejemplo: Validación de las entradas en la fuente
    - Contraejemplo: Excepciones no disciplinadas

Modularidad (x)

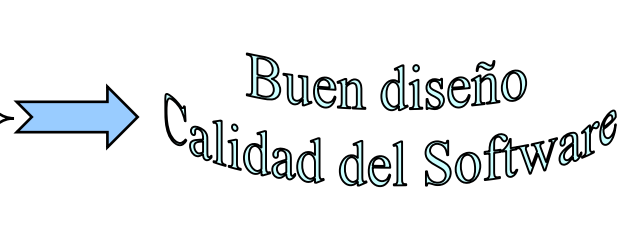
- Correspondencia directa (*direct mapping*)
  - Debe existir una relación consistente entre el modelo del problema y la estructura de la solución
  - Mantener la estructura de la solución compatible con la estructura del dominio del problema modelado
  - Afecta a
    - **Continuidad**: Facilita y limita el impacto del cambio
    - **Descomposición**: La descomposición del dominio del problema es un buen punto de partida para la descomposición del software
- Interfaces pequeñas
  - Si dos componentes se comunican, deben intercambiar la menor información posible
  - Se refiere al tamaño de las conexiones entre módulos más que a su número
  - Afecta a la **Continuidad** y la **Protección**

Modularidad (xi)

- Interfaces explícitas
  - Cuando dos componentes se comunican, esto debe ser evidente en la especificación de al menos uno de ellos
  - Afecta a la **Descomposición**, **Composición**, **Continuidad**, **Comprensión**
- Pocas interfaces
  - Cada componente debe comunicarse con el menor número posible de componentes
  - Restringe el número de canales de comunicación entre los módulos de una arquitectura software
  - Afecta a la **Continuidad**, **Protección**, **Composición**, **Comprensión**
- Ocultación de la Información
  - Cada módulo debe seleccionar un subconjunto de las propiedades del módulo como información oficial del módulo, para ponerlas a disposición de los autores de módulos o de módulos clientes
  - Afecta a **Continuidad**, **Descomposición**, **Composición**, **Comprensión**

Modularidad (xii)

- Diseño modular efectivo
  - Se logra desarrollando módulos con una función claramente definida y evitando una excesiva interacción con otros módulos
  - Suma de la modularidad y los conceptos de abstracción y ocultación de la información [Presman, 2002]
  - Ventajas
    - Módulos más fáciles de desarrollar
    - Módulos más fáciles de mantener y probar
    - Facilidad para su reutilización



  - La independencia se mide mediante dos criterios cualitativos
    - Cohesión
    - Acoplamiento

Modularidad – Cohesión (i)

- La cohesión es la medida de la relación funcional de los elementos de un módulo. Aquí un elemento hace referencia a cualquier componente del módulo, instrucción o grupo de instrucciones, definiciones de datos, procedimientos… que lleva a cabo algún trabajo o define algún dato
- Estudia la medida de la relación que existe entre los elementos de un mismo módulo
- Organización de los elementos de forma que los que tengan una mayor relación para realizar una tarea, pertenezcan al mismo módulo y los elementos no relacionados se encuentren en módulos separados
  - Un subsistema o módulo tiene un alto grado de cohesión si mantiene "unidas" cosas que están relacionadas entre ellas y mantiene fuera el resto
- Favorece la comprensión y el cambio de los sistemas
- Fortaleza de la "asociatividad" funcional de las actividades
- Un módulo cohesivo lleva a cabo una sola tarea dentro de un procedimiento software

Modularidad – Cohesión (ii)

1. Funcional
2. Secuencial
3. Comunicación
4. Procedural
5. Temporal
6. Lógica
7. Coincidencia

Buena

Mala

Caja negra

Caja gris

Caja transparente

Modularidad – Cohesión (iii)

- Cohesión funcional
  - Todos los elementos que componen el módulo están relacionados en el desarrollo de una función única y perfectamente definida
  - Devuelve un resultado sin efectos colaterales
  - Fácil comprensión, reutilizable, fácil reemplazamiento
  - Los elementos contribuyen a la ejecución de una tarea relacionada con el problema
  - Grado más alto de cohesión
- Cohesión secuencial
  - Un módulo representa el empaquetamiento físico de varios módulos con cohesión funcional, donde cada uno proporciona la entrada al siguiente
  - Un servicio secuencialmente cohesivo reduce el acoplamiento encapsulando funcionalidad relacionada
  - Varios módulos con cohesión funcional trabajan sobre la misma estructura de datos, pero han de existir tantos puntos de entrada como número de funciones realice dicho módulo
  - Menor grado de cohesión que la cohesión funcional

Modularidad – Cohesión (iv)

- Cohesión de comunicación
  - Todos los módulos que acceden o manipulan ciertos datos se mantienen unidos y el resto se excluye
  - Los elementos contribuyen en actividades que utilizan los mismos datos de entrada o de salida
    - Dado un ISBN como entrada a un servicio: *encontrar el título del libro*, *encontrar el precio del libro*, encontrar la editorial del libro
  - No importa el orden en la ejecución de las actividades
  - **Evitar** la tentación de incluir todas las actividades definidas en un servicio
  - Si se localiza un conjunto de actividades no secuenciales actuando sobre los mismos datos es un indicador de cohesión de comunicación

Modularidad – Cohesión (v)

- Cohesión procedural
  - Composición de partes de funcionalidad que se organizan secuencialmente pero que, por otra parte, tienen poca relación entre sí
  - Mal mantenimiento
- Cohesión temporal
  - Un módulo que posee cohesión temporal es aquél cuyos elementos están implicados en actividades que están relacionadas con el tiempo
  - Las operaciones que se realizan durante la misma fase de la ejecución del programa se mantienen unidas
  - Organiza actividades relacionadas de forma secuencial por tiempo
  - Mal mantenimiento

Modularidad – Cohesión (vi)

- Cohesión lógica
  - Agrupa una serie de actividades en la misma categoría general
    - Cuando existe alguna relación entre los elementos del módulo, aunque sea débil
  - En algunos casos puede dar lugar a confusiones por no estar bien definidas las fronteras entre los diferentes elementos del módulo
  - La actividad a ejecutar se determina normalmente por un parámetro de entrada
- Cohesión por coincidencia
  - Se relacionan utilidades que no se pueden ser situadas de forma lógica en otras unidades cohesivas
  - Cuando entre los elementos que componen el módulo no existe ninguna relación con sentido
    - Procede de un programa que ya existe y que simplemente se ha dividido en módulos
    - Se crean módulos con grupos de sentencias que se repiten en mismo módulo o en módulos diferentes. Es un intento de evitar la duplicidad de código
    - Un programa ya creado se fragmenta por alguna causa, como puede ser la limitación de memoria

Modularidad – Acoplamiento (i)

- El acoplamiento es una medida de la interconexión entre los módulos de una estructura de software [Pressman, 2006]
- El acoplamiento depende de la complejidad de la interconexión entre los módulos, el punto donde se realiza una entrada o referencia a un módulo y los datos que se pasan a través de la interfaz
- Minimizar el acoplamiento implica un buen diseño
  - Un acoplamiento bajo indica un sistema bien dividido y puede conseguirse mediante la eliminación o reducción de relaciones innecesarias
- Acoplamiento mínimo
  - Ningún módulo tiene que preocuparse de los detalles de la construcción interna del resto de los módulos

Modularidad – Acoplamiento (ii)

- Se produce una situación de acoplamiento cuando un elemento de un diseño depende de alguna forma de otro elemento del diseño
- El **objetivo** es conseguir un acoplamiento lo más bajo posible
  - Conseguir que cada componente sea tan independiente como sea posible
- Una conectividad sencilla entre los módulos da como resultado
  - Un software más fácil de entender
  - Un software menos propenso al efecto "ola"
    - El causado cuando los errores que se producen en un lugar del sistema se propagan por el resto del sistema
- Un bajo acoplamiento significa que un cambio en un componente no implicará un cambio en otro
- Cuando el nivel de acoplamiento es bajo, el diseño, por regla general, se mantiene y extiende mejor

Modularidad – Acoplamiento (iii)

- El acoplamiento depende de varios factores
  - Referencias hechas de un componente a otro
    - El componente **A** puede invocar al componente **B**; por lo tanto **A** depende de **B** para completar su función o proceso
  - Cantidad de datos pasados de un componente a otro
    - El componente **A** puede pasarle a **B** un parámetro, el contenido de un vector secuencial o un bloque de datos
  - El grado de control que un componente tiene sobre el otro
    - El componente **A** puede pasar una señal de control (*flag*) a un componente **B**. El valor de la señal de control indica al componente **B** el estado de un recurso o subsistema, qué proceso debe invocar o si debe o no invocar alguno
  - El grado de complejidad de la interfaz entre los componentes
    - **A** pasa un parámetro a **B** y éste puede ejecutarse
    - **C** y **D** intercambian valores antes de que **D** pueda completar su ejecución

Modularidad – Acoplamiento (iv)

- En general, los módulos están **fuertemente acoplados** si utilizan variables compartidas o intercambian información de control
- Acoplamiento débil
  - Garantizar que los detalles de la representación de datos se mantiene dentro de un componente
  - Interfaz con otros componentes mediante lista de parámetros
  - Si es necesario que exista información compartida se debe limitar a aquellos componentes que necesitan acceder a la información
  - Se debe evitar información compartida de forma global

Modularidad – Acoplamiento (v)

Mejor

1. Normal
2. De datos
3. Por estampado
4. De control
5. Externo
6. Común
7. De contenido

Peor

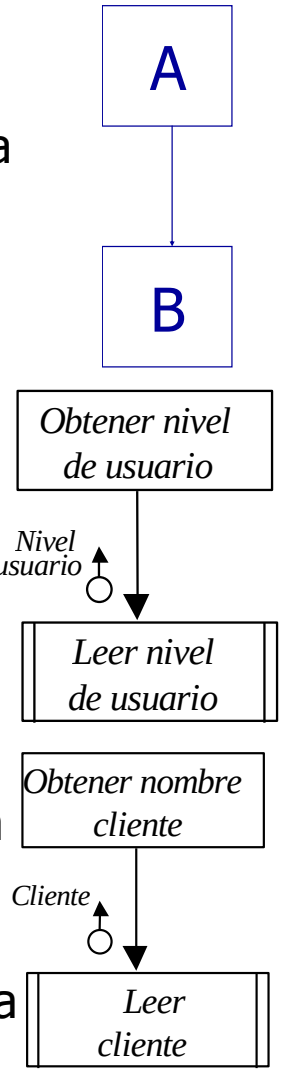
Bajo

Débil

Alto

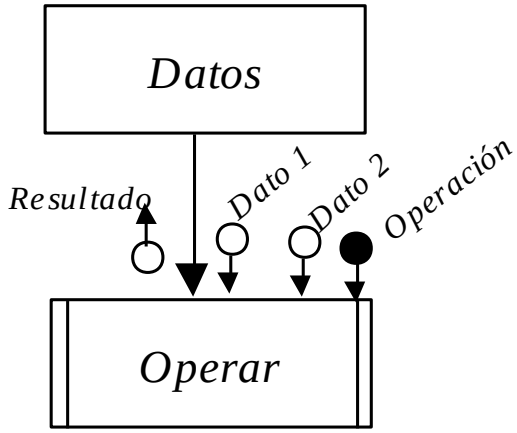
Modularidad – Acoplamiento (vi)

- Niveles bajos de acoplamiento
  - **Acoplamiento normal**
    - Dos módulos **A** y **B** están normalmente acoplados sí: un módulo **A** llama a otro **B,** y **B** retorna el control a **A**
    - Dos módulos están acoplados normalmente cuando no se pasan ningún parámetro entre ellos, sólo existe la llamada de uno a otro
  - **Acoplamiento de datos**
    - Los módulos se comunican mediante parámetros
    - Cada parámetro es una unidad elemental de datos
    - Todas las E/S al y desde el módulo llamado son argumentos de datos y no de control
    - Reducir el número de parámetros
  - **Acoplamiento por estampado**
    - Dos módulos están acoplados por estampado si ambos hacen referencia a la misma estructura de datos
    - No estructura de datos global
    - No es deseable si el módulo que recibe esa estructura de datos, necesita sólo parte de los elementos que se le pasan



Modularidad – Acoplamiento (vii)

- Niveles moderados se caracterizan por el paso de control entre módulos
  - **Acoplamiento de control**
    - Paso de parámetros de control entre módulos
    - Influencia de un módulo en la ejecución de otro
      - Una variable controla las decisiones en un módulo superior o subordinado
  - **Acoplamiento externo**
    - Los módulos están ligados a un entorno externo al software. Por ejemplo la E/S acopla un módulo a dispositivos, formatos y protocolos de comunicación. Este acoplamiento es esencial pero deberá estar limitado a unos pocos módulos



Modularidad – Acoplamiento (viii)

- Niveles altos de acoplamiento
  - **Acoplamiento común**
    - Los módulos acceden a datos en un área de datos global (un área de memoria accesible por ejemplo). Comparten una estructura de datos global
    - Viola los principios básicos de encapsulamiento y modularidad
    - El diagnostico de problemas en estructuras con acoplamiento común es costoso en tiempo y difícil de realizar
  - **Acoplamiento de contenido**
    - Se da cuando un módulo hace uso de datos o de información de control mantenidos dentro de los límites de otro módulo
      - Un módulo modifica algún elemento en el otro módulo
      - Un módulo utiliza una variable local del otro
      - Desde un módulo se salta a otro, pero la sentencia a la que se pasa no está definida como punto de entrada
      - Dos módulos comparten los mismos contenidos
    - Inaceptable

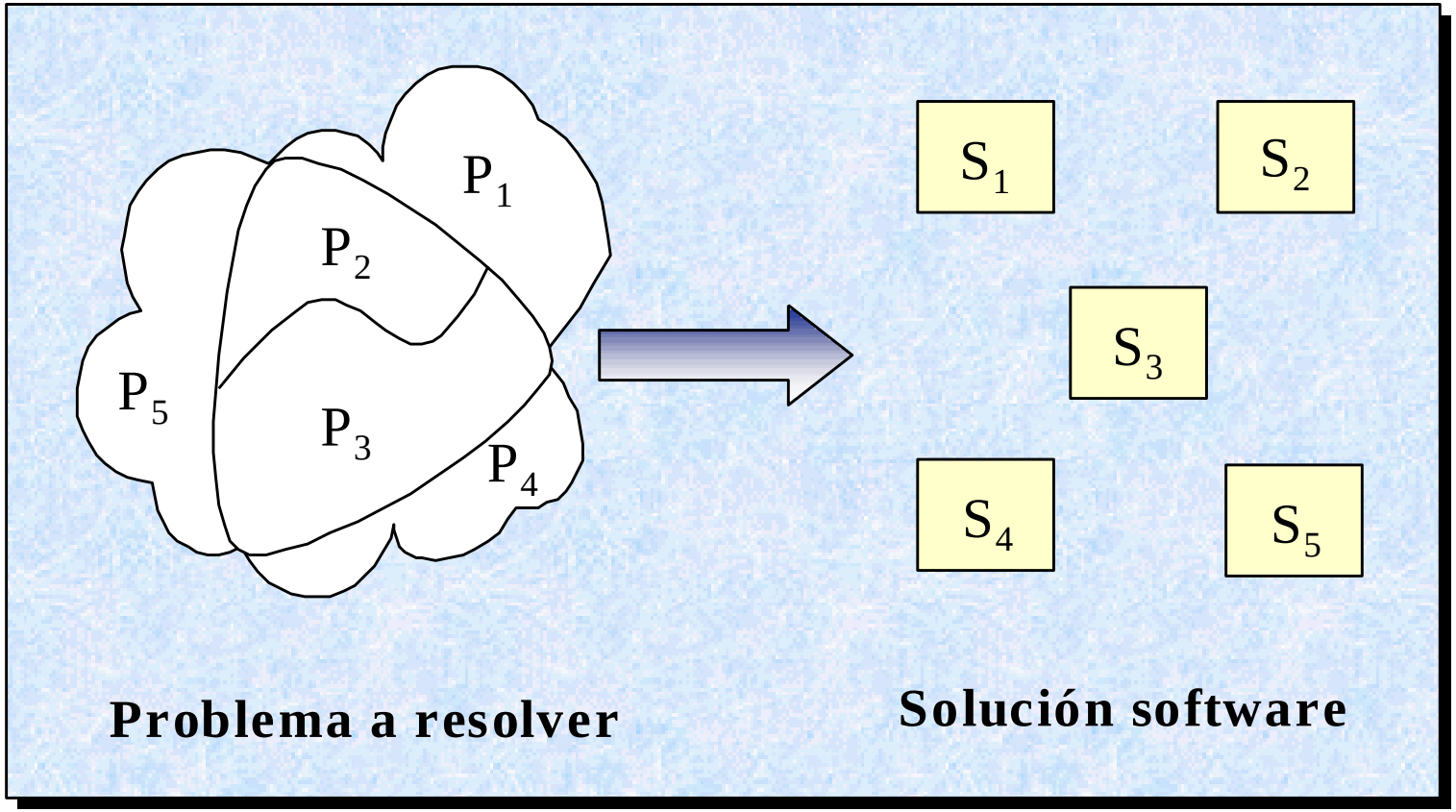
Arquitectura del software (i)

- La arquitectura del software alude a la estructura global del software y las formas en que esa estructura proporciona integridad conceptual a un sistema [Shaw y Garlan, 1995]
- La arquitectura del software es la estructura lógica y física de un sistema, forjada por todas las decisiones de diseño estratégicas y tácticas aplicadas durante el desarrollo [Booch, 1994]
- Una arquitectura software es la descripción de los subsistemas y componentes de un sistema software y de las relaciones entre ellos. Los subsistemas y componentes se especifican habitualmente desde diferentes puntos de vista para mostrar las propiedades funcionales y no funcionales relevantes de un sistema software. La arquitectura software de un sistema es un "artefacto", es el resultado de la actividad de diseño del sistema [Buschmann et al., 1996]

Arquitectura del software (ii)

- En su forma más simple se refiere a
  - La estructura jerárquica de los módulos
  - Sus interacciones
  - Sus estructuras de datos
- En un sentido más amplio
  - Generalización de los componentes
    - Elementos principales del sistema y sus interacciones
- Un objetivo del diseño software es crear una versión arquitectónica de un sistema
  - Esta versión sirve como estructura desde la cual llevar a cabo actividades de diseño más detalladas
- Un conjunto de patrones arquitectónicos permiten que el ingeniero del software reutilice los conceptos a nivel de diseño

Arquitectura del software (iii)



Arquitectura del software (iv)

- Propiedades que deberían especificarse como parte de un diseño arquitectónico [Shaw y Garlan, 1995]
  - **Propiedades estructurales**
    - Define los componentes de un sistema (módulos, objetos, filtros…) y la manera en que estos componentes se empaquetan e interactúan unos con otros
  - **Propiedades extra-funcionales**
    - La descripción del diseño arquitectónico deberá ocuparse de cómo la arquitectura de diseño consigue los requisitos para el rendimiento, capacidad, fiabilidad, seguridad, capacidad de adaptación y otras características del sistema
  - **Familias de sistemas relacionados**
    - El diseño arquitectónico deberá dibujarse sobre patrones repetibles que se basen comúnmente en el diseño de familias de sistemas similares. En esencia, el diseño deberá tener la habilidad de volver a utilizar bloques de construcción arquitectónicos

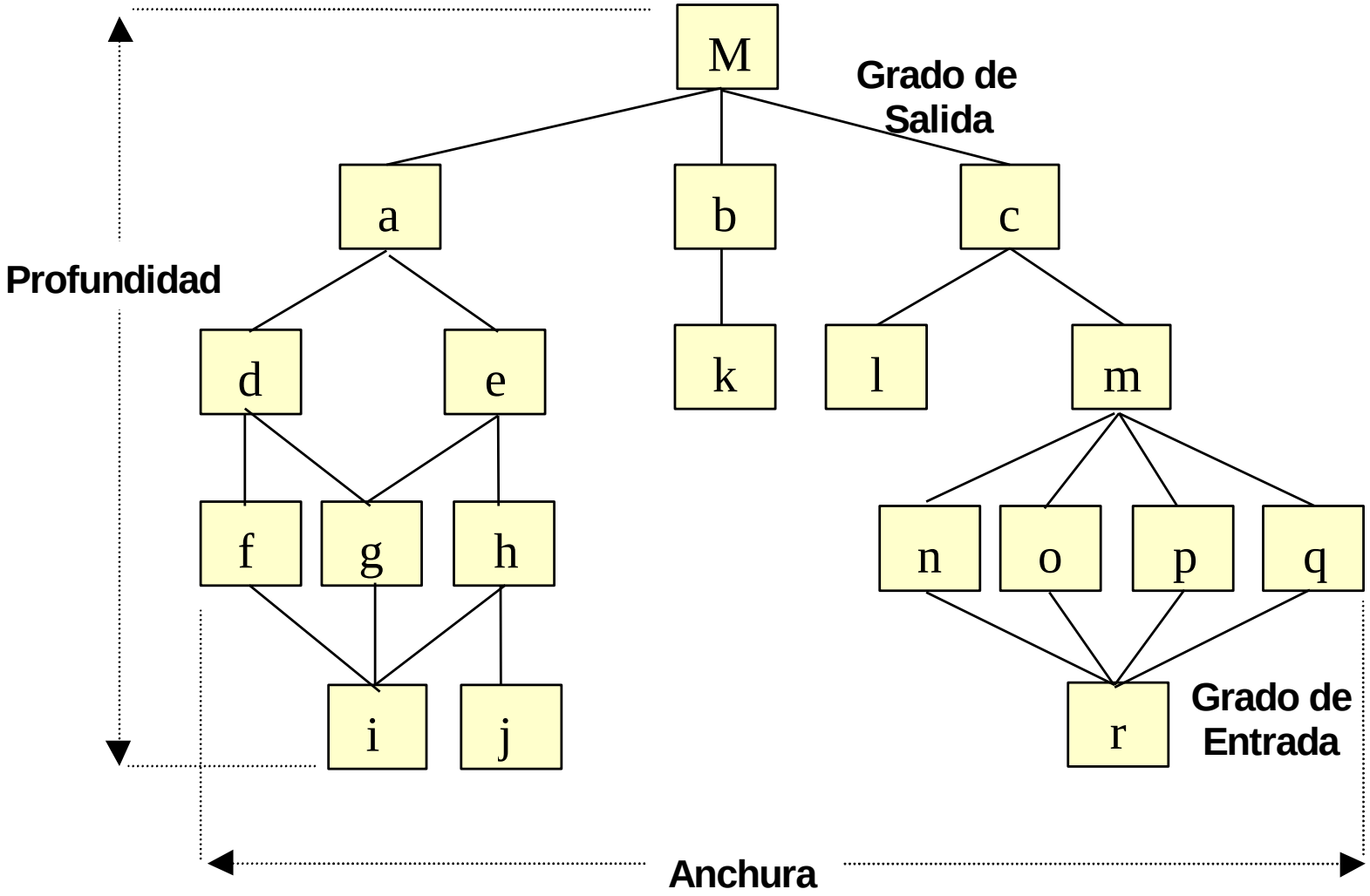
Arquitectura del software (v)

- El diseño arquitectónico se puede representar mediante uno o más modelos diferentes [Garlan y Shaw, 1995]
  - Modelos estructurales
    - Representan la arquitectura como una colección organizada de componentes de programa
  - Modelos de marco de trabajo
    - Aumentan el nivel de abstracción del diseño en un intento de identificar los marcos de trabajo (patrones) repetibles del diseño arquitectónico que se encuentran en aplicaciones similares
  - Modelos dinámicos
    - Tratan los aspectos de comportamiento de la arquitectura del programa, indicando cómo puede cambiar la estructura o la configuración del sistema en función de los acontecimientos externos
  - Modelos de proceso
    - Se centran en el diseño del proceso técnico de negocio que tiene que adaptar el sistema
  - Modelos funcionales
    - Se pueden utilizar para representar la jerarquía funcional de un sistema

Jerarquía de control (i)

- También denominada estructura de programa
- Representa la organización jerárquica de los módulos
- Jerarquía de control basada en el flujo de control entre diferentes partes de un programa
- No presenta detalles procedimentales
- No se tiene que aplicar necesariamente a todos los estilos arquitectónicos
- La forma más común de representarla es mediante un grafo que represente el control jerárquico para las arquitecturas de llamada y retorno

Jerarquía de control (ii)



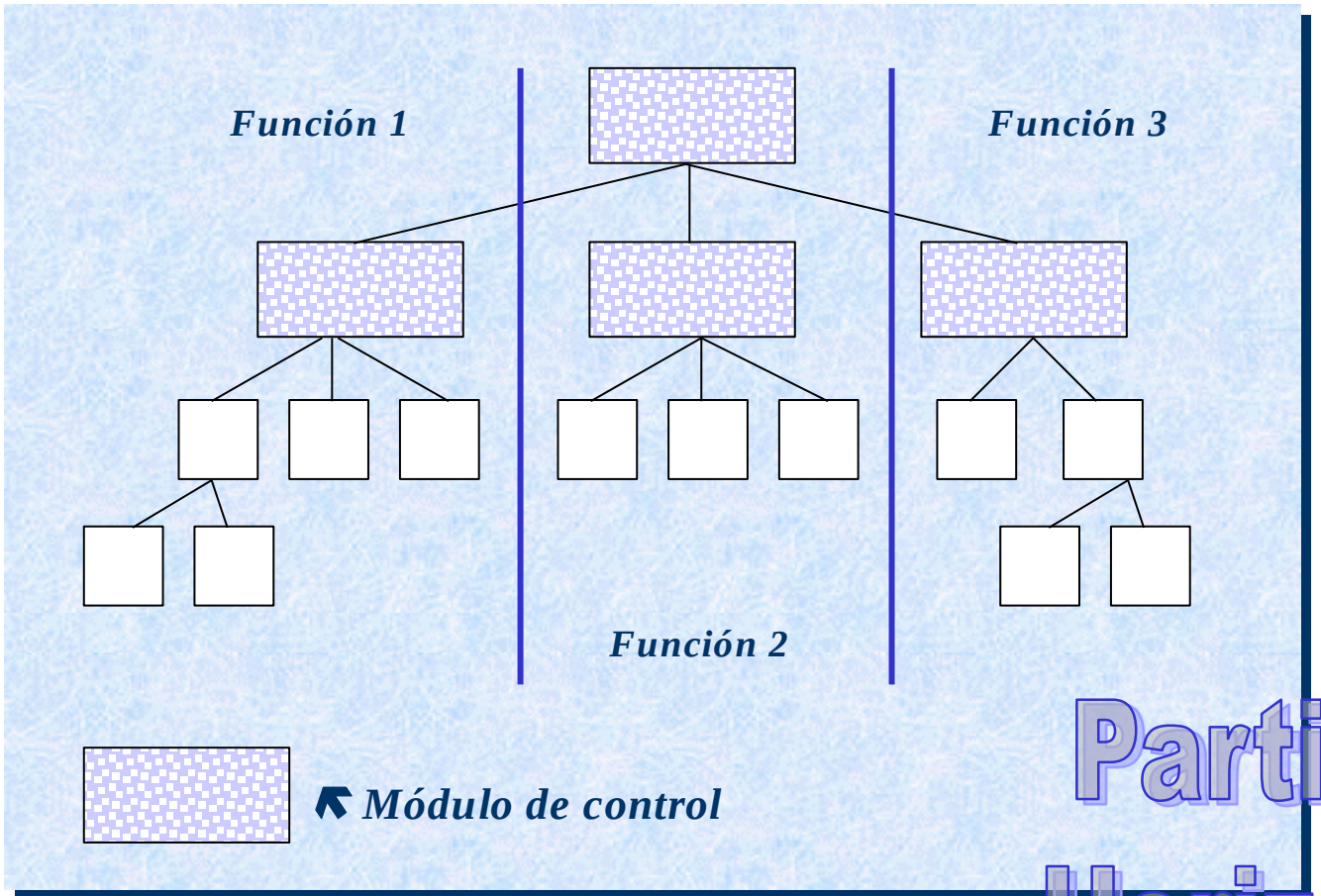
División estructural (i)

- La estructura de un programa debe partirse horizontal y verticalmente
- La partición horizontal define ramas separadas de la jerarquía modular para cada función principal del programa
  - Módulos de control
  - Enfoque entrada/proceso/salida
- Beneficios de la partición horizontal
  - Proporciona software más fácil de probar
  - Lleva a un software más fácil de mantener
  - Propaga menos efectos secundarios
  - Proporciona software más fácil de ampliar
- Puntos en contra de la partición horizontal
  - Aumenta la comunicación entre módulos, pudiendo complicar el control global del flujo del programa

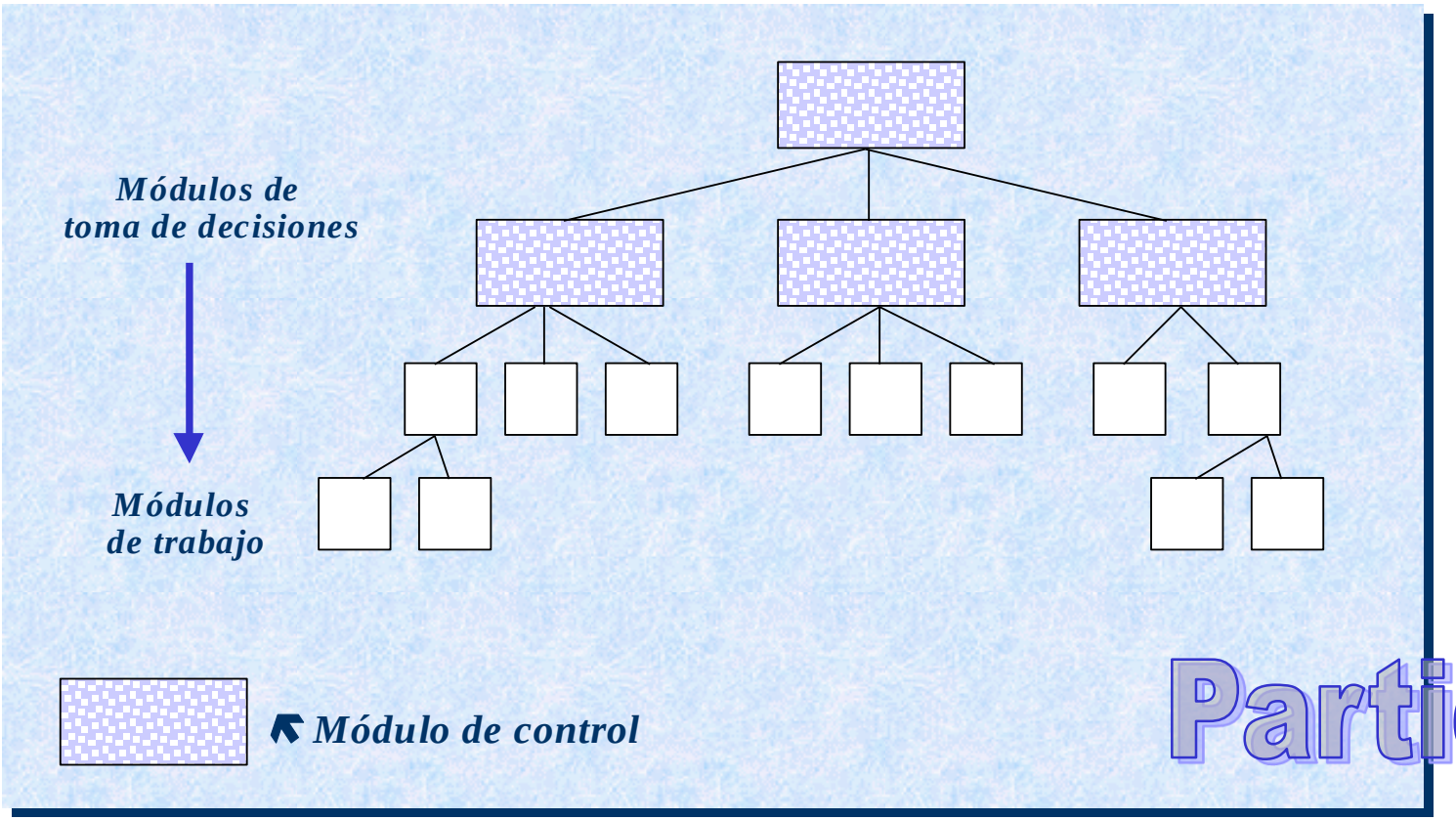
División estructural (ii)

- Partición vertical o descomposición en factores (*factoring*)
- La partición vertical expresa que el control, toma de decisiones, y el trabajo se distribuyan de forma descendente en la arquitectura del programa
- Los módulos de nivel superior deben realizar funciones de control y poco trabajo de procesamiento
- Los módulos que residen en la parte baja de la arquitectura deben de ser los que realicen las tareas de entrada, cálculo y salida

División estructural (iii)



División estructural (iv)



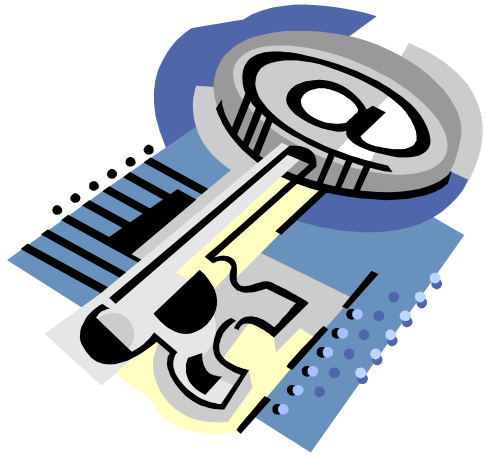
Partición Vertical

Estructura de datos

- Representación de la relación lógica existente entre los elementos individuales de datos
- La estructura de datos dicta la organización, los métodos de acceso, el grado de asociatividad y las alternativas de procesamiento para la información
- Estructuras clásicas
  - Elemento escalar
  - Vector secuencial
  - Espacio n-dimensional
  - Lista enlazada

Procedimiento del software

- Se centra en los detalles de procesamiento de cada módulo individual
- Debe proporcionar una especificación precisa del procesamiento
- Existe relación entre la estructura del programa y el procedimiento
- Debe incluir una referencia a todos los módulos subordinados al módulo que se describe



4. Aportaciones principales del tema

Aportaciones principales (i)

- El diseño es una representación significativa de ingeniería de algo que se va a construir
- El diseño del software es fundamental como base para un producto software de calidad
- El diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice
- Cada uno de los elementos del modelo de análisis proporciona la información necesaria para crear los cuatro modelos de diseño que se requieren en una especificación completa de diseño
  - Diseño arquitectónico
  - Diseño de datos
  - Diseño procedimental
  - Diseño de la interfaz

Aportaciones principales (ii)

- El diseño arquitectónico define la relación entre los elementos estructurales principales del software, los patrones de diseño que se pueden utilizar para lograr los requisitos, y las restricciones que afectan a la manera en que se pueden aplicar los patrones de diseño arquitectónico
- Una arquitectura del software es el producto de trabajo de desarrollo que ofrece la mayor inclinación a invertir en lo que se refiere a calidad, planificación y coste

Aportaciones principales (iii)

- Importancia del diseño de la interfaz de usuario
  - Para muchos usuarios de sistemas de ordenador, la frustración y la ansiedad forman parte de la vida diaria. Se esfuerzan por aprender un lenguaje de órdenes o un sistema de selección de menús que, se supone, les ayudará en su trabajo. Algunas personas sufren casos tan serios de *shock* con el ordenador, de terror al terminal o de neurosis de red, que evitan utilizar sistemas informáticos (Ben Shneiderman)
- Heurísticas en el diseño de pantallas
  - La simplicidad, la claridad y la comprensibilidad son las características deseadas
  - Situación de la información en pantalla de forma adecuada
    - Se indica como punto de partida la esquina superior izquierda de la pantalla, y se reservan áreas específicas de la pantalla para diferentes tipos de información, proporcionando una composición agradable a la vista
  - Selección de la información que se coloca en la pantalla
    - Sólo la información esencial
  - Cuidar cómo situar la información en la pantalla
    - Tipos de letra utilizar...
  - No introducir errores
  - Combinaciones de teclas estándar
  - Cuidado del color

Aportaciones principales (iv)

- El diseño no es escribir código y escribir código no es diseñar
- La consistencia del diseño y la uniformidad es crucial cuando se van a construir sistemas de gran tamaño
  - Se deberá establecer un conjunto de reglas de diseño para el equipo del software antes de comenzar a trabajar
- Los principios del diseño cuidan que un ingeniero de software cree un diseño que muestre los factores de calidad tanto internos como externos
- Un sistema software ha de descomponerse en un conjunto de módulos cohesivos y débilmente acoplados
- La ocultación de la información sirve para prevenir accidentes, no fraudes
- La ocultación de la información conduce a la modularidad efectiva
- No se debe modularizar de más, la simplicidad de cada módulo se eclipsará con las complejidad de la integración

Aportaciones principales (v)

- La cohesión es la medida de la relación funcional de los elementos de un módulo
- El acoplamiento es una medida de la interconexión entre los módulos de una estructura de programa
- Los módulos están fuertemente acoplados si utilizan variables compartidas o intercambian información de control
- Se debe evitar información compartida de forma global
  - No se deben usar variables globales

5. Cuestiones y ejercicios

Cuestiones y ejercicios

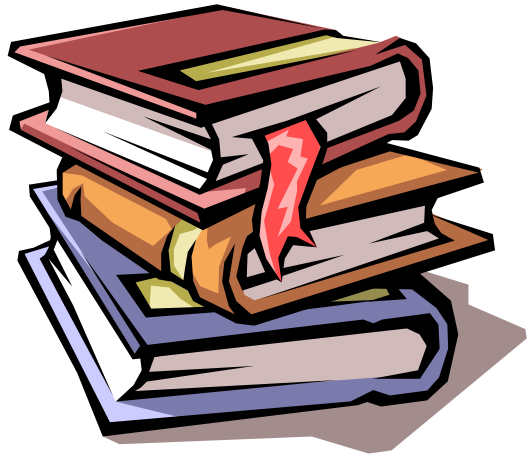
- Estudiar el acoplamiento y la cohesión en diferentes sistemas software
- Aplicar un enfoque de refinamiento paso a paso para desarrollar tres niveles diferentes de abstracción procedimental para el problema del salto del caballo
- Establecer la relación existente entre el concepto de ocultación de la información y el concepto de independencia modular

6. Lecturas complementarias

Lecturas complementarias

- **Davis, A.** "*201 Principles of Software Development*". McGraw-Hill, 1995
  - Libro dedicado a los principios de diseño del software
- **Guttag, J.** "*Abstract Data Types and the Development of Data Structures*". Communications of the ACM, 20(6):396-404. June 1977
  - Artículo que sirve de repaso al concepto de tipo abstracto de dato y su utilización para el diseño de las estructuras de datos. También es un ejemplo de la idea de que la abstracción y el refinamiento alcanzan tanto a los procesos como a los datos de un sistema software
- **Parnas, D. L.** "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972
  - Artículo clásico donde David Parnas enuncia su afamado principio de ocultación de la información
- **Vienneau, R. L., Senn, R.** "*A State of the Art Report: Software Design Methods*". http://www.dacs.dtic.mil/techs/design/Design.Title.html. March 1995
  - Informe dedicado al estado del arte del diseño software
- **Wirth, N.** "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April 1971
  - Artículo clásico donde, a través del problema de las ocho reinas, Niklaus Wirth va aplicando el principio de refinamiento sucesivo para obtener el diseño final

7. Referencias



Referencias (i)

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986

[Belady, 1990] Belady, L. A. "*Leonardo: The MCC Software Research Project*". En Ng, P. A., Yeh, R. T. (Eds.). *Modern Software Engineering: Foundations and Perspectives*. Van Nostrand Reinhold, 1990

[Booch, 1994] Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994

[Buschmann et al., 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996

[Coad y Yourdon, 1991] Coad, P., Yourdon, E. "*Object-Oriented Design*". Yourdon Press, 1991

[Dahl et al., 1972] Dahl, O.-J., Dijkstra, E., Hoare, C. A. R. "*Structured Programming*". Academic Press, 1972

[Dennis, 1973] Dennis, J. "*Modularity*". En *Advanced Course on Software Engineering*, F. L. Bauer (Ed.). Springer-Verlag, pp. 128-192, 1973

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995

Referencias (ii)

[Garlan y Shaw, 1995] Garlan, D., Shaw, M. "*An Introduction to Software Architecture*". En Ambriola, V., Tortora, G. (Eds.), *Advances in Software Engineering and Knowledge Engineering, Vol. 1*. World Scientific Publishing Company, 1995

[Graham, 1994] Graham, I. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition*". IEEE Computer Society Press, 1999

[Jackson, 1975] Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975

[Meyer, 1997] Meyer, B. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997

[Myers, 1978] Myers, G. "*Composite/Structured Design*". Van Nostrand Reinhold, 1978

[Parnas, 1972] Parnas, D. L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972

[Pressman, 1992] Pressman, R. S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992

[Pressman, 2002] Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill. 2002

[Pressman, 2006] Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 6ª Edición. McGraw-Hill. 2006

Referencias (iii)

[RAE, 2001] Real Academia Española "*Diccionario de la Lengua Española*". Vigésimo segunda edición. http://www.rae.es. [Última vez visitado, 10-12-2007]. 2001

[Shaw y Garlan, 1995] Shaw, M., Garlan, D. "*Formulations and Formalisms in Software Architecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995

[Shaw y Garlan, 1996] Shaw, M., Garlan, D. "*Software Architecture: Perspectives on a Emerging Discipline*". Prentice-Hall, 1996

[Sommerville, 2005] Sommerville, I. "*Ingeniería del Software*". 7ª Edición, Addison-Wesley. 2005

[Stevens, 1991] Stevens, W. P. "*Software Design: Concepts and Methods*". Prentice Hall Intenational Ltd., 1991

[Stevens et al., 1974] Stevens, W. P., Myers G. J., Constantine, L. L. "*Structured Design*". IBM Journal, 13(2):115-119, 1974

[Taylor, 1959] Taylor, E. S. "*An Interim Report on Engineering Design*". Massachusetts Institute of Technology, 1959

[Warnier, 1974] Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974

[Wasserman, 1983] Wasserman, A. "*Information Systems Design Methodology*". En *Software Design Techniques*. P. Freeman, A. Wasserman (Eds.), 4th Edition. IEEE Computer Society Press, 1983

Referencias (iv)

[Wasserman, 1996] Wasserman, A. "*Toward a Discipline of Software Engineering*". IEEE Software, 13(6):23-31. November/December 1996

[Webster, 1988] Webster, D. E. "*Mapping the Design Information Representation Terrain*". Computer, 21(12), 8-23. December 1988

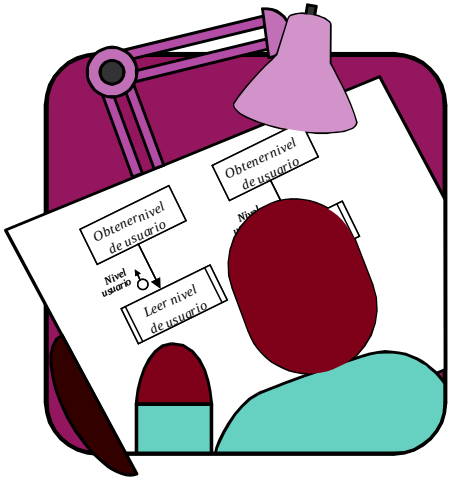
[Wirfs-Brock et al., 1990] Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice-Hall, 1990

[Wirth, 1971] Wirth, N. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April 1971

Ingeniería del Software

Tema 5: Principios del diseño del software

Dr. Francisco José García Peñalvo
(fgarcia@usal.es)
Miguel Ángel Conde González
(mconde@usal.es)
Sergio Bravo Martín
(ser@usal.es)



3º I.T.I.S.
Fecha de última modificación: 16-10-2008

Universidad de Salamanca – Departamento de Informática y Automática