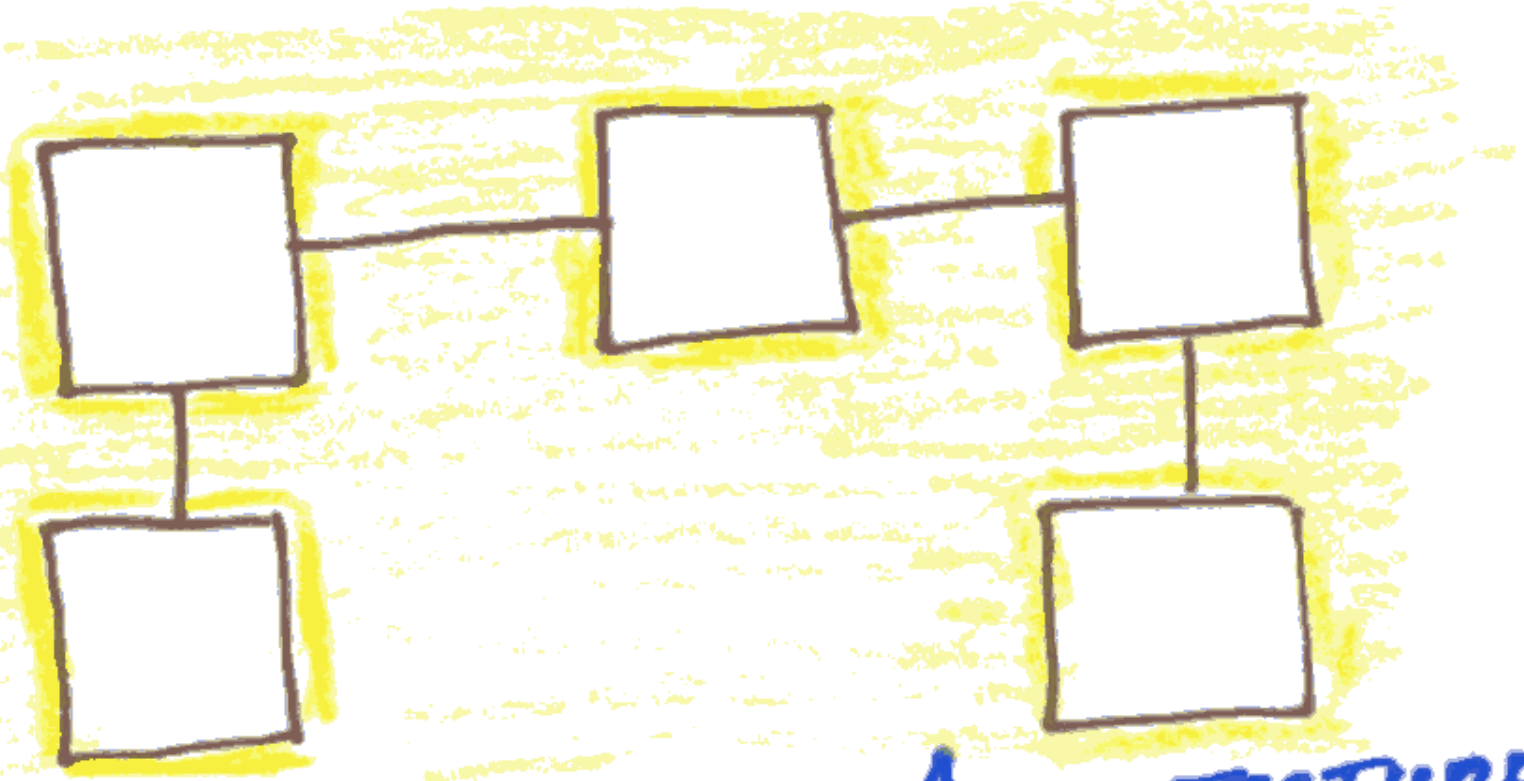

Universidad
cenfotec
Tecnologías de Información y Comunicación

Arquitectura Conceptual

(Componentes)

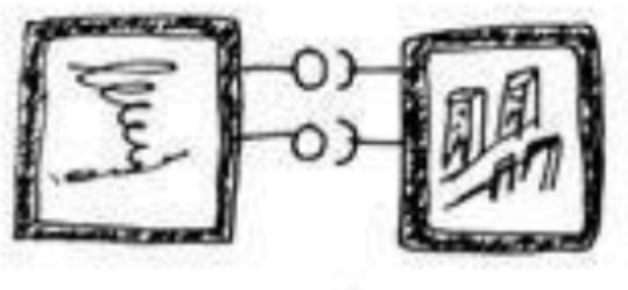


- Conceptual Architecture
  - Component Responsibilities
  - System-Wide Characteristics
  - Communicating Up & Out

ABSTRACT

Componentes y relaciones

- La arquitectura conceptual identifica los componentes de alto nivel del sistema y las relaciones entre ellos
- Un componente es:
  - Una unidad modular de software funcional
  - Accedida a través de una o más interfaces

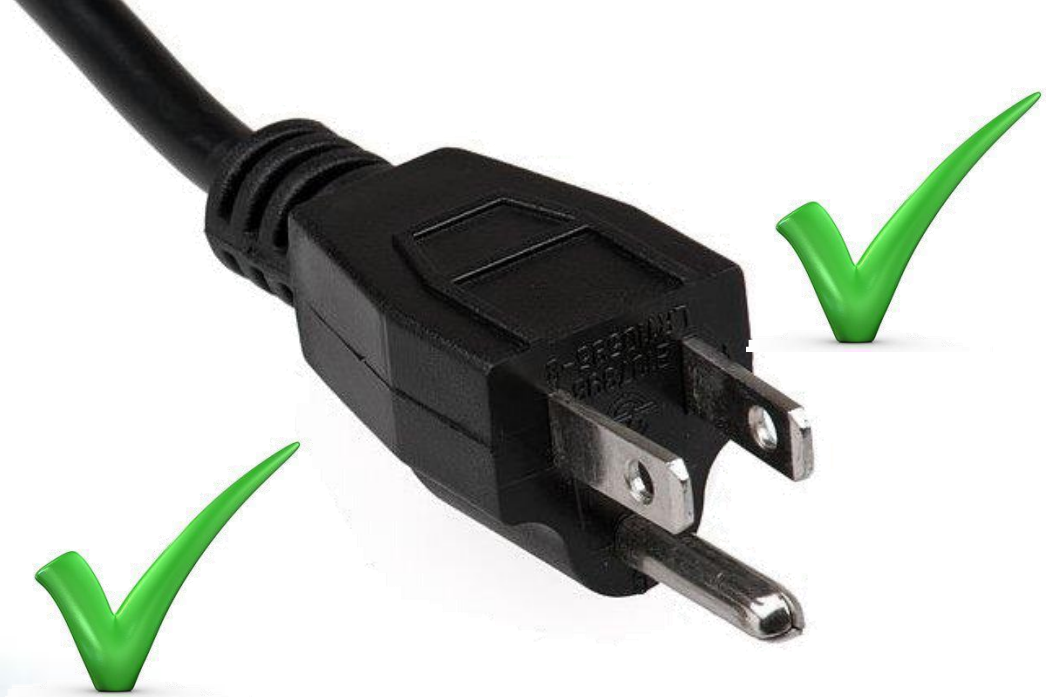


Separación de responsabilidades!

Qué es un componente?

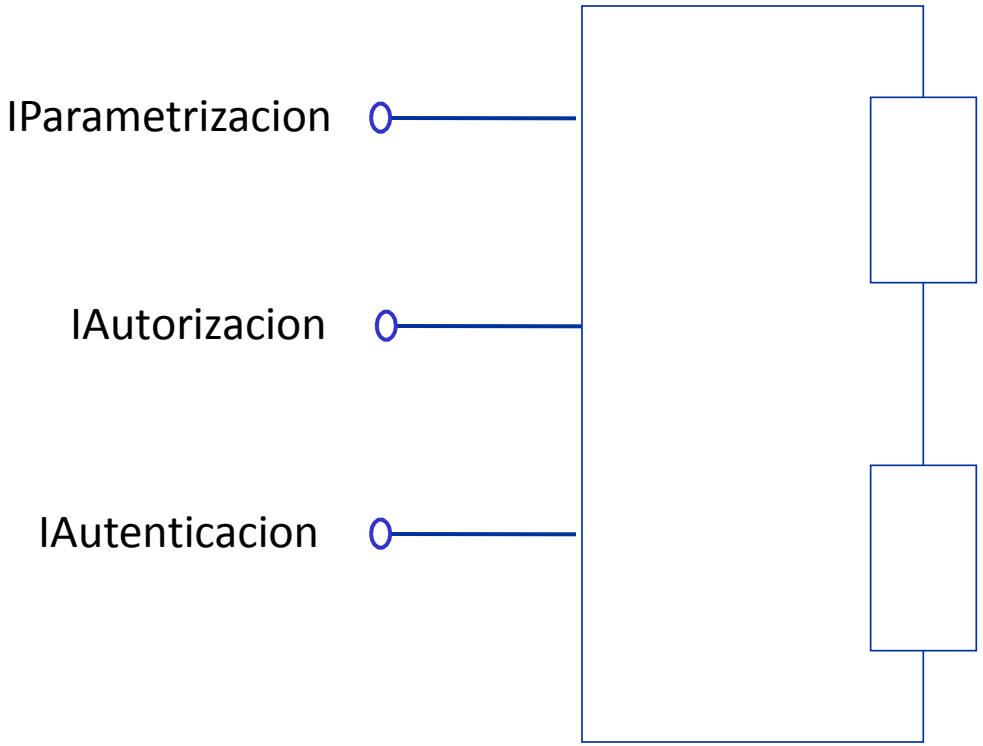
- “elemento de software que se conforma a un modelo de componentes y que puede ser **instalado** y **combinado** de manera **independiente**” (Heineman & Council, 2001,pp 7)

- “unidad de composición con **interfaces** que especifican **su contrato**” (Szypersky, 1997, pp 34)

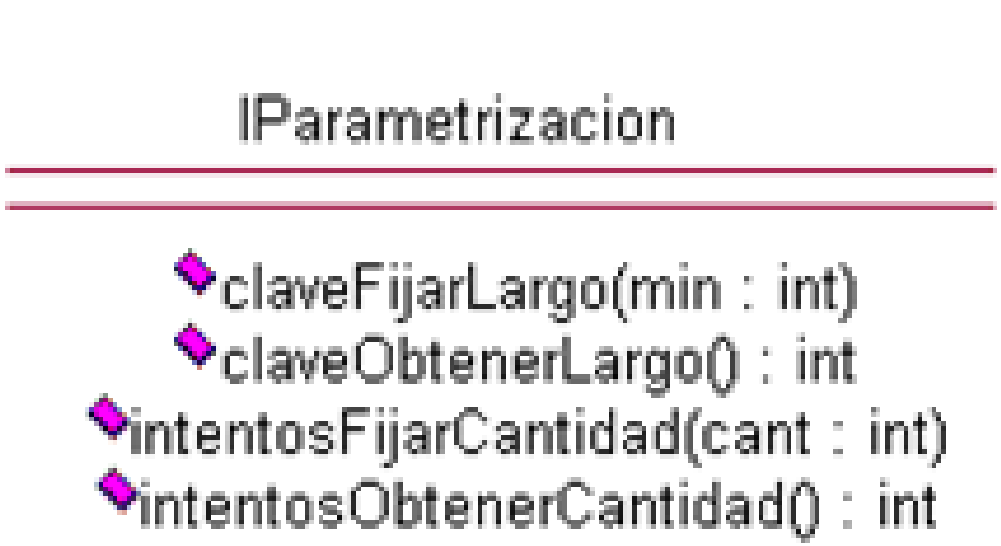


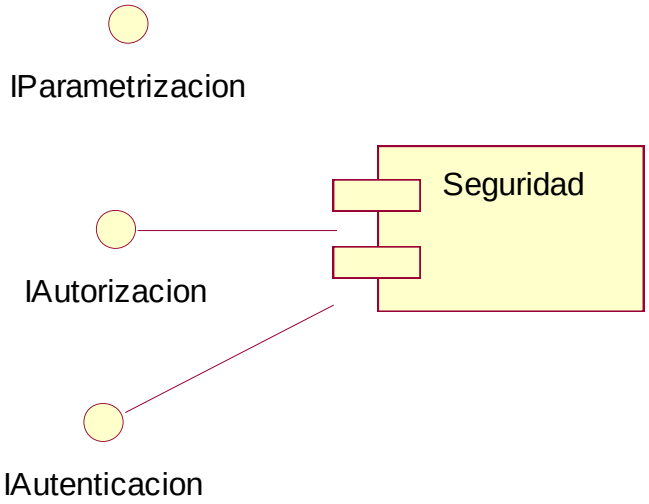
Ejemplo: Seguridad

- Este es un componente para administrar la seguridad de un sistema.
- Parametrización, gestión de usuarios, grupos, derechos, etc.



Seguridad: Especificación

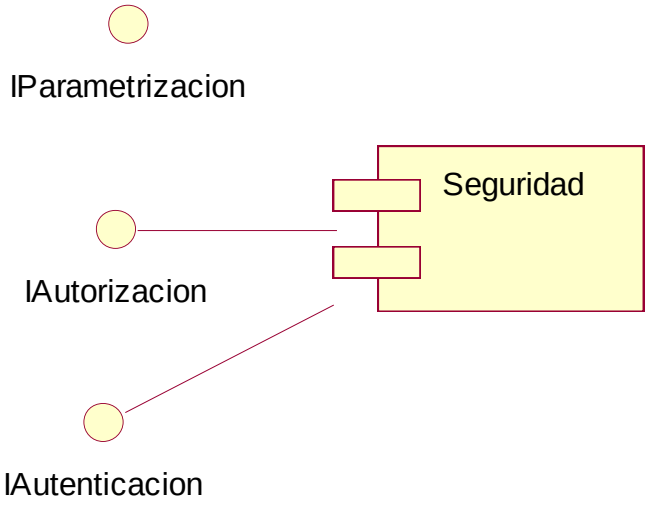




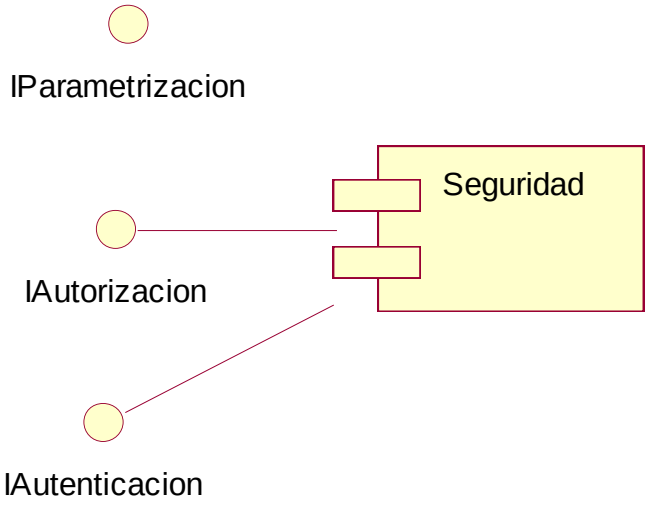
Seguridad: Especificación

IAutorizacion

- derechoCrear(id : String, descripcion : String)
- derechoObtenerTodos() : java.util.Vector
- derechoBorrar(id : String)
- grupoCrear(id : String, desc : String)
- grupoDarDerecho(grp : String, der : String)
- grupoQuitarDerecho(grp : String, der : String)
- grupoObtenerDerechos(grp : String) : java.util.Vector
- grupoObtenerTodos() : java.util.Vector
- grupoAgregarUsuario(usr : String, grp : String)
- grupoQuitarUsuario(grp : String, usr : String)
- grupoBorrar(id : String)
- usuarioCrear(id : String, cla : String)
- usuarioDarDerecho(usr : String, der : String)
- usuarioQuitarDerecho(usr : String, der : String)
- usuarioObtenerDerechos(usr : String) : java.util.Vector
- usuarioObtenerGrupos(usr : String) : java.util.Vector
- usuarioObtenerTodos() : java.util.Vector
- usuarioBorrar(id : String)



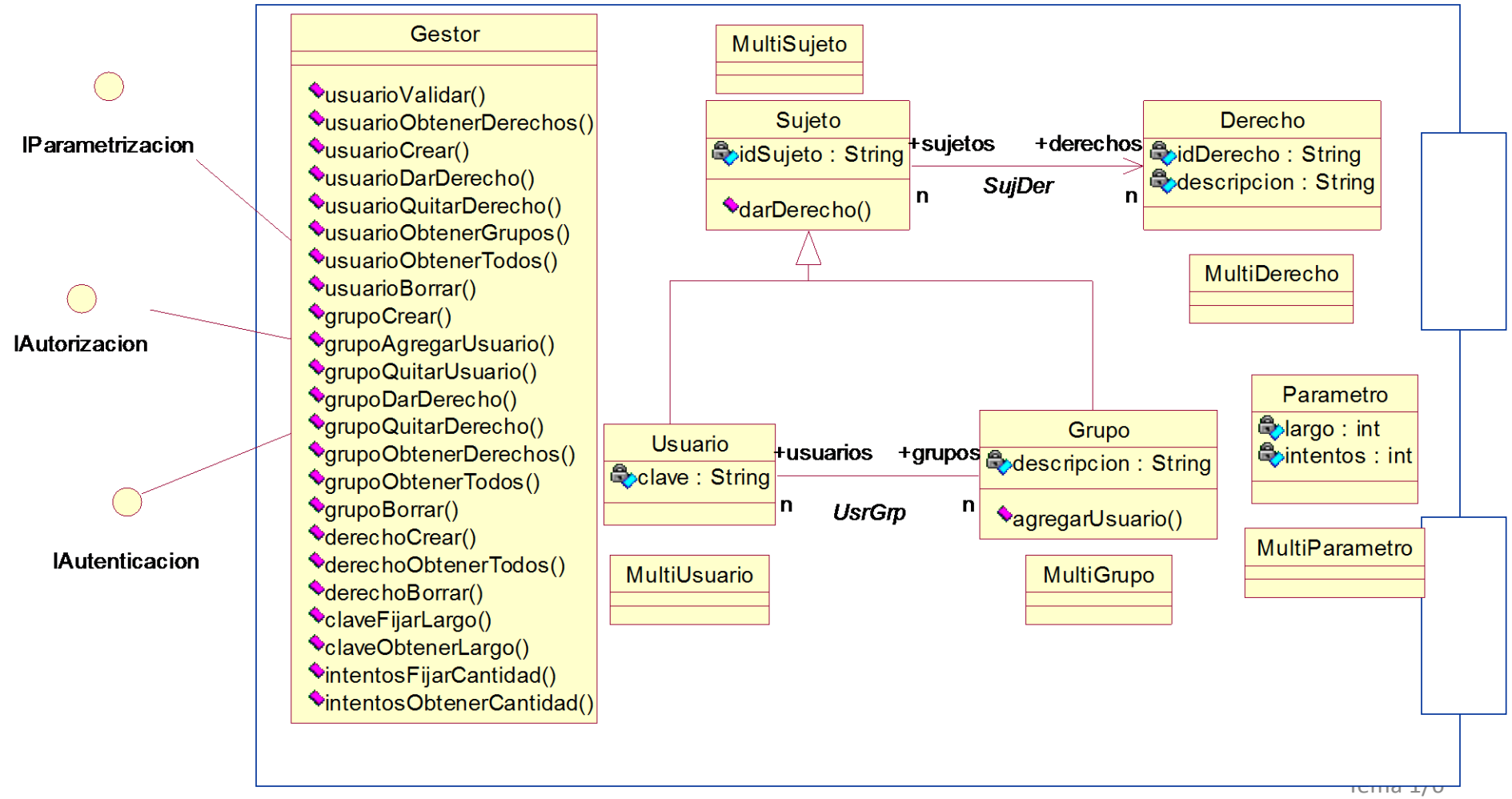
Seguridad: Especificación



IAutenticacion

usuarioValidar(usr : String, clave : String) : boolean
usuarioObtenerDerechos(usr : String) : java.util.Vector

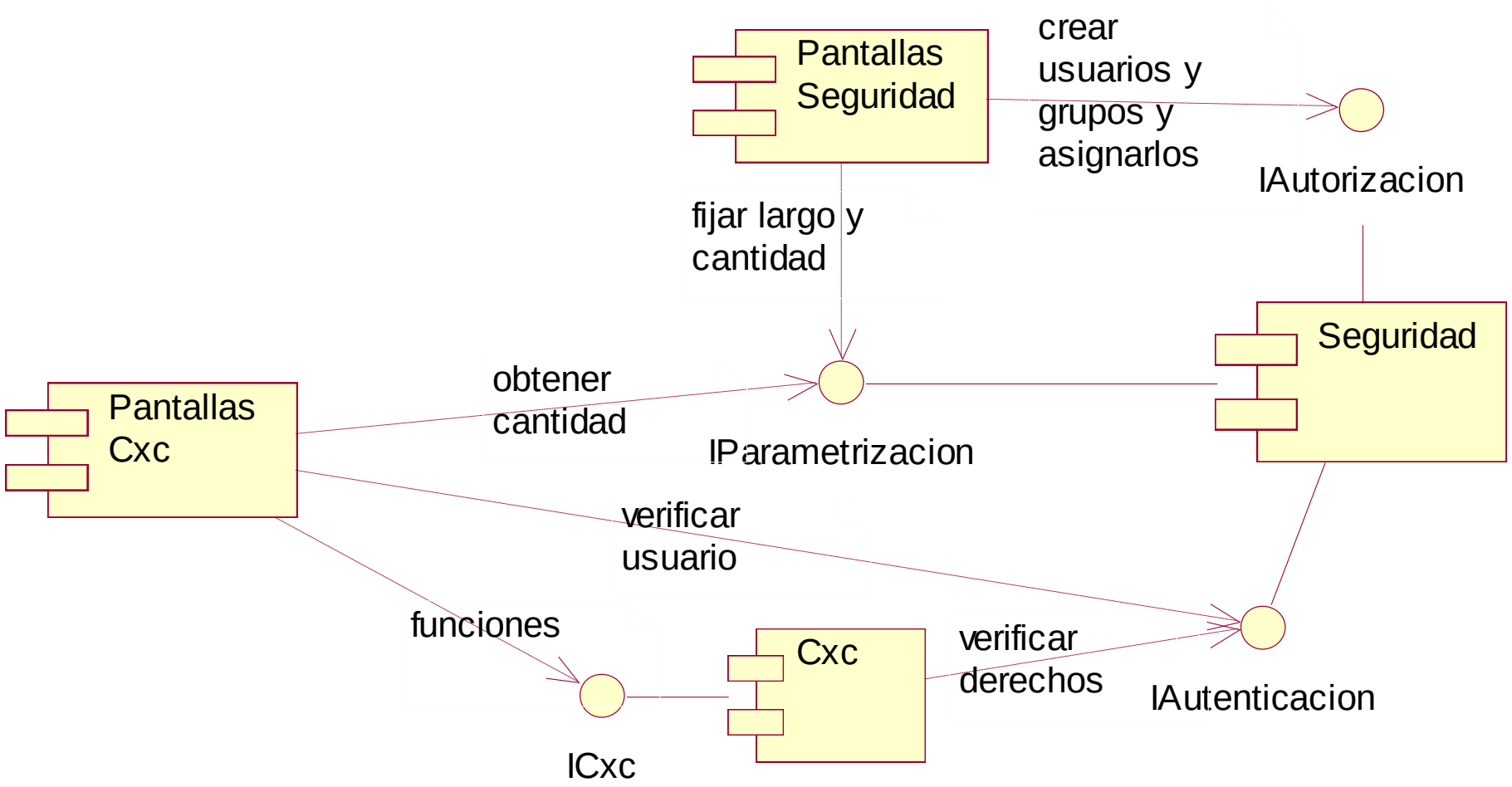
Seguridad: Diseño



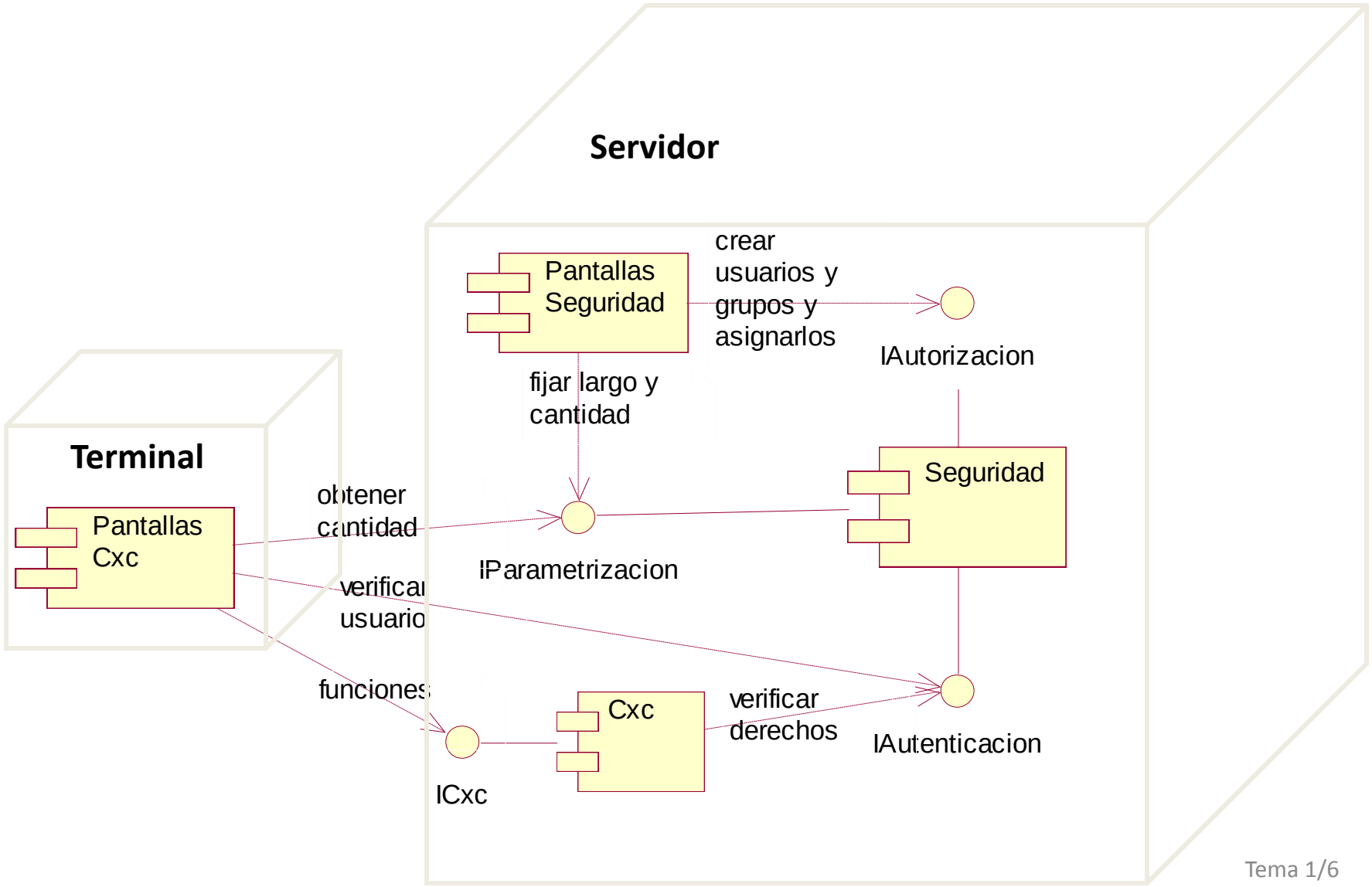
Empacado de componentes

- COM: DLL
- .NET: DLL
- EJB: JAR, con descriptor en XML
- WSDL

Diagramas de componentes en UML



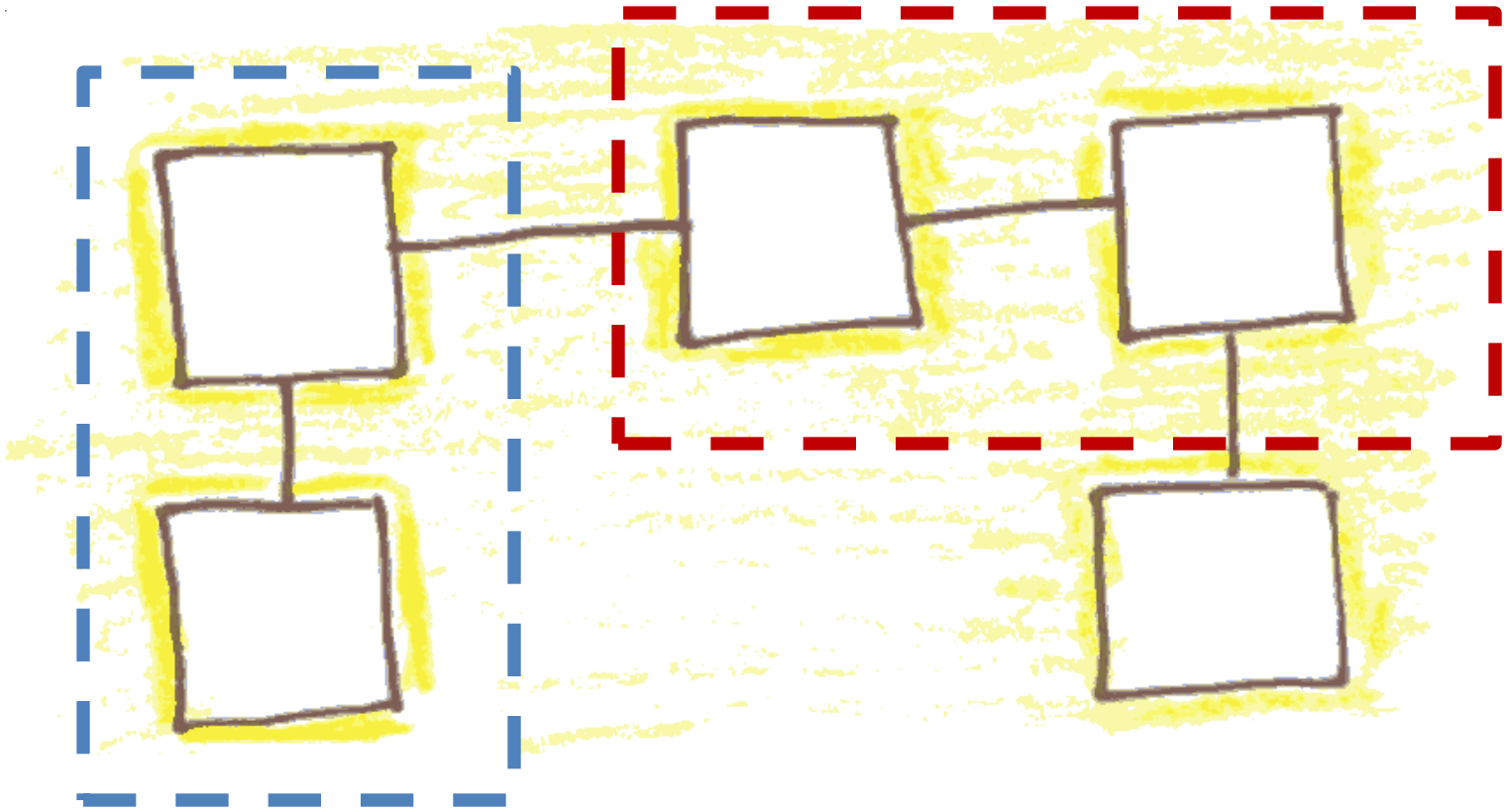
Diagramas de Distribución



¿Y los demás subsistemas de la aplicación?

…la aplicación no solamente tiene los componentes de seguridad…

Diagrama de arquitectura (conceptual)

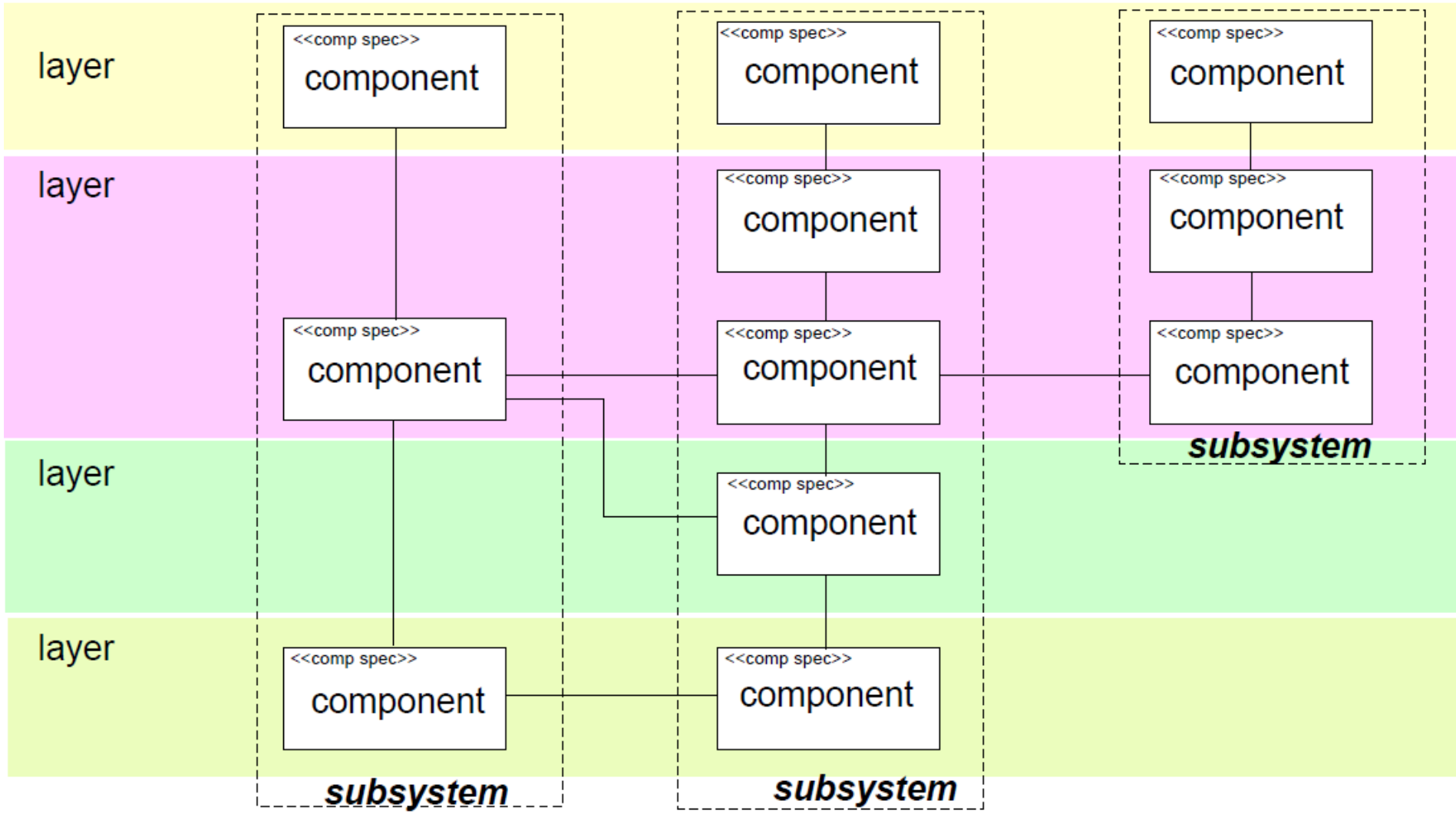


Sub sistemas

- Para aquellos sistemas de complejidad mayor, puede ser útil la identificación de sub sistemas o fragmentos del sistema mayor (‘*chunks*’)
  - Facilita la comprensión y descripción del sistema a los stakeholders
  - Se divide el trabajo de forma que partes del mismo pueden ser asignados a diferentes equipos
  - Promueve el enfoque o especialización

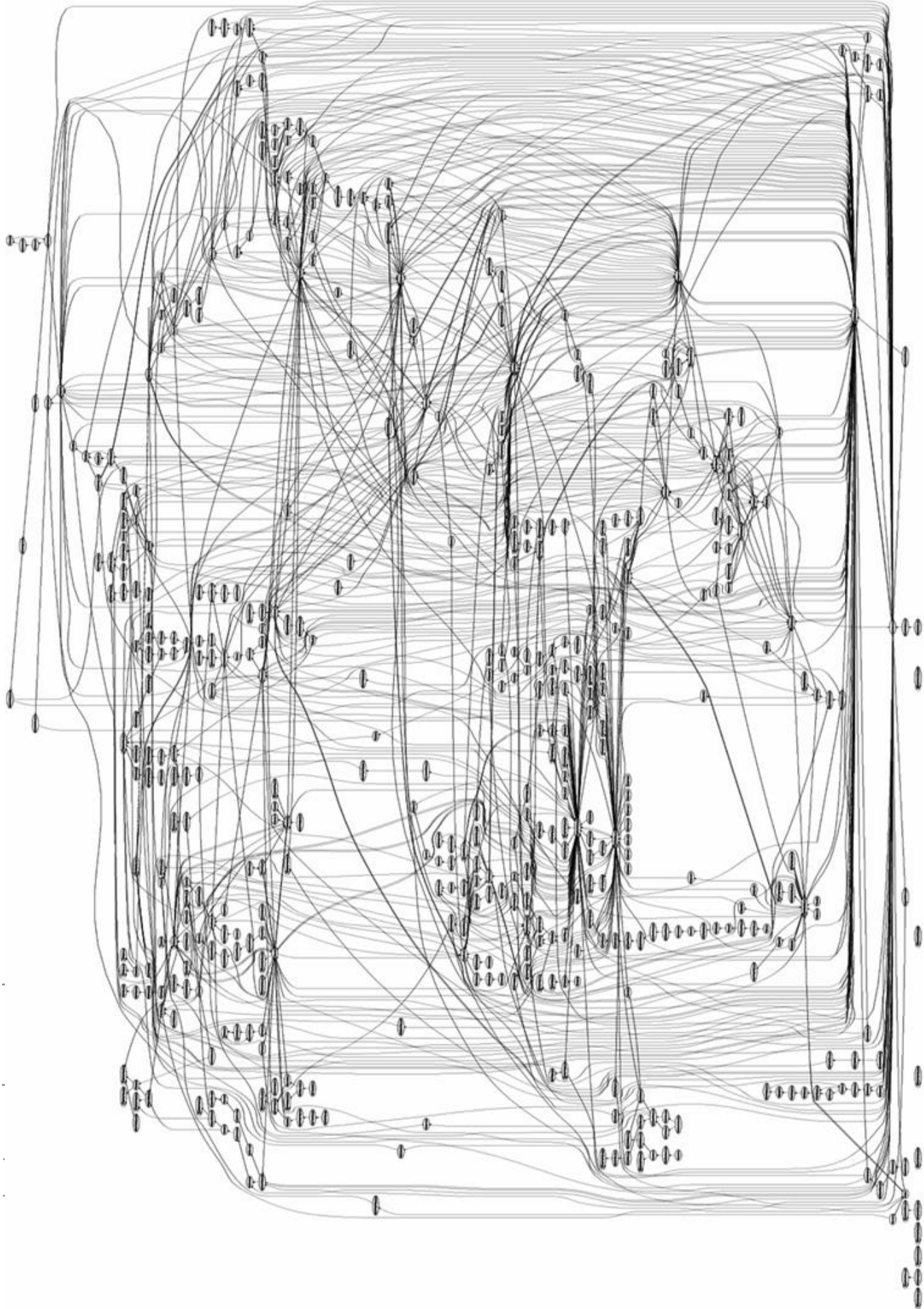
Divide y vencerás!

Diagrama de arquitectura (identificación de sub sistemas)



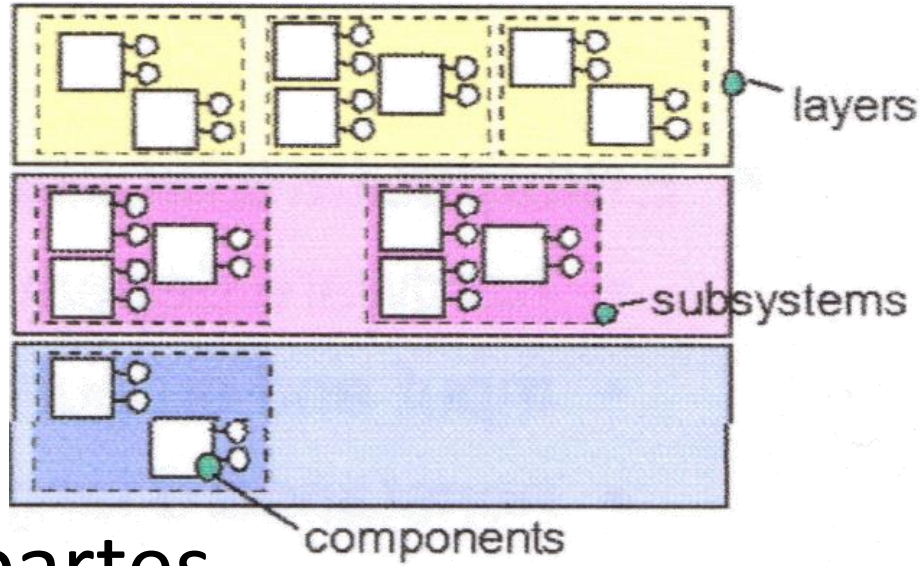
Descomposición : Axiomas

- Axiomas fundamentales de diseño
  - **–Separación de responsabilidades:**
    - Un problema complejo puede ser resuelto de mejor forma expresado por la solución de problemas independientes cada uno más pequeño
  - **–Comprensión**
    - La mente no puede manipular fácilmente más de 7 cosas al mismo tiempo
  - **–Simplificar, simplificar, simplificar**
    - “La solución más simple es usualmente la más correcta” Rechtin 1991



Descomposición : Técnicas

- **Divide y vencerás**
  - Divida el problema en partes pequeñas, más manejables y entendibles
- **Separación de responsabilidades**
  - Asigne distintas responsabilidades a partes separadas del sistema



Descomposición : Técnicas

- **'Ocultamiento' de información**
  - Esconda los detalles detrás de las interfaces, las entrañas del componente son secretas y no disponibles desde el exterior
  - La meta es proteger los clientes del componente evitándoles conocer detalles irrelevantes y
  - Evitar que conozcan cambios internos del componente

Proteja la implementación

```
def elevar (a, b):
    resultado = 1
    while b >= 0 :
        if b == 0:
            return resultado
        else:
            resultado = resultado * a
            b = b - 1
```

```
def elevar (a, b):
    if b==0:
        return 1
    else:
        return a * elevar (a, b-1)
```

```
def elevar (a, b):
    return pow(a,b)
```

```
int elevar (int a, int b)
 {
   if (b == 0) return 1;
   else return a * elevar (a, b-1);
 }
```

Programas que acceden la funcionalidad

```
x = elevar (2, 3)
```

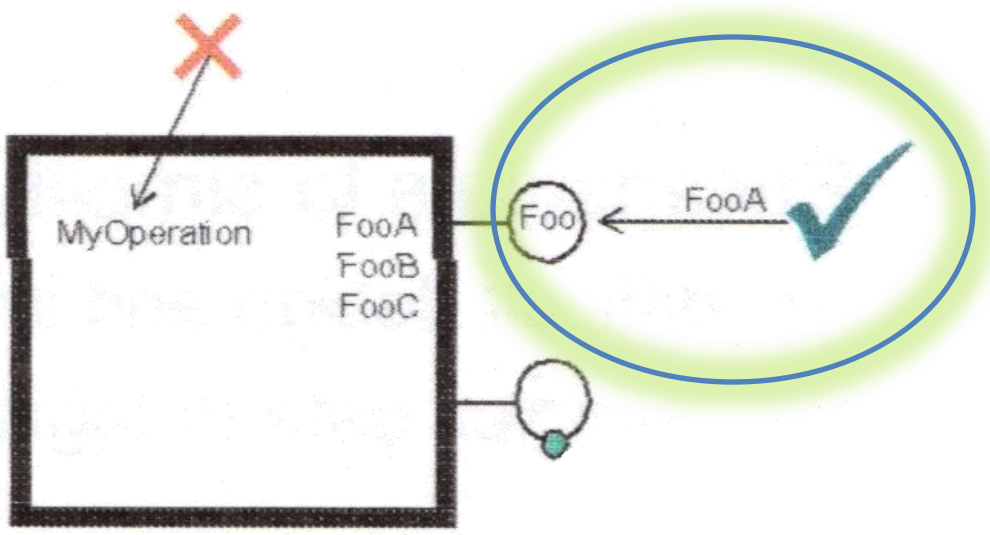
```
x = elevar (2, 3)
```

```
x = elevar (2, 3)
```

```
w = elevar (2,3)
```

Descomposición : Técnicas

- **Encapsulamiento**
  - Solo permita accesos a través de las interfaces, los clientes deben depender de las especificaciones más no de las implementaciones



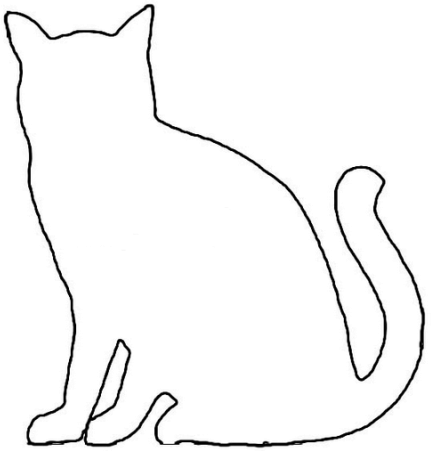
Descomposición : Técnicas

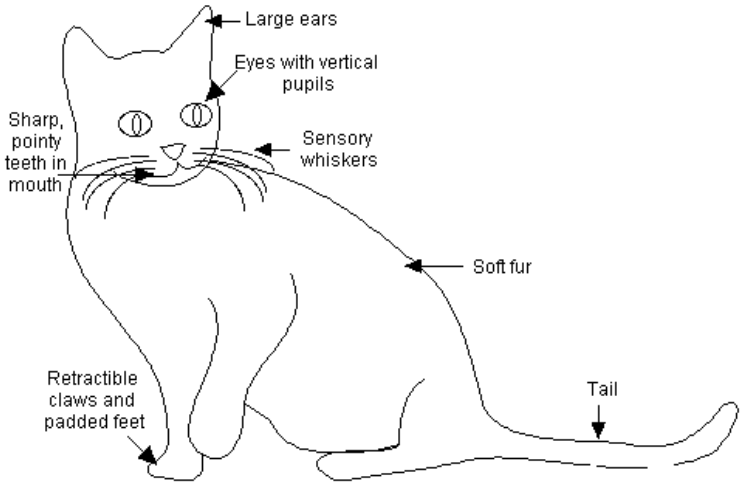
- **Contratos**
  - Si se desea hacer cumplir el encapsulamiento se deben proveer contratos de interfaz bien definidos, completos y sobre todo accesibles
- ***Factoring***
  - "Factorice" los elementos comunes para simplificar el diseño y aumentar la reutilización
- **Abstracción**
  - En esta parte evite el bajo nivel de detalle

Niveles de abstracción

- Alto nivel
- Bajo nivel

Menos detalle **Más detalle**



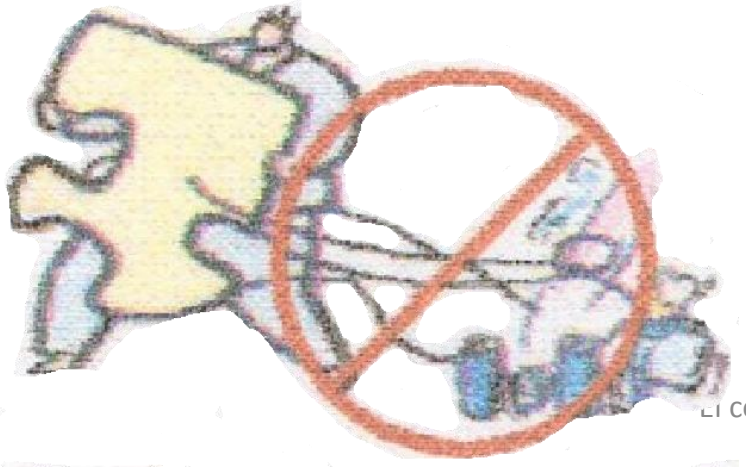


Lineamientos de identificación

- No más hoja de trabajo en blanco!
- Inicie con lo que tiene a disposición
  - **Componentes** identificados previamente
  - Mantenga un catálogo de componentes ‘a la vista’
- Identifique componentes candidatos de alto nivel que serán parte de la solución

Lineamientos de identificación

- Aplique principios de diseño
  - **Independencia** usado para alcanzar entendimiento, robustez, reusabilidad, 'modificabilidad'
    - Cohesión interna alta y bajo acoplamiento
    - Agrupe piezas de funcionalidad fuertemente relacionadas
    - Considere el uso de patrones de diseño
    - Considere ubicar juntas las piezas sometidas a cambios constantes



Lineamientos de identificación

- Si puede cree prototipos no funcionales
- Haga modelos
- Evalúe alternativas (cuán bien se cubren los requerimientos)
- Mantenga actualizado su inventario de componentes

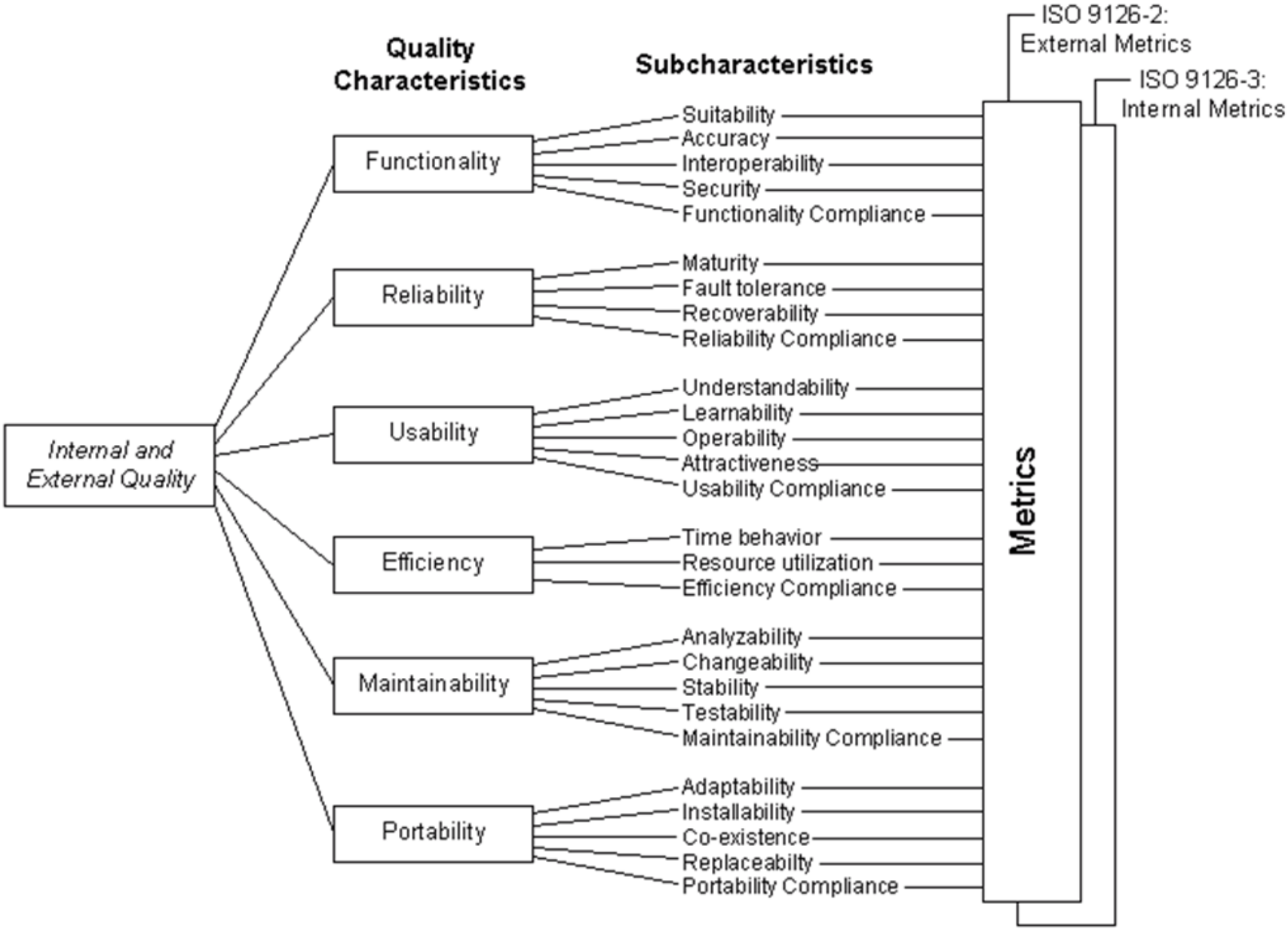
Validación: arquitectura conceptual

- Muestre como se refleja la Meta-Arquitectura en la Arquitectura Conceptual
- **Identifique *issues* pontenciales**
- Realice un análisis de las consideraciones de seguridad, rendimiento y escalabilidad

Validación: arquitectura conceptual

- Evalúe los componentes según los siguientes criterios:
  - **Claridad**
    - ¿Cada componente tiene claramente una y solo una responsabilidad definida?
  - **Acoplamiento**
    - ¿Existen componentes con un número sorprendente de interacciones?
  - **Cobertura**
  - ¿Todas las funcionalidades han sido asignadas a los componentes?

Atributos de calidad ISO 9126



Arquitectura Conceptual

(Componentes)


Universidad
cenfotec
Tecnologías de Información y Comunicación