Contenido

[Capítulo 1 y 2: Introducción 5](#_gjdgxs)

[Definición de Vista 5](#_30j0zll)

[Definición de Arquitectura de Software 5](#_1fob9te)

[Otra definición: 5](#_3znysh7)

[Importancia de la Arquitectura 5](#_2et92p0)

[Tipos de Arquitectura 6](#_tyjcwt)

[Estructura de Módulo 6](#_3dy6vkm)

[Estructura de Componentes y Conectores 6](#_1t3h5sf)

[Estructura de Asignación 6](#_4d34og8)

[¿Qué estructura usar? 7](#_2s8eyo1)

[Resúmen 7](#_1yyy98l)

[Capítulo 18: Documentación 9](#_17dp8vu)

[¿Cuáles son los puntos de vista relevantes? 9](#_3rdcrjn)

[La documentación para obtener una vista contiene 9](#_26in1rg)

[La documentación que se aplica a todos los puntos de vista contiene 9](#_35nkun2)

[Principales Vistas 10](#_1ksv4uv)

[Vistas de Módulos (Module View) 10](#_44sinio)

[Vistas de Componentes y Conectores 14](#_2jxsxqh)

[Vista de despliegue 14](#_z337ya)

[Capítulo 4 a 11: Atributos de Calidad 15](#_3j2qqm3)

[Confiabilidad 15](#_1y810tw)

[Modificabilidad (Escalabilidad) 15](#_4i7ojhp)

[Eficiencia 16](#_2xcytpi)

[Seguridad 16](#_1ci93xb)

[Usabilidad 16](#_3whwml4)

[Testeabilidad 16](#_2bn6wsx)

[Portabilidad 16](#_qsh70q)

[Interoperabilidad 17](#_3as4poj)

[Disponibilidad 17](#_1pxezwc)

[¿Cómo se relacionan los atributos de calidad? 19](#_49x2ik5)

[Tácticas 20](#_2p2csry)

[1. Tácticas de disponibilidad 20](#_147n2zr)

[Detección de errores 20](#_3o7alnk)

[Recuperación de errores – preparación y arreglo 20](#_23ckvvd)

[Votación 20](#_ihv636)

[Redundancia activa 20](#_32hioqz)

[Redundancia pasiva 20](#_1hmsyys)

[Repuesto 20](#_41mghml)

[Recuperación de errores – reintroducción de componentes 21](#_2grqrue)

[Modo sombra 21](#_vx1227)

[Resincronización de estados 21](#_3fwokq0)

[Checkpoint/rollback 21](#_1v1yuxt)

[Prevención de errores 21](#_4f1mdlm)

[Remoción del servicio 21](#_2u6wntf)

[Transacciones 21](#_19c6y18)

[Monitoreo de procesos 21](#_3tbugp1)

[2. Tácticas de modificabilidad 21](#_28h4qwu)

[Localización de cambios 21](#_nmf14n)

[Coherencia semántica 21](#_37m2jsg)

[Anticipar cambios 21](#_1mrcu09)

[Generalizar módulos 22](#_46r0co2)

[Limitar opciones 22](#_2lwamvv)

[Prevenir efecto dominó 22](#_111kx3o)

[Ocultamiento de información 22](#_3l18frh)

[Mantener interfaces existentes 22](#_206ipza)

[Restringir caminos de comunicación 22](#_4k668n3)

[Utilizar intermediarios 22](#_2zbgiuw)

[Diferir tiempo de enlaces 22](#_1egqt2p)

[Registro en tiempo de ejecución 23](#_3ygebqi)

[Archivos de configuración 23](#_2dlolyb)

[Polimorfismo 23](#_sqyw64)

[Reemplazo de componentes 23](#_3cqmetx)

[Adherencia a protocolos 23](#_1rvwp1q)

[3. Tácticas de performance 23](#_4bvk7pj)

[Demanda de recursos 23](#_2r0uhxc)

[Reducir recursos que hacen falta para procesar un evento 23](#_1664s55)

[Reducir número de eventos procesados 23](#_3q5sasy)

[Manejo de recursos 23](#_25b2l0r)

[Introducir concurrencia 23](#_kgcv8k)

[Mantener copias de datos 23](#_34g0dwd)

[Aumentar recursos disponibles 23](#_1jlao46)

[Arbitraje de recursos 24](#_43ky6rz)

[Políticas de planificación 24](#_2iq8gzs)

[4. Técnicas de seguridad 24](#_xvir7l)

[Resistir ataques 24](#_3hv69ve)

[Autenticación de usuarios 24](#_1x0gk37)

[Autorización de usuarios 24](#_4h042r0)

[Confidencialidad de los datos 24](#_2w5ecyt)

[Integridad de los datos 24](#_1baon6m)

[Limitar la exposición de los datos 24](#_3vac5uf)

[Limitar el acceso 24](#_2afmg28)

[Recuperación de ataques 24](#_pkwqa1)

[Restaurar estado 24](#_39kk8xu)

[Identificar ataque 24](#_1opuj5n)

[Detección de ataques 25](#_48pi1tg)

[5. Táctica de Testeabilidad 25](#_2nusc19)

[Input / output 25](#_1302m92)

[Grabar y reproducir información 25](#_3mzq4wv)

[Separar interfaz de implementación 25](#_2250f4o)

[Hacer interfaces para testing (access routines / interfaces) 25](#_haapch)

[Monitoreo interno 25](#_319y80a)

[Built-in monitors 25](#_1gf8i83)

[6. Tácticas de usabilidad 25](#_40ew0vw)

[Tiempo de ejecución 25](#_2fk6b3p)

[Mantener el modelo de la tarea 25](#_upglbi)

[Mantener el modelo del usuario 25](#_3ep43zb)

[Mantener el modelo del sistema 25](#_1tuee74)

[Tiempo de diseño 25](#_4du1wux)

[Separar la interfaz gráfica de la implementación 25](#_2szc72q)

[Soporte al usuario 26](#_184mhaj)

[7. Tácticas de portabilidad 26](#_3s49zyc)

[Máquina virtual 26](#_279ka65)

[Interoperabilidad 26](#_meukdy)

[Incluír una capa de interoperabilidad 26](#_36ei31r)

[Utilizar servicios web, API, mediante un lenguaje como SQL 26](#_1ljsd9k)

[Patrones de Arquitectura 27](#_45jfvxd)

[Estilo –Capas Lógicas 27](#_2koq656)

[Estilo –Tiers (Capas físicas) 29](#_zu0gcz)

[Publicador/Subscriptor 30](#_3jtnz0s)

# Capítulo 1 y 2: Introducción

## Definición de Vista

Una vista es la representación de un conjunto coherente de elementos arquitectónicos y sus relaciones. Una vista es una representación de una estructura.

## Definición de Arquitectura de Software

La organización fundamental de un sistema representado por sus componentes, las relaciones entre ellos y con su contexto, y los principios que guían su diseño y evolución.

La arquitectura de software abarca el conjunto de decisiones significativas sobre la organización de un sistema de software incluyendo:

•Elección de elementos estructurales y sus interfaces, los cuales componen al sistema.  
•Comportamiento tal como se especifica en las colaboraciones entre dichos elementos   
•Composición de estos elementos estructurales y su comportamiento en sub-sistemas más grandes.  
•Estilo arquitectónico que guía esta organización.  
•La arquitectura de software también implica funcionalidad, usabilidad, flexibilidad, eficiencia, reutilización, comprensibilidad, limitantes económicos y tecnológicos, trade-offsy cuestiones estéticas

Otra definición:  
La arquitectura de software de un programa o sistema de computación es la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades externamente visibles de esos elementos y las relaciones entre ellos.

## Importancia de la Arquitectura

Existen 3 razones fundamentales que determinan la importancia de la arquitectura de software:

•Comunicación entre los accionistas.

•Decisiones tempranas de diseño.

•Representación transferible de un sistema

* Accionistas: Clientes, Usuarios, Gerente de proyecto, Desarrollador, Tester, etc.
* Cada uno de los accionistas están interesados en características diferentes del sistema que son afectadas por la arquitectura.
* La arquitectura provee un lenguaje común en el cual cada asunto o problema puede ser expresado, negociado y resuelto.

•Define restricciones sobre la implementación.

•Inhibe o posibilita un atributo de calidad del sistema.

•Permite predecir la calidad del sistema.

•Facilita el razonamiento y gestión del cambio.

•Permite estimaciones de costo y presupuesto más precisos  
•Líneas de productos comparten una misma arquitectura base.

•Facilita la construcción de software a partir de grandes elementos desarrollados externamente.

•Permite el desarrollo basado en “templates”.

•Puede ser la base para el entrenamiento de nuevos miembros de un proyecto.

•Organizar el desarrollo

•Promover el reuso

## Tipos de Arquitectura

### Estructura de Módulo

*Descomposición*

Muestra los módulos con una relación de “es un submódulo de”. Ayuda a comprender módulos complejos descomponiéndolos en módulos más simples y con menos responsabilidades funcionales.

*Usos*

Muestra los módulos con una relación de “usa”. Ayuda a comprender cómo los módulos se relacionan entre sí desde el punto de vista de que la operación de uno depende de la presencia de un módulo correcto del cual éste depende.

*En Capas*

Cuando las relaciones de “usa” son controladas de manera particular, se desprende un sistema de capas, en donde se agrupa un set coherente de funcionalidad.

*Clase o Generalización*Muestra las clases, con sus herencias. En base a la estructura de clases, se puede considerar el reúso y la agregación incremental de funcionalidad.

### Estructura de Componentes y Conectores

Son ortogonales a las estructuras de módulos. Tienen como relación el “attachment”.

*Proceso*

Muestra los procesos o threads que se conectan entre sí para comunicación, sincronización y/o exclusión.

*Concurrencia*

Muestra los “threads lógicos” mediante una estructura que permite tempranamente detectar las ocurrencias de eventos asociados a la ejecución concurrente. Además permite planificar la ejecución paralela e identificar las ocasiones en las que habrá que manejar la contención.

*Datos compartidos o Repositorio*

Se ocupa de ilustrar como se crea, almacena y accede a datos persistentes. Es particularmente útil cuando se estructura el sistema de manera de usar varios repositorios. Muestra también como los elementos del software de tiempo de ejecución consumen y producen los datos, para así manejar la performance e integridad.

*Cliente-Servidor*Los componentes de esta estructura son clientes y servidores y los conectores son los protocolos y mensajes necesarios para que el sistema funcione. Especialmente útil para planificar la distribución física y balanceo de cargas.

### Estructura de Asignación

*Deployment*

Muestra como el software es asignado a los elementos de hardware que los procesan y comunican. Los elementos son software, entidades de hardware y vías de comunicación.

*Implementación*

Muestra como los elementos de software (usualmente módulos) son mapeados a la estructura de archivos en el sistema en el entorno de desarrollo, integración o configuración.

*Asignación de trabajo*Muestra como se le asigna la responsabilidad de implementar e integrar los módulos al equipo de desarrollo apropiado.

### ¿Qué estructura usar?

Seguramente no todas, por más que todas estén presentes en el sistema que se desea desarrollar. La recomendación que se hace es elegir los aspectos más relevantes y específicos del sistema que se está desarrollando, documentando aquellos aspectos del sistema que sean de vital importancia para entender el desarrollo y funcionamiento del mismo.

Se introduce también la aproximación 4+1, que contiene las siguientes vistas:

Lógica (Módulos): Los elementos son las abstracciones claves, que son manifestados en el mundo orientado a objetos como objetos o clases de objetos.

Procesos (Componente-Conector): Muestra la concurrencia y distribución de funcionalidad.

Desarrollo (Asignación): Muestra la organización de módulos de software, librerías, subsistemas y unidades de desarrollo. Mapea el software al ambiente de desarrollo.

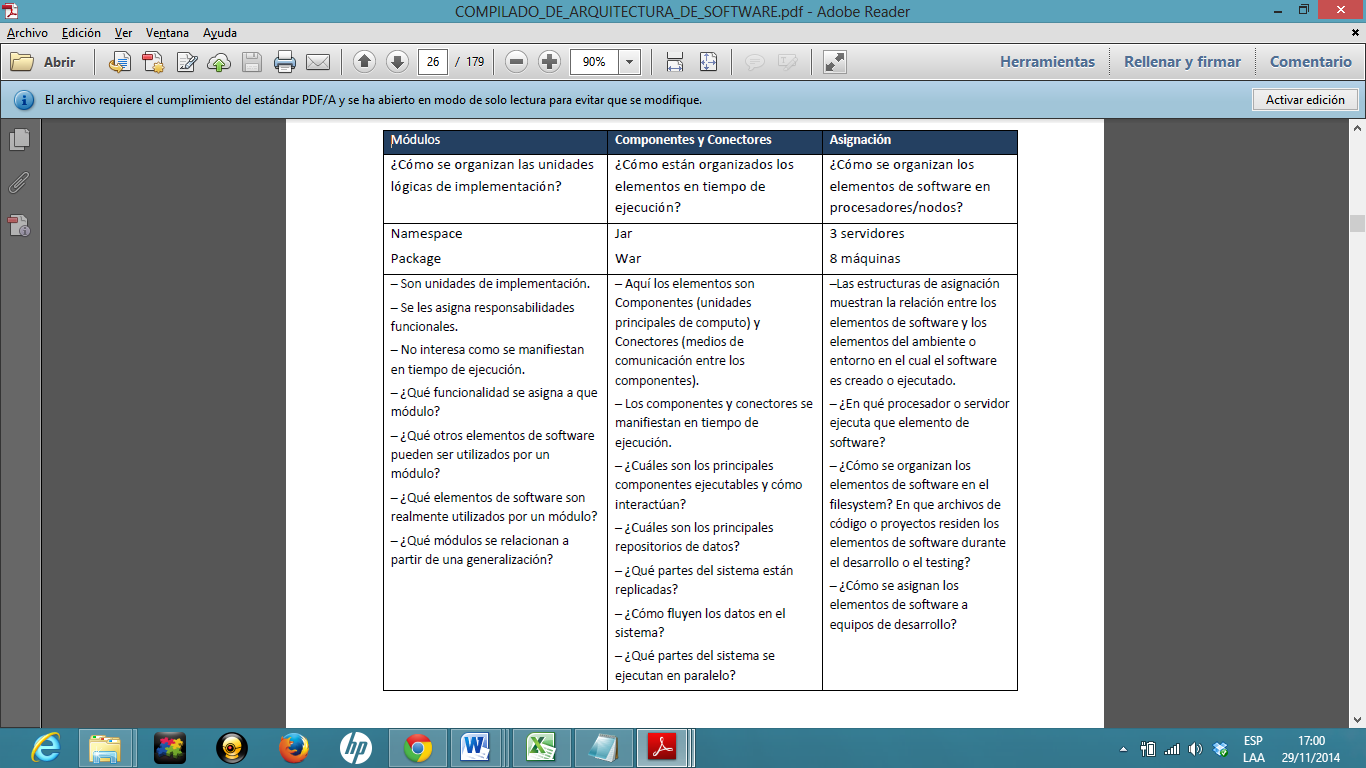
Física (Asignación): Mapea demás elementos en nodos de proceso y comunicación. Es llamada también vista de Deploy.

### Resumen

1. Estructuras de Módulos: Los elementos son módulos, unidades de implementación. Los módulos representan una forma basada en código para considerar el sistema. Son áreas asignadas de responsabilidad funcional.
2. Estructuras de Componentes & Conectores: Aquí los elementos son componentes en tiempo de ejecución (unidades principales de computación) y conectores (vehículos de comunicación entre componentes).
3. Estructuras de Asignación: Muestran la relación entre elementos de software y los elementos en uno o más entornos externos en los cuales el software es creado y ejecutado.

Estas estructuras responden a 3 preguntas que el diseño arquitectónico involucra:

* ¿Cómo debe ser estructurado el sistema como un conjunto de unidades de código (módulos)?
* ¿Cómo debe ser estructurado el sistema como un conjunto de elementos que tienen comportamiento en tiempo de ejecución (componentes) e interacciones (conectores)?
* ¿Cómo debe el sistema relacionarse con estructuras que no son software en su entorno?



# Capítulo 18: Documentación

*“Documentar la arquitectura es una cuestión de documentar puntos de vista relevantes, y después agregar la documentación que se aplica a más de una vista”*

## ¿Cuáles son los puntos de vista relevantes?

Depende de tus objetivos. La documentación de arquitectura puede servir para muchos propósitos: una declaración de misión para los ejecutores, una base para el análisis, la especificación para la generación automática de código, el punto de partida para la comprensión del sistema y la recuperación de activos, o el modelo para la planificación del proyecto.

*Diferentes puntos de vista también se exponen los diferentes atributos de calidad en diferentes grados.* Por lo tanto, los atributos de calidad que son de mayor interés para usted y los demás interesados en el desarrollo del sistema afectará a la elección de qué puntos de vista serán usados en el documento.

Por ejemplo, una vista en capas le dirá acerca de la portabilidad de su sistema, una vista de despliegue le permitirá razonar sobre el rendimiento del sistema y la fiabilidad, y así sucesivamente.

Diferentes puntos de vista dan apoyo diferentes objetivos y usos. Esto es fundamental, por eso que no abogan por una visión particular o una colección de puntos de vista. Los puntos de vista del documento deben depender de los usos que usted espera hacer de la documentación. Diferentes puntos de vista pondrá de relieve los diferentes elementos del sistema y / o relaciones.

### La documentación para obtener una vista contiene

- Una presentación principal, generalmente gráfica, que muestra los principales elementos y las relaciones de la vista

- Un catálogo de los elementos que explica y define los elementos que se muestran en la vista y las listas de sus propiedades

- Una especificación de las interfaces de los elementos y el comportamiento

- Una guía que explica la variabilidad cualquier mecanismo de construcción disponible para la adaptación de la arquitectura

- Justificación y diseño de la información

### La documentación que se aplica a todos los puntos de vista contiene

- Una introducción a todo el paquete, incluyendo una guía de lectura que ayuda a un actor encontrar un pedazo deseado de información de forma rápida

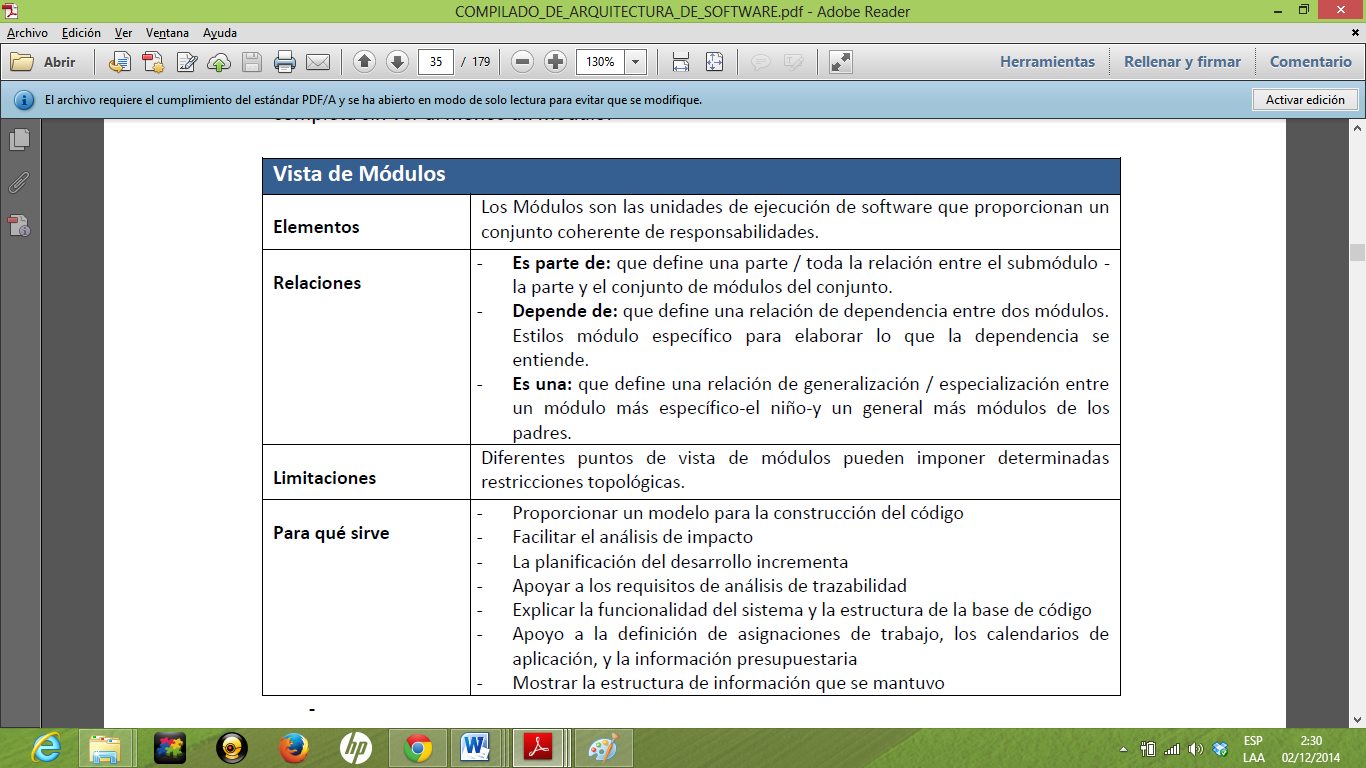
- La información que describe cómo los puntos de vista se relacionan entre sí, y para el sistema en su conjunto

- Las restricciones y los fundamentos de la arquitectura global

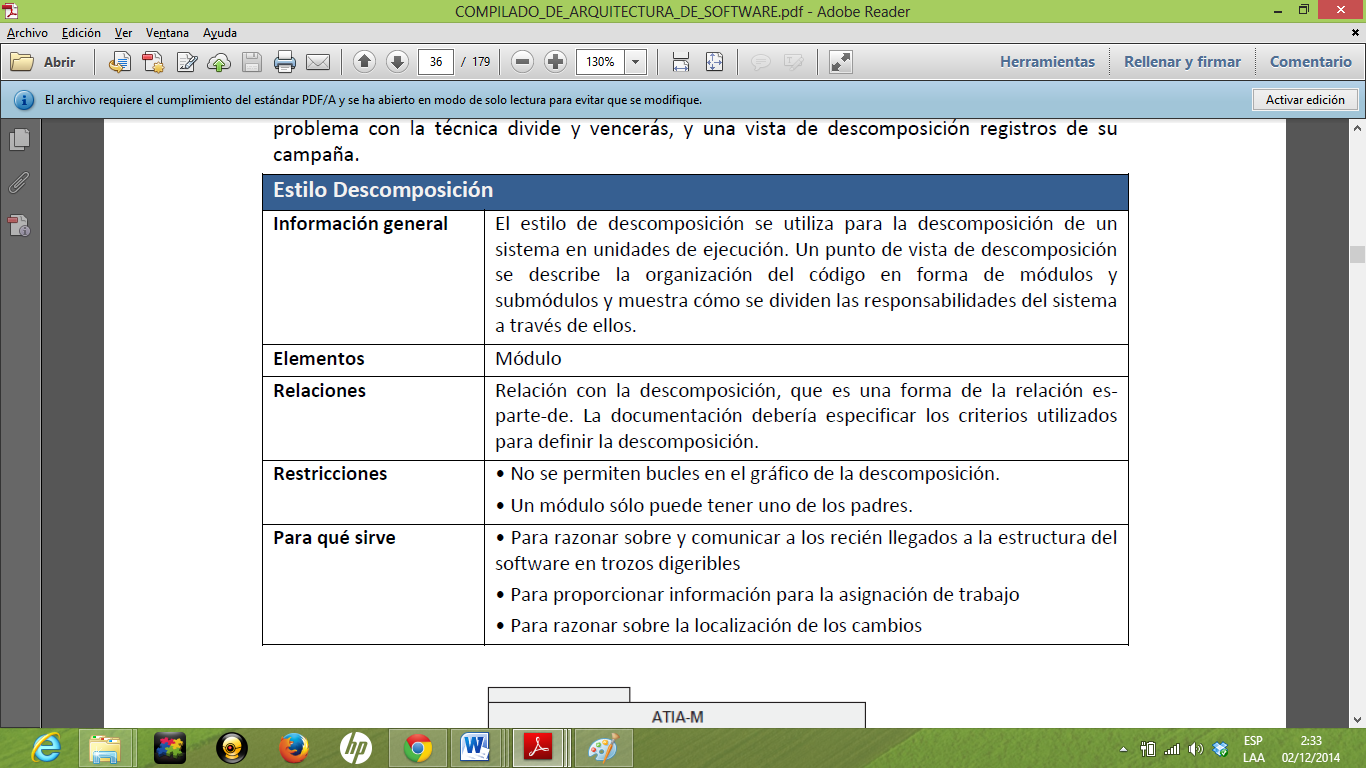
- La información de gestión que sean necesarios para mantener con eficacia el paquete completo

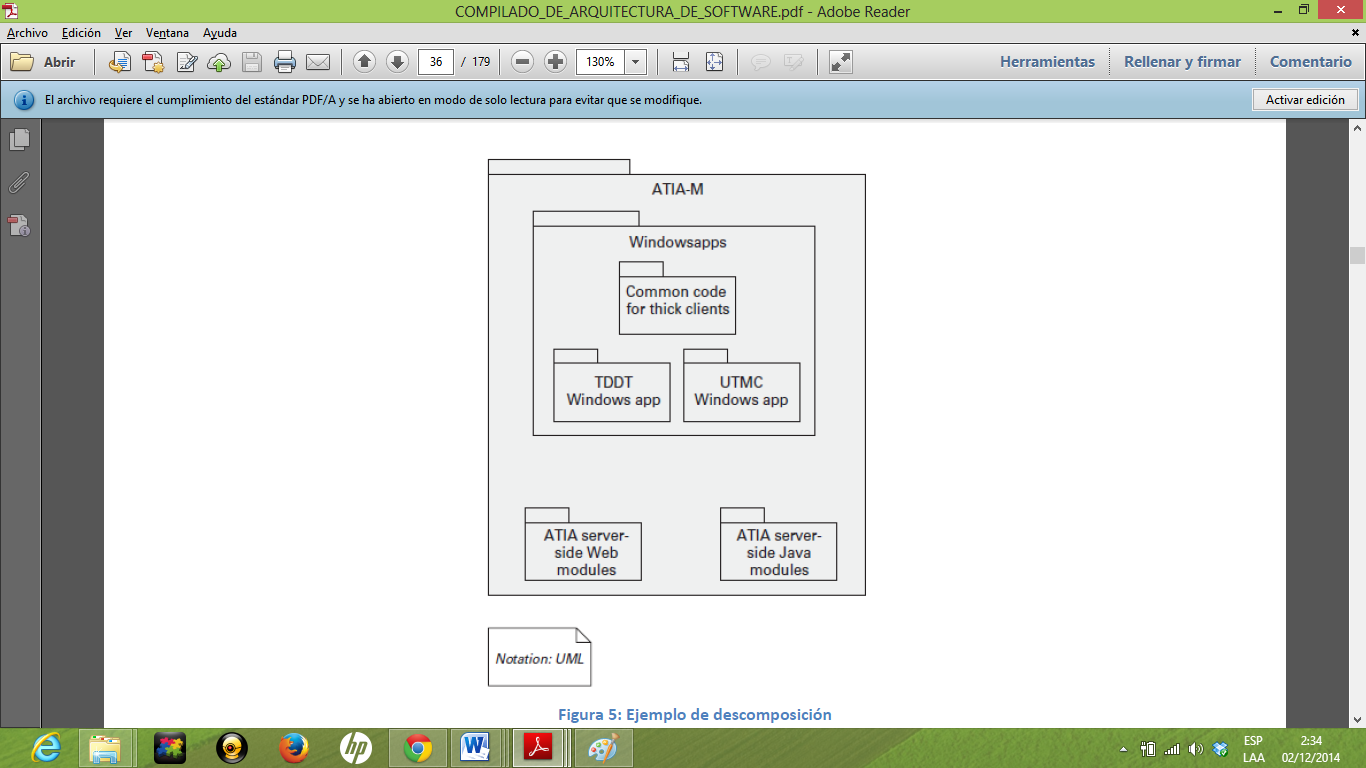
## Principales Vistas

### Vistas de Módulos (Module View)

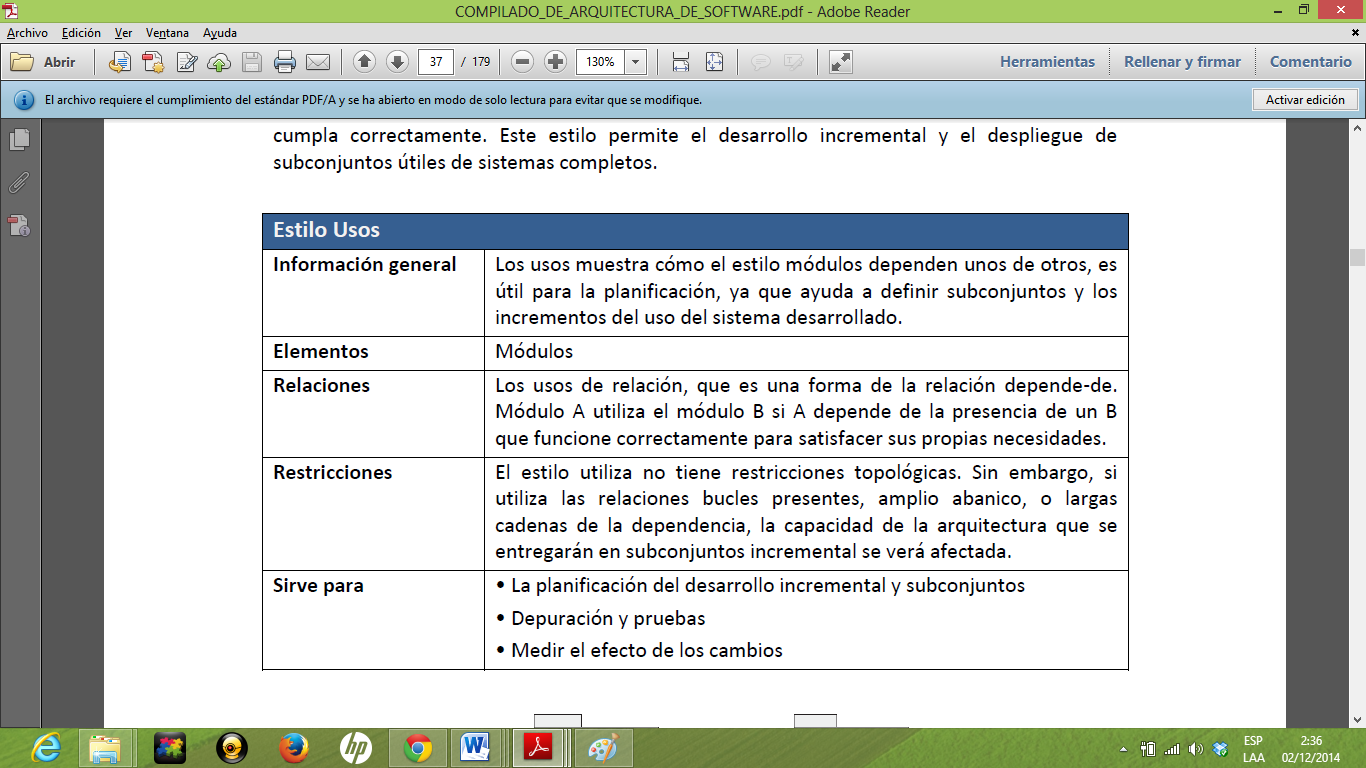


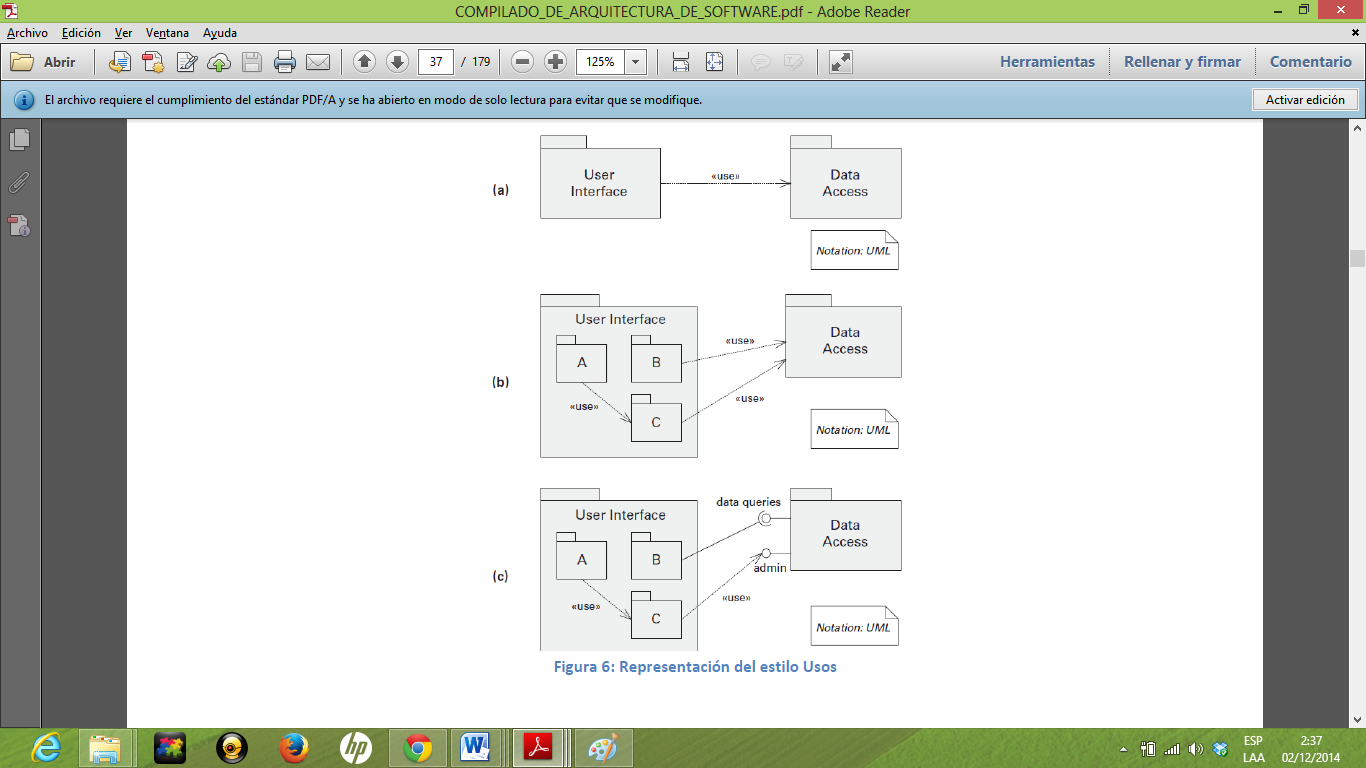
#### Estilo Descomposición



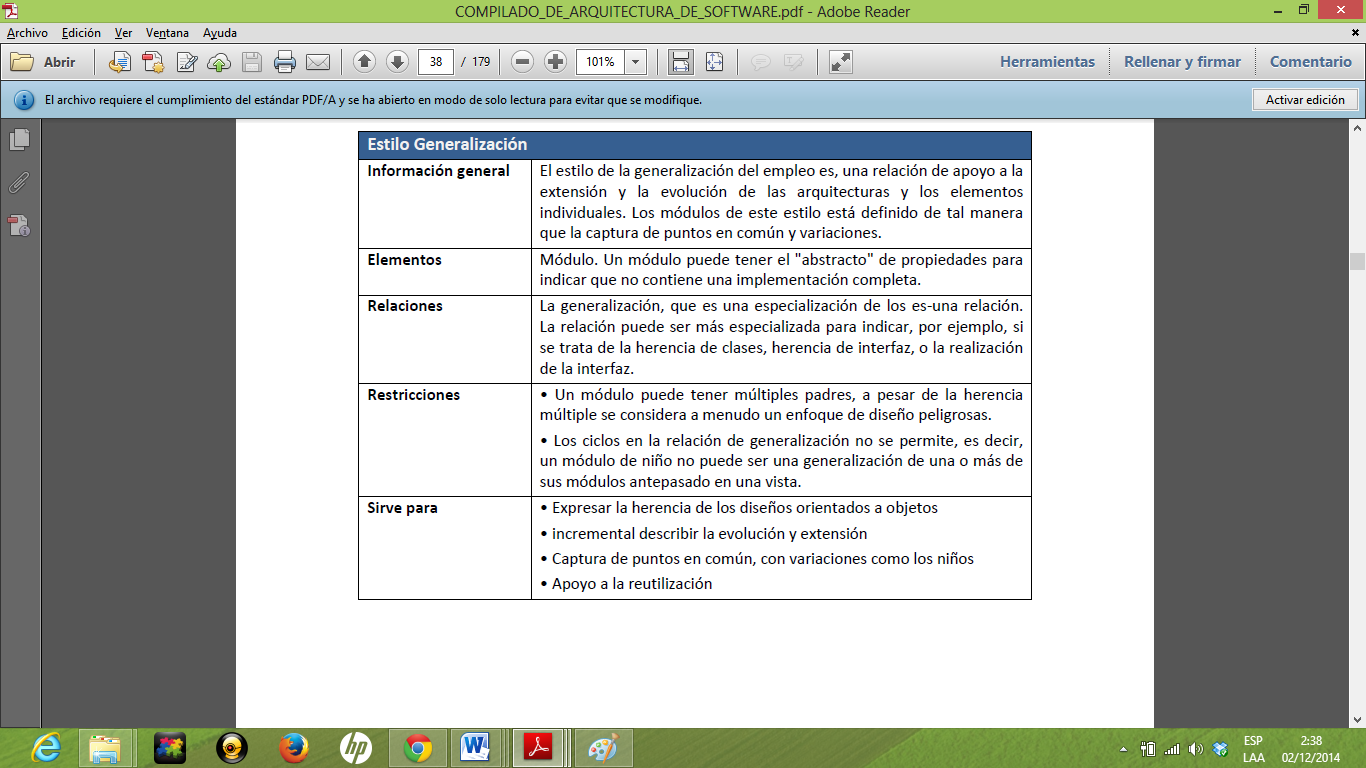


#### Estilo Usos

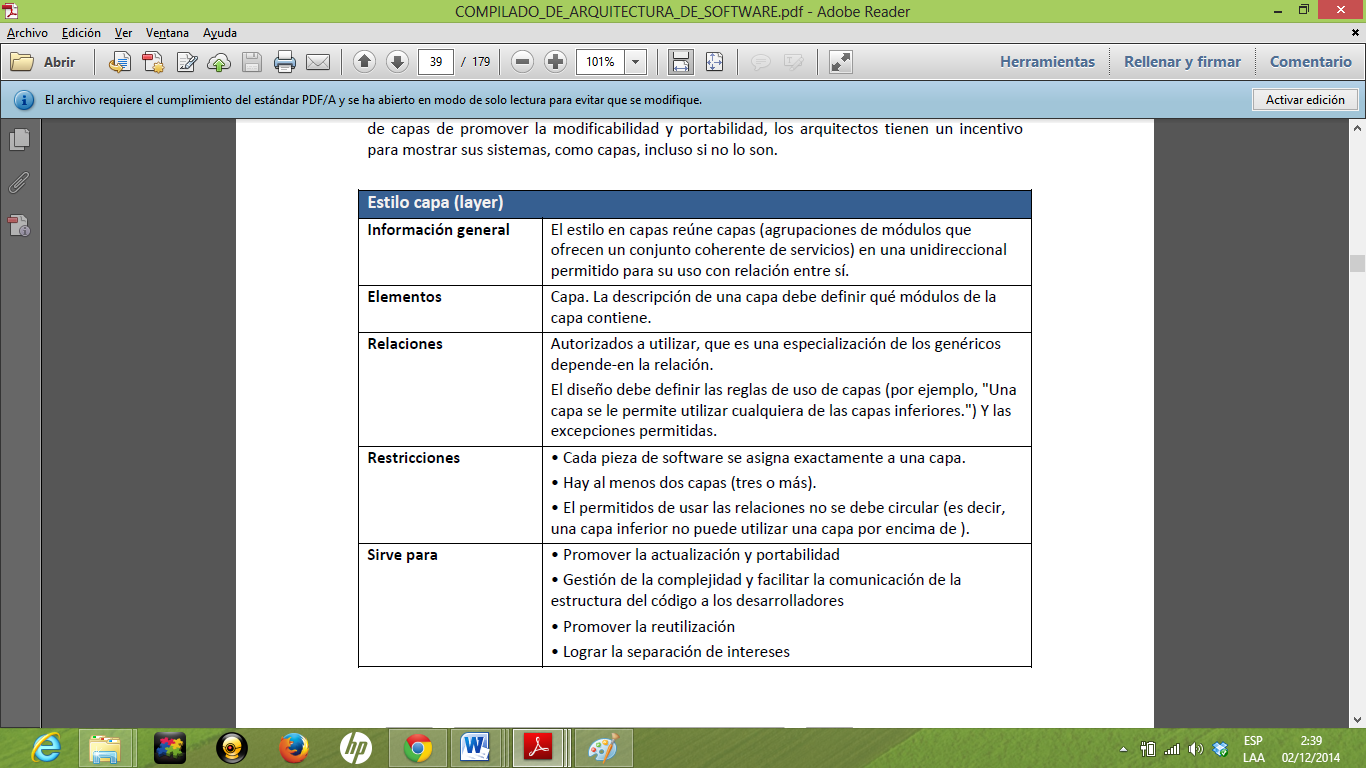


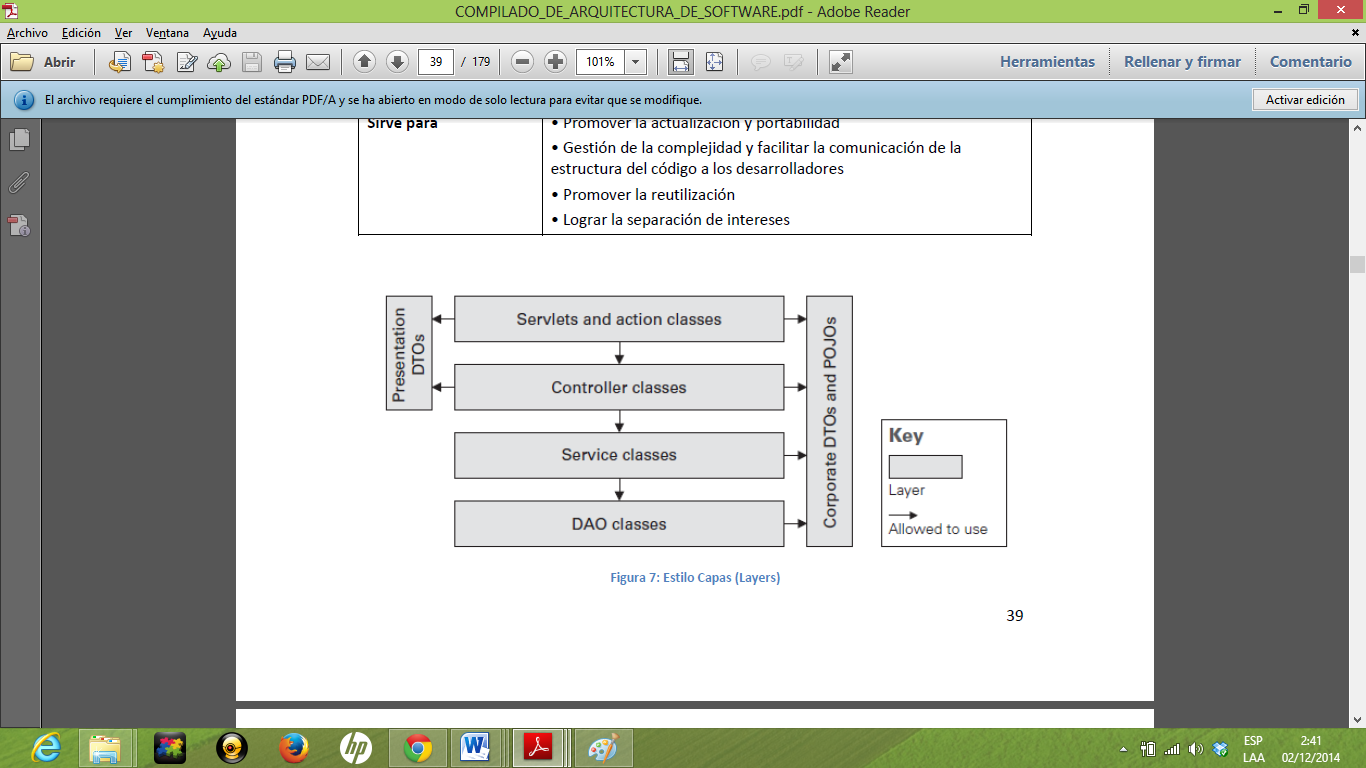


#### Estilo generalización

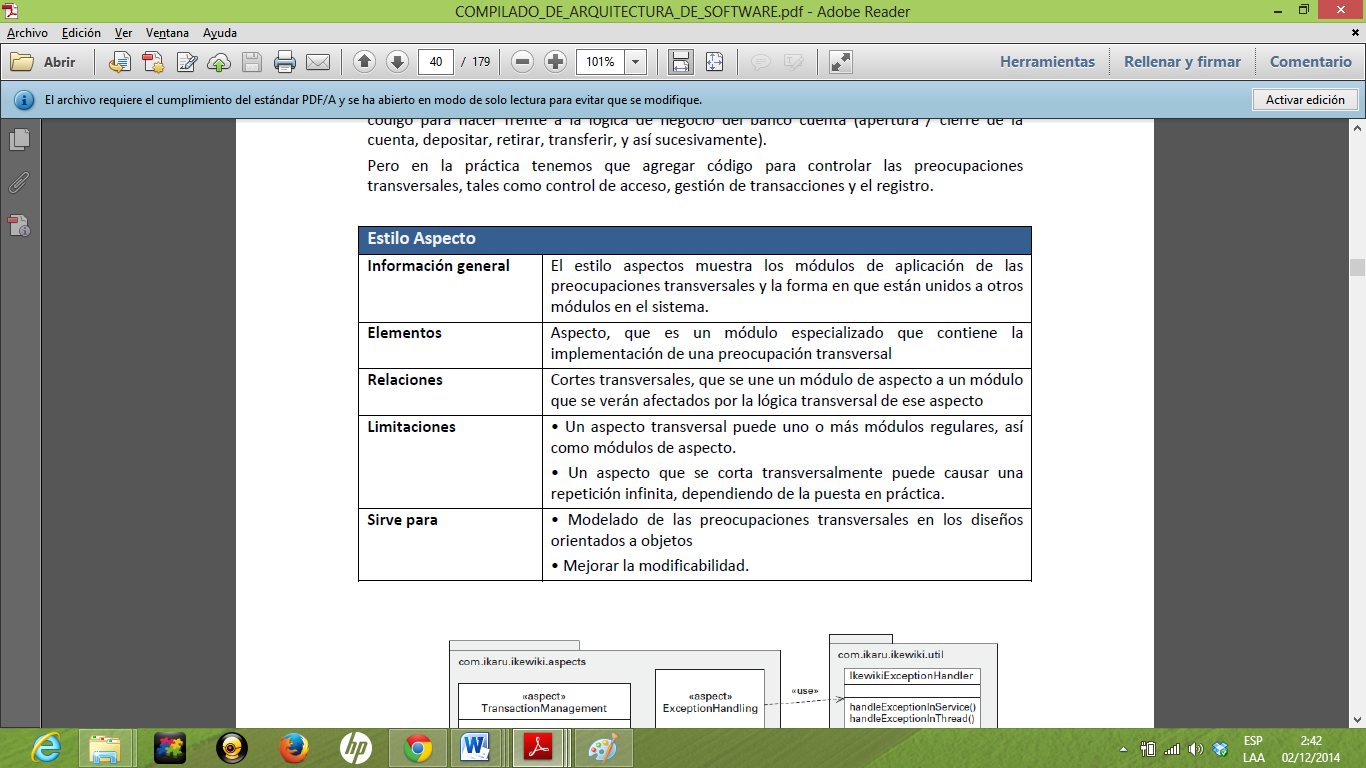


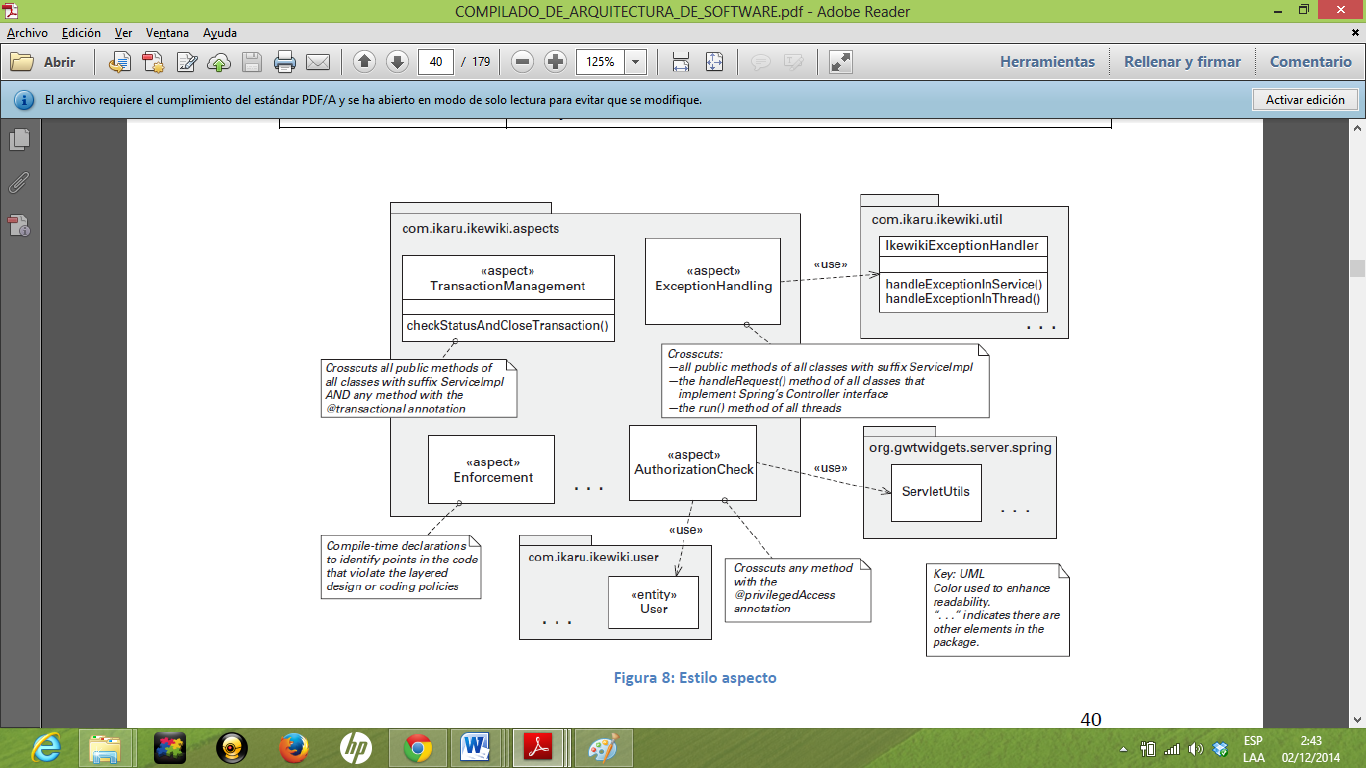
#### Estilo capas



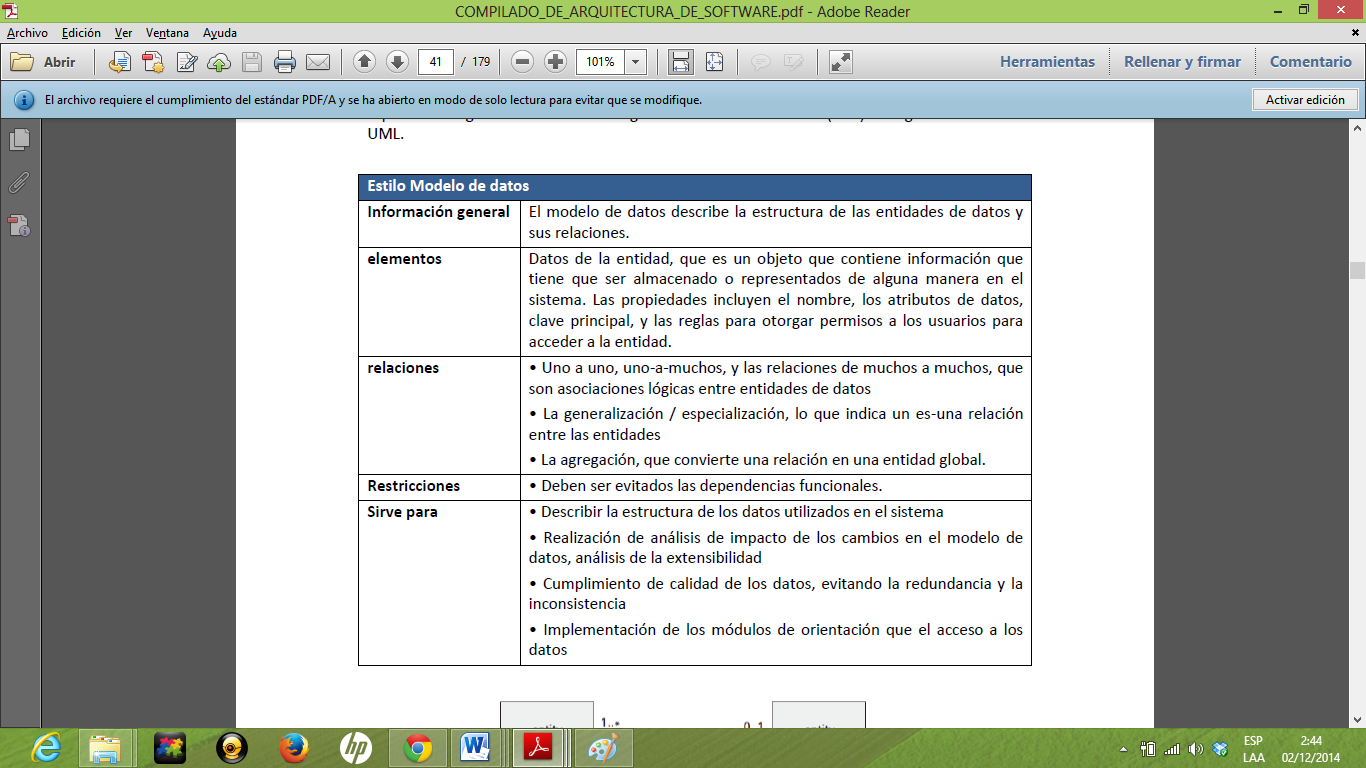


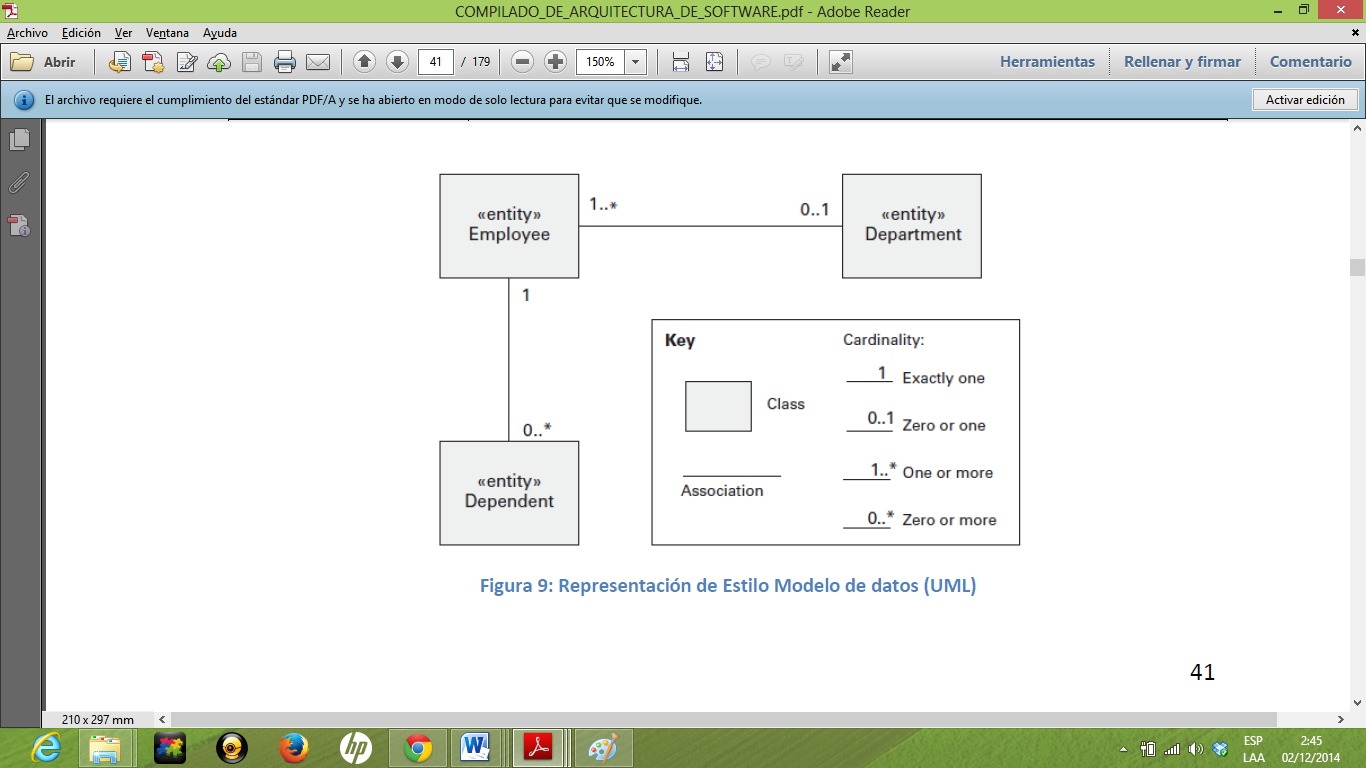
#### Estilo aspecto





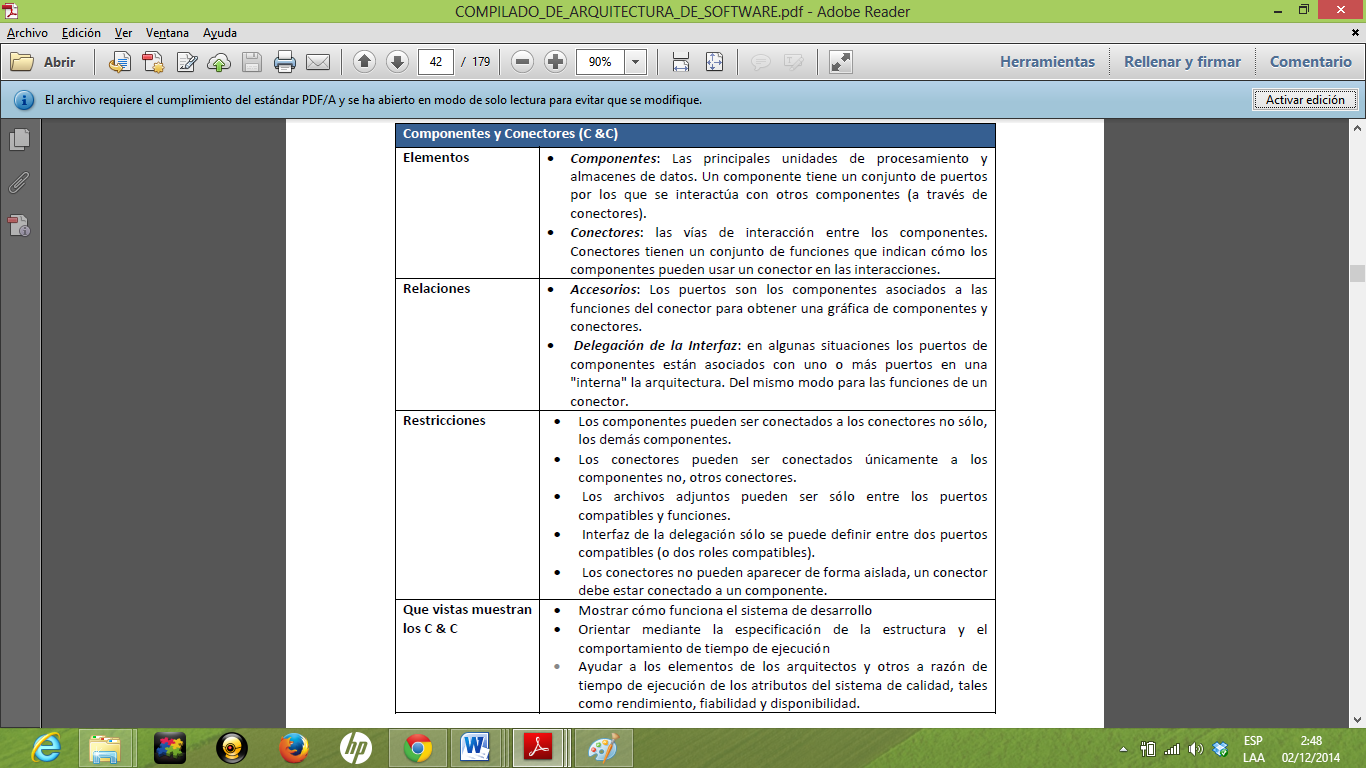
#### Estilo modelo de datos





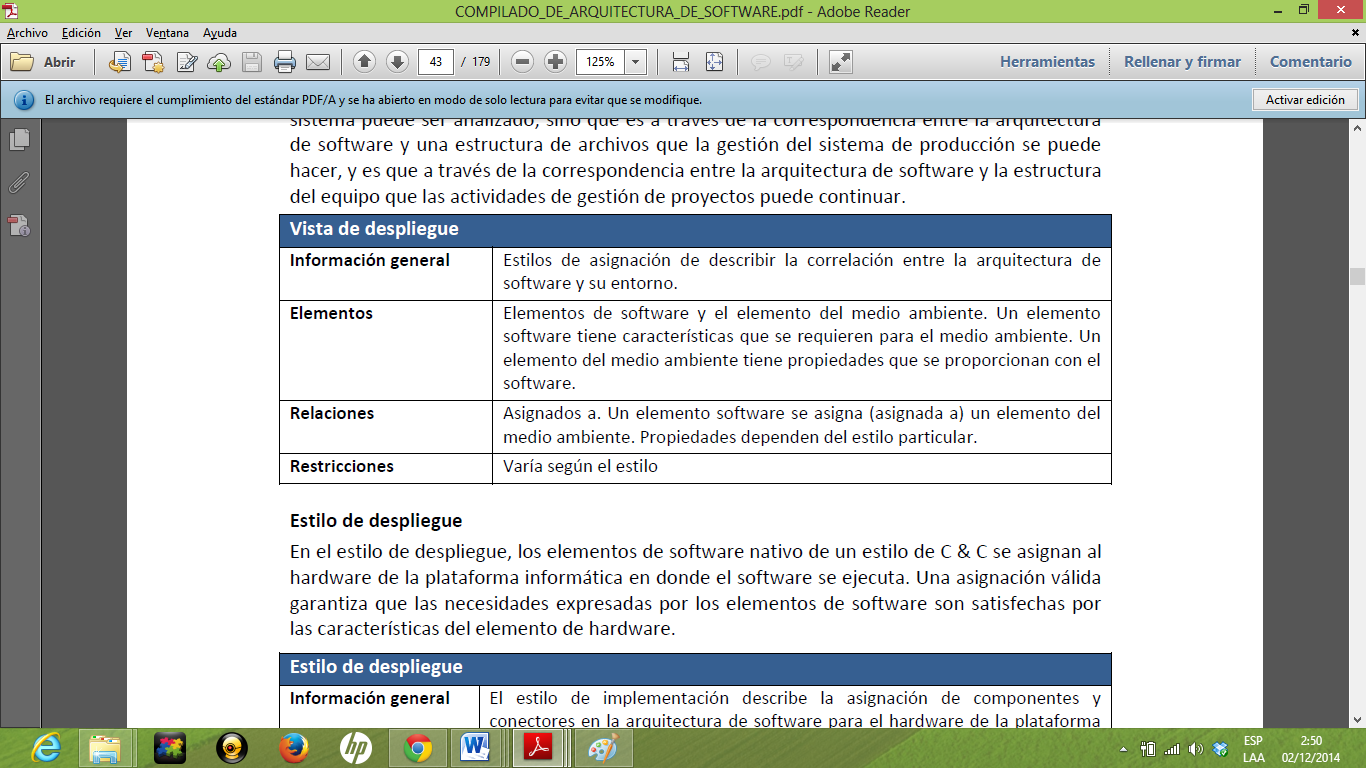
### Vistas de Componentes y Conectores

*Un punto de vista C & C se muestran los elementos que tienen alguna presencia en tiempo de ejecución, tales como procesos, objetos, clientes, servidores y almacenamiento de datos*. Estos elementos se denominan componentes. Además las vistas de los componentes y conectores incluyen como elementos de las vías de interacción, tales como enlaces y protocolos de comunicación, los flujos de información y el acceso al almacenamiento compartido. Estas interacciones se presentan como conectores en los puntos de vista de C & C.



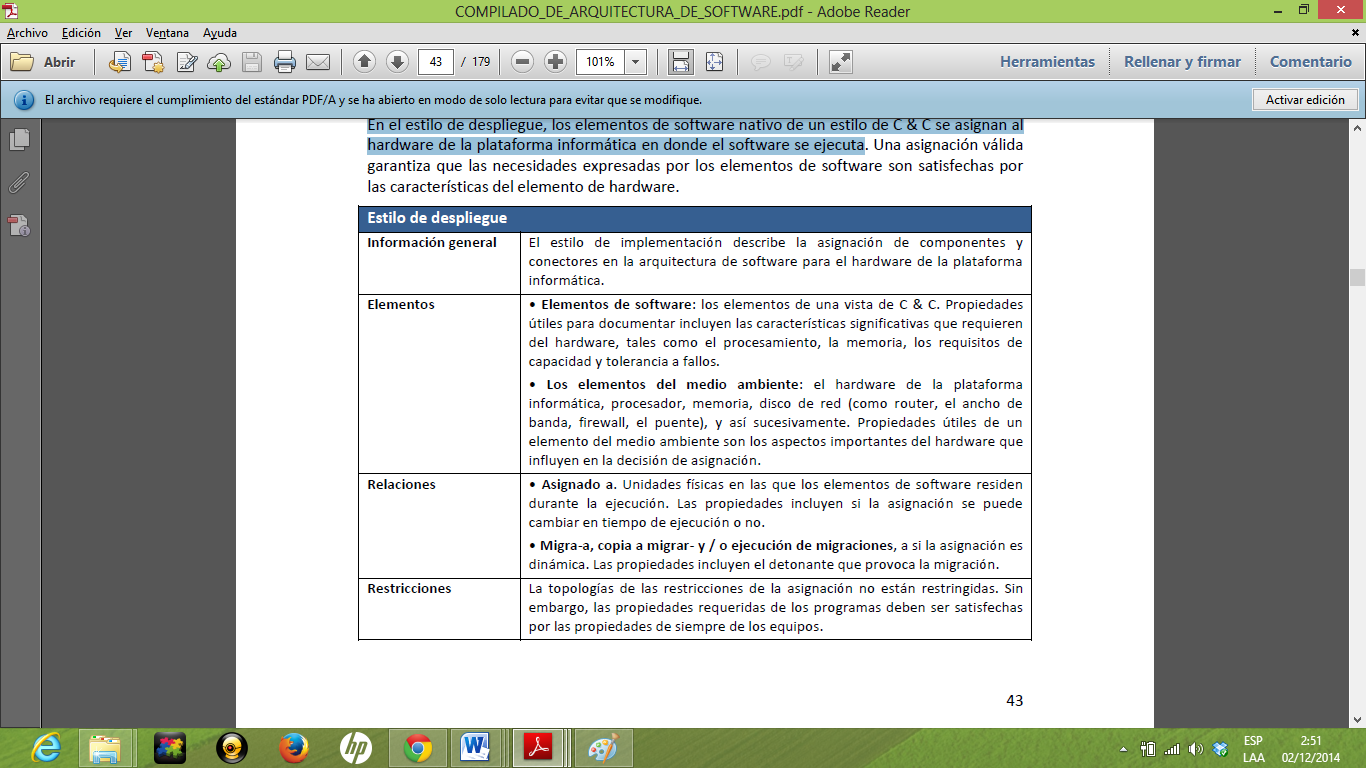
### Vista de despliegue

Elementos de software en una arquitectura de software que interactúan con elementos que no son de software en un entorno en el que el software es desarrollado, desplegado y ejecutado.



#### Estilo de despliegue

En el estilo de despliegue, los elementos de software nativo de un estilo de C & C se asignan al hardware de la plataforma informática en donde el software se ejecuta



# Capítulo 4 a 11: Atributos de Calidad

Los atributos de calidad al igual que los RFs se afectan a las distintas partes de un sistema.

Existe dependencia entre los atributos de calidad por lo que es imposible lograrlos todos para un sistema, es necesario priorizar.

## Confiabilidad

* + *Tolerancia a fallas (Disponibilidad)*
    - Capacidad de mantener el nivel de performance especificado en caso de fallas en el software
    - Trata con las fallas del sistema y sus consecuencias asociadas. (Falla: es observable por el usuario, Defecto: es la causa de la falla. Puede recuperarse sin que se haga observable)
  + *Capacidad de recuperación (Recoverability)*
    - Capacidad e restablecer el nivel de performance y los datos involucrados

## Modificabilidad (Escalabilidad)

* + Es la capacidad del sistema de permitir realizar cambios (anticipados o no anticipados) con bajo impacto.
  + Que puede cambiar
    - Código / funcionalidad.
    - Plataforma (Hardware, SO, Middleware): Portabilidad.
    - Atributos de calidad (eficiencia, seguridad, etc).
    - Capacidad (usuarios, procesamiento): Escalabilidad.

## Eficiencia

* + Es la cualidad que refleja la habilidad del sistema de lograr los requerimientos de performance al tiempo que minimiza los recursos de su ambiente computacional (R. Taylor).
  + Refiere al tiempo requerido para responder a determinados eventos o al número de eventos procesados en un intervalo dado de tiempo.
  + Subatributos
    - *Latencia*: Especifica el tiempo requerido por parte del sistema para responder al algún estimulo.
    - *Capacidad*: Especifica cuantos usuarios, terminales, nodos, registros, etc. debe soportar el sistema manteniendo las restricciones de tiempo.
    - *Throughput*: Especifica cuantos eventos puede responder el sistema en un intervalo dado de tiempo

## Seguridad

* + Es la habilidad del sistema de resistir intentos de uso no autorizados mientras que se continúa ofreciendo los servicios a los usuarios legítimos.
  + Ataque o amenaza: Intento de violar la seguridad
  + *No repudiación*: Implica que una transacción no pueda negarse por ninguna de las partes involucradas.
  + *Confidencialidad*: Datos o servicios están protegidos del acceso no autorizado.
  + *Integridad*: Datos o el servicio se entregan según lo acordado.
  + *Aseguramiento*: Implica que se garantice la identidad de quienes participan en una transacción.
  + *Disponibilidad*: Un sistema esta disponible para su uso legitimo.
  + *Auditoria*: El sistema lleva registro de sus actividades lo que permite reconstruir una transacción.

## Usabilidad

* + El esfuerzo requerido por parte del usuario para realizar sus tareas con el sistema.
  + El tipo de asistencia que brinda el sistema a los usuarios.

## Testeabilidad

* + Facilidad con la cual se puede demostrar las fallas de un sistema de software a partir de pruebas (testing)
  + Testing representa el 40% del costo de desarrollo de un sistema.
  + Sistema testeable: debe ser posible controlar el estado interno y entradas de cada componente y observar las salidas/resultados.
  + Suelen utilizarse herramientas, librerías externas o incluso metodologías especificas para realizar las pruebas de software.

## Portabilidad

* + Capacidad para transferir un producto de software de un ambiente a otro.
    - El ambiente puede ser organizacional, hardware o software.

## Interoperabilidad

La interoperabilidad es el grado en que dos o más sistemas pueden intercambiar eficazmente información significativa a través de interfaces en un contexto particular.

La definición incluye no sólo tener la posibilidad de intercambiar datos (interoperabilidad sintáctica), sino también tener la capacidad de interpretar correctamente los datos que se intercambian (interoperabilidad semántica).

Un sistema no puede ser interoperable de forma aislada. Cualquier discusión sobre la interoperabilidad del sistema tiene que identificar con quién, con qué, y en qué circunstancias, de ahí la necesidad de incluir el contexto.

La Interoperabilidad se ve afectada por los sistemas previstos para interoperar. Si ya conocemos las interfaces de sistemas externos con los que nuestro sistema debe interoperar, entonces podemos diseñar ese conocimiento en el sistema. O podemos diseñar nuestro sistema para interoperar de forma más genérica , de modo que la identidad y los servicios que ofrece otro sistema se pueden unir más adelante en el ciclo de vida, en tiempo de compilación o en tiempo de ejecución .

Hay varios marcos que caracterizan a la interoperabilidad, los cuales parecen definir niveles de interoperabilidad

* El nivel más bajo significa sistemas que no comparten datos en absoluto, o no lo hacen con algún éxito.
* El nivel más alto significa sistemas que funcionan a la perfección, nunca cometió ningún error de interpretación de comunicaciones, y comparten el mismo modelo semántico subyacente del mundo en el que trabajan.

## Disponibilidad

Habilidad del sistema de enmascarar o reparar fallas de forma que el periodo acumulado de interrupción no exceda un valor requerido sobre un intervalo de tiempo especificado

Un fallo puede dar por defectos internos o externos. Un defecto puede ser un fallo si no se corrige o enmascara. Un fallo es observable por el usuario, un defecto no. Refiere a la tolerancia a fallas

ESCENARIO GENERAL

* Fuente de estimulo

Fallo o defecto

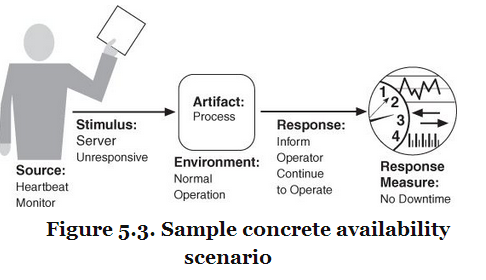
* Estimulo
* De omision: un componente que no responde
* De choque: un componente repetidamente no responde
* De sincronización: un componente responde antes o después de lo esperado
* De respuesta: Da un valor incorrecto
* Artefacto

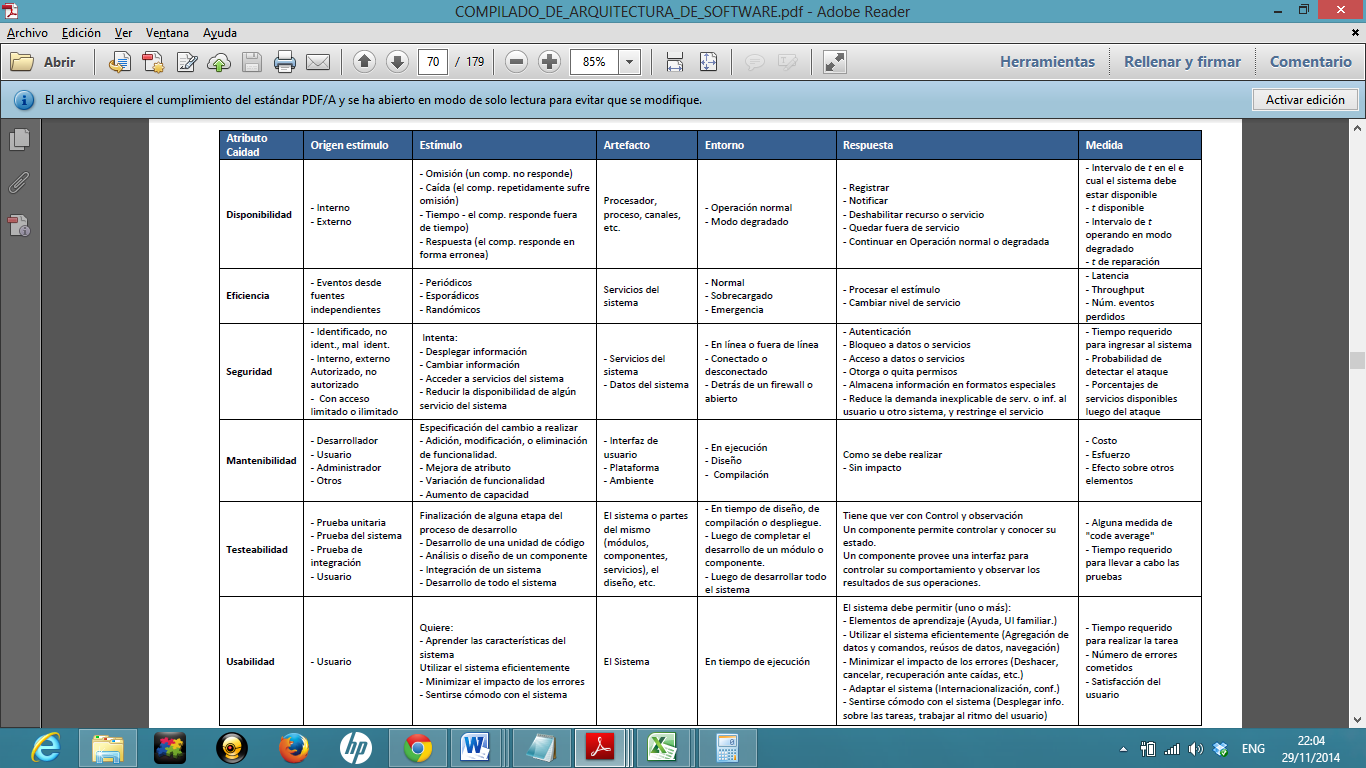
Recurso que se desea esté disponible

* Medio ambiente

Sistema operando bajo restricciones por ocurrencia de defectos/fallos anteriores, donde se puede dar la baja. Sistema funcionando full donde se puede bajar el rendimiento por fallos/defectos

* Respuesta
* Registrar la falla
* Notificar a usuario u otros sistemas
* Cambiar el modo de operación a menor capacidad o funcionalidades
* Apagar sistemas externos
* Continuar operando normalmente
* Medida de respuesta
* % de disponibilidad
* Tiempo de reparacion
* Horas que debe estar disponible
* Horas en las que debe estar disponible





## ¿Cómo se relacionan los atributos de calidad?

# Tácticas

Son decisiones comunes de arquitectura orientadas a lograr o satisfacer determinado atributo de calidad.

Para resolver los requerimientos no funcionales se utilizan estilos, a los cuales se suman mecanismos o tácticas.

Una táctica se puede refinar en varias tácticas.

# 1. Tácticas de disponibilidad

Objetivo: Que un error no se transforme en una falla. Que el error se arregle o se enmascare.

Tácticas

### Detección de errores

#### Ping echo

Hacer ping y esperar que el componente conteste. Si no hay respuesta, se asume que el componente falló.

#### Latencia

El componente envía cada determinado tiempo una señal de que está operativo. Si no se recibe la señal durante mucho tiempo se asume que falló.

#### Excepciones

## Recuperación de errores – preparación y arreglo

### Votación

Varios procesos corriendo en procesadores redundantes y todos toman la misma entrada, pero con algoritmos diferentes, desarrollado por personas diferentes. Todos contestan y se usan algoritmos para ver qué resultado tomar.

### Redundancia activa

Todos reciben la entrada, la procesan y envían la respuesta. Se toma la respuesta del que conteste más rápido. De esta forma todos los componentes se mantienen sincronizados.

### Redundancia pasiva

El componente primario recibe la entrada, la procesa y envía la respuesta. Luego le avisa a los componentes redundantes para que se sincronicen. Si el componente primario falla, se analiza el grado de sincronización de los redundantes.

### Repuesto

Se mantiene un componente de repuesto para sustituir al que falle. Al ocurrir un fallo, el componente debe ser sincronizado con la última versión consistente del componente primario con sus configuraciones y estados.

## Recuperación de errores – reintroducción de componentes

### Modo sombra

El componente que falla se mantiene funcionando en “modo sombra” mientras se recupera. Imita su comportamiento normal durante un breve período.

### Resincronización de estados

Mecanismos para sincronizar los componentes redundantes con el componente primario.

### Checkpoint/rollback

Se crean checkpoints a lo largo de la ejecución para poder volver a un estado consistente (rollback) en caso de que el componente falle.

## Prevención de errores

### Remoción del servicio

Se retira el componente antes de que falle. Si el cambio es automático, la arquitectura debe soportarlo; si el cambio es manual, el sistema debe soportarlo.

### Transacciones

Se trata de empaquetar una serie de pasos secuenciales para deshacer todo el conjunto de una vez, en caso de falla.

### Monitoreo de procesos

Cuando se detecta un error, el monitor mata el proceso defectuoso y lo reinicializa en un estado que pueda mantener el sistema funcional

# 2. Tácticas de modificabilidad

Objetivo: hacer cambios, testearlos y generarlos sin costos de tiempo y presupuesto

## Localización de cambios

### Coherencia semántica

Se trata de mantener las relaciones dentro del módulo, minimizando las dependencias hacia otros módulos para tareas similares o con el mismo significado semántico.

#### Abstraer servicios comunes

### Anticipar cambios

La diferencia entre esta táctica y la anterior es que la anterior asume que los cambios serán semánticamente coherentes. Esta táctica se centra específicamente en los cambios esperados y la minimización de los módulos a tocar, muchas veces armando módulos en torno a la funcionalidad y no a la semántica.

### Generalizar módulos

Hacer que el componente reaccione a las entradas de manera similar, para que los cambios se reflejen solo en la modificación de las entradas y no en el módulo en general.

### Limitar opciones

Se trata de limitar los cambios posibles. Por ejemplo, limitando los cambios de plataforma a una línea de plataformas en particular.

## Prevenir efecto dominó

Limitar las modificaciones a los módulos que no dependen directamente del cambio. Se quiere modificar el módulo A, que tiene relaciones con el B, hay que considerar el tipo de relación:

* Sintaxis de datos/servicios: el módulo B consume datos/servicios producidos por A, que tienen una sintaxis determinada.
* Semántica de datos/servicios: el módulo B consume datos/servicios producidos por A, que tienen una semántica determinada.
* Secuencia
  + De datos: El módulo B consume los datos producidos por A en determinada secuencia
  + De control: el módulo B ejecuta luego de A, que debe haber ejecutado con restricciones de tiempo.

### Ocultamiento de información

Se centra en ocultar la información a los demás módulos, logrando menos acoplamiento.

### Mantener interfaces existentes

Diseñar interfaces que perduren independientemente de los cambios que se puedan llegar a realizar.

### Restringir caminos de comunicación

Se centra en restringir los módulos que comparten información (reducir módulos que producen y consumen información)

### Utilizar intermediarios

Asignar intermediarios que centralicen la interacción entre 2 módulos

* Sintaxis de datos: Repositorios que convierten la sintaxis de los datos
* Sintaxis de servicios: Facade, bridge, mediator, strategy y proxy
* Identidad de interfaz: Brokers enmascara los cambios de identidad de interfaces
* Localización: servidor de nombres
* Existencia: factory asegura la creación de instancias

## Diferir tiempo de enlaces

Minimizar tiempo y costo del cambio y habilitar a no desarrolladores a hacer los cambios

### Registro en tiempo de ejecución

Operaciones plug and play para manejar la registración, ej.: suscribirse a servicios.

### Archivos de configuración

Archivos con parámetros de startup

### Polimorfismo

Para aprovechar el enlace tardío

### Reemplazo de componentes

Crear componentes intercambiables

### Adherencia a protocolos

Permitir enlaces de procesos en tiempo de ejecución

# 3. Tácticas de performance

Objetivo: generar respuestas en el tiempo requerido

## Demanda de recursos

### Reducir recursos que hacen falta para procesar un evento

#### Aumentar la eficiencia computacional

Cambiar un recurso por otro o cambiar el algoritmo que procesa el evento

#### Reducir overhead

Eliminar intermediarios o eliminar procesos de control

### Reducir número de eventos procesados

Manejar la velocidad de llegadas de eventos

Manejar la frecuencia de muestreo de datos

Controlar el uso de recursos

Manejar el tiempo de dedicación a un evento

Manejar el tamaño de colas de eventos

## Manejo de recursos

### Introducir concurrencia

Se centra en crear operaciones que procesen en paralelo

### Mantener copias de datos

Se centra en crear copias de datos para que la concurrencia no bloquee procesos

### Aumentar recursos disponibles

Se centra en mejorar los recursos: más y mejor

## Arbitraje de recursos

### Políticas de planificación

#### FIFO

#### Prioridad fija

Por importancia semántica, deadline y tiempo de aparición

#### Asignación dinámica

Round robin y deadline más próximo primero

#### Asignación estática

Se fijan las prioridades cuando el sistema está bajo

# 4. Técnicas de seguridad

Objetivo: detectar, resistir y recuperarse de ataques

## Resistir ataques

### Autenticación de usuarios

Asegura que el usuario (personal o genérico) que entra al sistema sea quien dice ser (psw, one-time psw, certificados digitales, dispositivos biométricos)

### Autorización de usuarios

Control de acceso y modificación mediante grupos, permisos al usuario, roles o listas de usuarios

### Confidencialidad de los datos

Encriptar datos y canales de comunicación

### Integridad de los datos

Entregar los datos que se esperan. Para controlar esto los datos pueden tener información redundante (CRC, MD5) encriptada con los datos o de forma independiente.

### Limitar la exposición de los datos

Diseñar la ubicación de los servicios para exponer lo menos posible los datos

### Limitar el acceso

Firewalls, proxys, dmz (zonas desmilitarizadas)

## Recuperación de ataques

### Restaurar estado

Superponer tácticas de disponibilidad para recuperar el sistema a un estado consistente

### Identificar ataque

Mantener traza de auditoría

## Detección de ataques

A través de sistema de detección de intrusos

# 5. Táctica de Testeabilidad

Objetivo: probar de manera fácil cuando se modifica el software y encontrar errores.

## Input / output

### Grabar y reproducir información

Capturar información a través de una interfaz y mostrarla

### Separar interfaz de implementación

Utilizar objetos ficticios para que devuelvan los datos que se necesitan probar, es fácil intercambiarlos. Es una técnica para comenzar desde lo más complicado

### Hacer interfaces para testing (access routines / interfaces)

Monitorear el estado de los componentes y outputs

## Monitoreo interno

### Built-in monitors

Para tomar medidas de estado, carga, capacidad, seguridad o cualquier otra información accesible a través de una interfaz.

# 6. Tácticas de usabilidad

Objetivo: que sea fácil para el usuario completar una tarea deseada y dar soporte.

## Tiempo de ejecución

### Mantener el modelo de la tarea

Determinar el contexto con el fin de que el sistema pueda tener una idea de lo que el usuario está tratando de hacer y proveerle ayuda

### Mantener el modelo del usuario

Determinar el conocimiento del usuario sobre el sistema y su comportamiento, en términos de tiempo de respuesta esperado y otros aspectos específicos a un usuario o grupo de usuarios.

### Mantener el modelo del sistema

Determina el comportamiento esperado del sistema para poder brindar feedback apropiado al usuario

## Tiempo de diseño

### Separar la interfaz gráfica de la implementación

Localizar cambios

### Soporte al usuario

# 7. Tácticas de portabilidad

Objetivo: llevar a un sistema de una organización a otra (productos modularizados y parametrizados)

## Máquina virtual

## Interoperabilidad

### Incluír una capa de interoperabilidad

### Utilizar servicios web, API, mediante un lenguaje como SQL

# Patrones de Arquitectura

## Estilo –Capas Lógicas

El patrón arquitectónico de capas ayuda a estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, de modo que cada uno de estos grupos se encuentre, a través de interfaces bien definidas.

Características:

* Útil en aplicaciones que involucran distintas clases de servicios que se pueden agrupar en forma jerárquica.
* Se quiere:
  + Separar interfaces/organizar código
  + localizar los cambios para minimizar su impacto
  + Separar las responsabilidades entre módulos
  + Minimizar el acoplamiento entre módulos
  + Reusar módulos

Los componentes en cada capa:

* deben pertenecer al mismo nivel de abstracción
* deben ser cohesivos

Los elementos de una capa se comunizan con sus pares y con elementos de las capas subyacentes:

* Estricto –solo con capas inmediatamente inferiores
* Relajado –con capas de cualquier nivel inferior

Las capas se deben comunicar mediante interfaces bien definidas

Invocaciones:

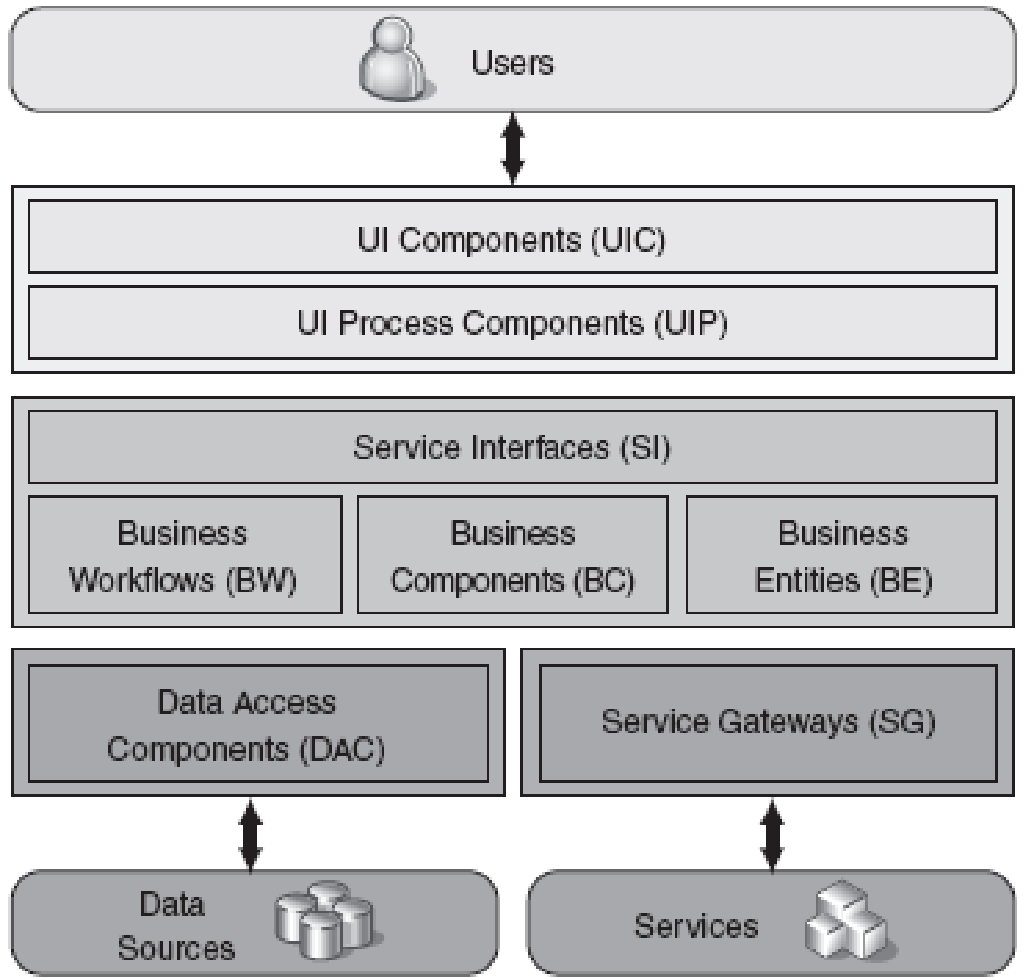
* De arriba hacia abajo
  + La interacción comienza con la capa superior y se propaga a las capas inferiores (situación más común)
* De abajo hacia arriba
  + La interacción comienza en la capa de menor nivel de abstracción y se propaga hacia las capas superiores (por ejemplo captura de eventos de bajo nivel)

Heurísticas de diseño

* Las capas deben agrupar funcionalidades de la solución en niveles de servicios bien definidos
* Se deben definir la interfaces de cada capa en función de las necesidades de la capa superior
* Las interfaces se pueden implementar utilizando distintos tipo de conectores
  + Llamadas a procedimientos, mensajes, WS, callbacks, etc.

Vista: Lógica

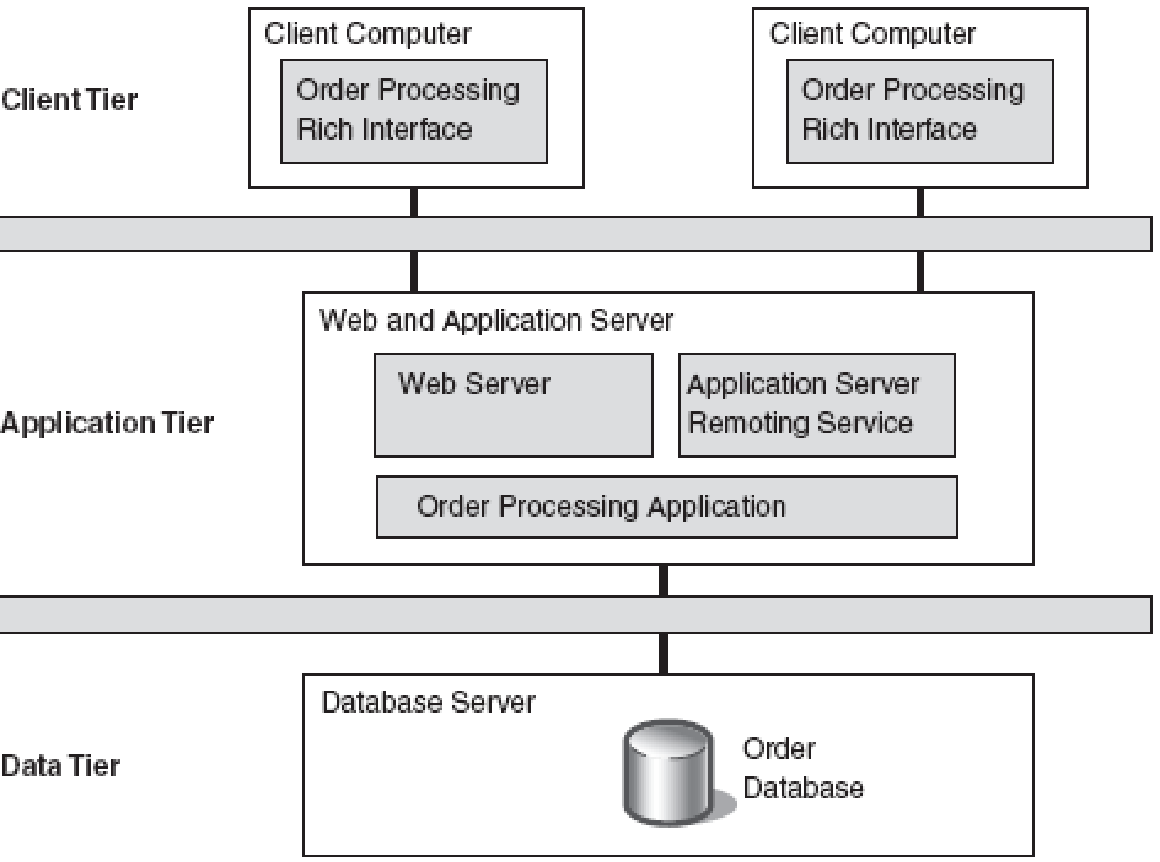
Implementación: Paquetes/namespaces



* Usos conocidos: Modelo OSI (ISO), máquinas virtuales (JVM), sistemas de información, sistemas operativos.
* Beneficios:
  + Soporta reusó: Las capas inferiores no dependen de las superiores, de modo que estas pueden ser reutilizadas en otros escenarios y permiten ser intercambiadas.
  + Soporte para la estandarización: Niveles de abstracción claramente definidos permiten el desarrollo de tareas e interfaces estandarizadas.
  + Eficiencia y disponibilidad: Las capas pueden ser distribuidas físicamente mejorando escalabilidad, tolerancia a fallos y eficiencia.
  + Testeabilidad: Interfaces bien definidas e intercambiabilidad soportan la testeabilidad del sistema.
* Desventajas:
  + Cambios en cascada: Cambios de comportamiento en una capa puede ser un problema severo.
  + Eficiencia: Datos, mensajes y excepciones deben ser transferidos entre un número de capas intermedias.
  + Dificultad para determinar la granularidad de las capas: Muy pocas capas no explotan correctamente el patrón. Excesivas capas agregan complejidad e impactan en la eficiencia.

## Estilo –Tiers (Capas físicas)

Es un estilo de despliegue que describe la separación de funcionalidades en segmentos similares al patrón de capas lógicas, pero cada uno de estos segmentos ubicados en nodos físicos separados. De este modo, define el despliegue de las capas lógicas de una aplicación. Soporta escalabilidad, disponibilidad y optimización de recursos.



Two Tier (2 Capas)

Client Tier: Contiene la lógica de presentación (interfaz de usuario) y la lógica de negocio o dominio.

* Database Tier: Representa otra máquina que hospeda la base de datos de la aplicación.

Three Tier (3 Capas)

* Client Tier: Puede contener la lógica de presentación (interfaz de usuario) o puede ser un cliente fino (thin client), como por ejemplo, un browser.
* Web/App Tier: Contiene la lógica de presentación (web) y/o la lógica de dominio de la aplicación en un servidor independiente.
* Database Tier: Representa otra máquina que hospeda la base de datos de la aplicación.

N-Tier (4 Capas o Más)

* Client Tier: Representa el browser (thin client)
* Web Tier: Contiene la lógica de presentación (web) en un servidor independiente.
* Business Logic Tier: Contiene la lógica de negocio de la aplicación en otro servidor.
* Database Tier: Representa otra máquina que hospeda la base de datos de la aplicación.

Se opta por este estilo cuando un sistema tiene categorías de funcionalidad bien definidas que:

* Son internamente cohesivas.
* Estables con respecto al cambio.
* Dependen entre ellas en forma predecible.
* No tienen dependencias cíclicas (esto es deseable)
* Tiene bajo acoplamiento con otras capas.

Motivo: Distribuir componentes en el hardware

Vista: Componentes y Conectores, Despliegue

Implementación: JEE

* Beneficios:
  + Modificabilidad: Las capas físicas son independientes, de modo que pueden variar independientemente sin afectar otras capas.
  + Escalabilidad: Capas lógicas se despliegan en capas físicas, de modo que existe buen soporte para escalabilidad horizontal.
  + Flexibilidad: Cada capa física puede ser gestionada independientemente.
  + Disponibilidad: A partir de la escalabilidad horizontal.
* Desventajas:
  + Seguridad: Aumenta la cantidad de objetivos de ataques.
  + Eficiencia: Comunicación remota.
  + Complejidad: En la infraestructura, configuración y gestión.
  + Costos.

## Publicador/Subscriptor

Descripción del Problema: Es necesario sincronizar automáticamente el estado de productores (Publicadores) y consumidores (Suscriptores) de datos.

Características:

* Puede utilizar un mediador (o las funciones pueden estar distribuidas entre los propios consumidores y productores).
* Cuando un publicador publica un dato se informa a todos los suscriptores (o a los interesados en ese dato) y reciben el dato.
* Se reduce el acoplamiento entre productores y consumidores.

Los clientes:

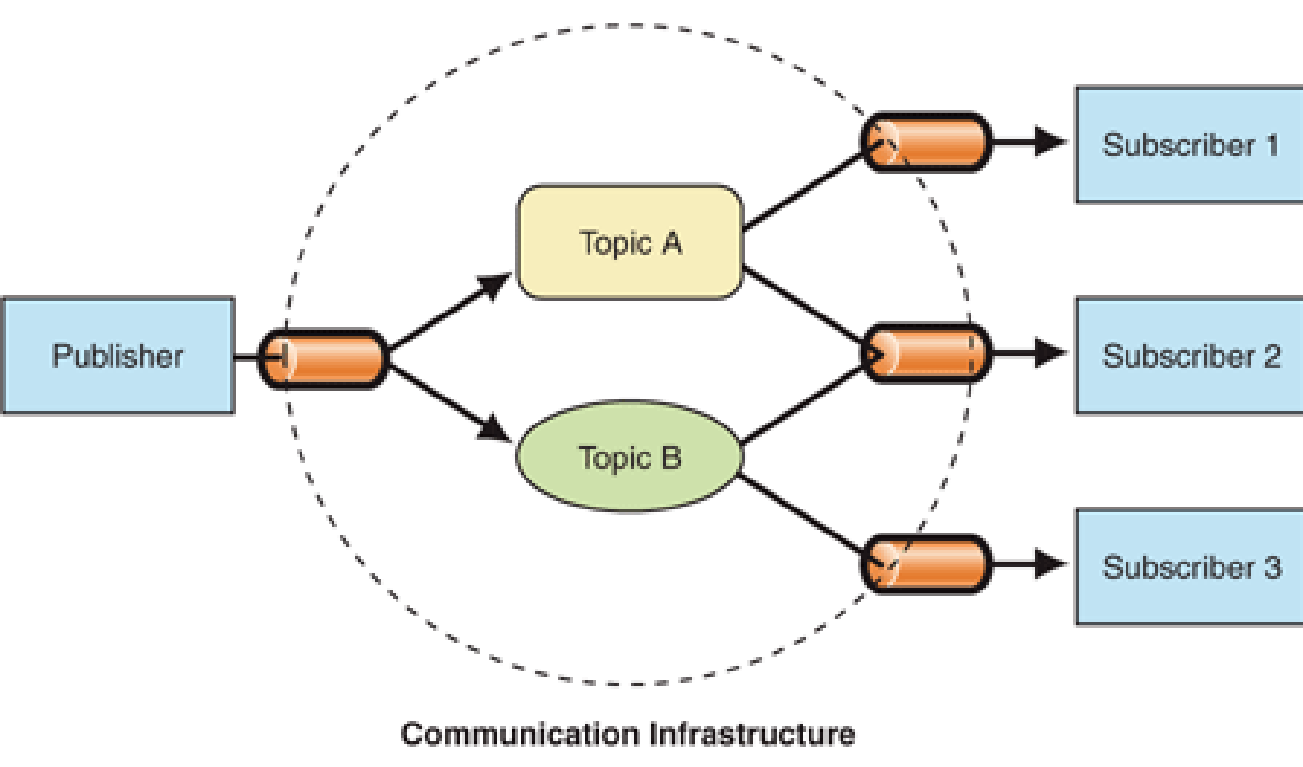
* Ejecutan en threads o procesos de control independientes.
* Pueden ser productores, consumidores o ambos.
* No conocen la estructura del repositorio.

El Mediador:

* Provee interfaces para publicar y suscribirse.
* Puede utilizar reglas particulares para notificar a varios suscriptores interesados en el mismo dato.
* La notificación puede ser sincrónica o asincrónica

Basados en listas (tópicos)

* Los suscriptores se suscriben a un tópico (subject)
* El publicador o el mediador:
  + mantienen una lista de suscriptores
  + Proveen interfaces para suscribirse o borrarse de la lista
  + Cuando se recibe o genera un mensaje (evento) con el tópico se envía a todos los suscriptores de la lista
* El suscriptor
  + Provee una interfaz para ser notificado del evento



Basados en Broadcast (tópicos)

* El publicador o el mediador:
  + Cuando se recibe o genera un mensaje (evento) con el tópico se envía a la red
* Todos los suscriptores
  + Reciben los mensajes y deciden si lo procesan basándose en el tópico

Motivo: Notificar eventos que se publican a suscriptores sin que se conozcan

Cualidades: Mantenibilidad (separar responsabilidades), escalabilidad, eficiencia (sincronización)

Vista: Componentes y Conectores

Implementación: JMS (Tópicos), WCF/Biztalk