Notas de Clase: DSI

07/06/2021

Workflow de Diseño

Podemos definir **diseño** como el proceso *iterativo* mediante el cual se aplican varias tecnicas y principios con el objetivo de definir un dispositivo, un proceso o un sistema con suficiente nivel de detalle como para permitir su realizacion fisica, permitiendo transformar un *modelo logico* en un *modelo fisico* de acuerdo a las *restricciones* del negocio.

Diferencias del Analisis y el Diseño en el PUD:

El diseño es un modelo *fisico* centrado en la arquitectura (caracteristica del PUD).

Las tendencias de la Ingenieria de Software son:

- Darle un papel protagonico a la etapa de requerimientos.
- Poner especial atencion en la arquitectura de software.
- Utilizar las mismas herramientas para modelar.
- Utilizar un proceso (por ej. PUD)

Los aspectos que debemos modelar del software son:

- Arquitectura
- Datos
- Procesos
- Interaccion Humano-maquina
- Formas de Entrada/Salida
- Procediemitnos Manuales (no se implementa solo con codigo)

Los primero es diseñarse siempre es la *arquitectura*.

El *diseño arquitectonico* es la pieza principal a partir de la cual se desarrollan los demos diseños, siempre alineados con el diseño arquitectonico que funciona como un plano. Modela los requerimeintos de calidad o *no funcionales*. Dado que involucra un proceso de decisiones significativas, es importante que se documente cada decision, su justificacion y contexto, dada la naturaleza evolutiva y cambiante de la tecnologia.

El *Diseño de Datos* busca transformar los requerimienos en las estructuras de datos necesarias para hacer persistir el software.

El *Diseño de los Procesos* transforma elementos estructurales en una descripcion procedimental de los componenetes del software. Por ejemplo si los CU son automaticos o temporales. Hay que tener en cuenta procesos para seguridad, autentiucacion de usuarios, backups y recuperacion, etc. Es decir, transforma las Realizaciones de CU de Analisis en Realizaciones de CU de Diseño.

El *Diseño de Interaccion Humano-Maquina* es la disciplina relacionada con el diseño, evaluacion e implementacion de sistemas computacionales interactivos para uso humano. Una parte de este diseño es la UX y la UI. Tambien se tienen el cuenta el diseño de la posicion corporal del usuario final del software. Busca que los usuarios puedan utilizar el software de la mejor manera posible.

El *Diseño de Formas de Entrada/Salida* describe como se ingresa informacion al software y como se presentaran las salidas del mismo.

Los sistemas *criticos* son aquellos que no deberian fallar dado la magnitud de sus consecuencias.

El *Diseño de los Procedimientos Manuales* describen como se integra el software ya puesto en produccion al Sistema de Negocio, teneindo en cuenta las adaptaciones necesarias por parte del negocio para que se pueda integrar correctamente. Suele usarse BPMN para representar dicha integracion.

14/06/2021

Diseño Arquitetonico / Diseño de Arquitectura de Software

Podemos definir a la *arquitectura* como el conjunto de decisiones significativas que tomamos para poder resolver los RNF teniendo en cuenta el contexto (es decir el negocio donde va a a funcionar dicho sistema), y se modela a traves de vistas. En PUD, una arquitectura estable y madura es indicacion de que es hora de salir de la etapa de Elaboracion para inicar la Etapa de Construccion.

Entonces el *diseño de la arquitectura* se define como un diseño estrategico (porque define aspectos globales de decision, que luego se implementan en tacticas mas detalladas) que se encarga de asignar modelos de requerimientos escenciales a una tecnologia especifica.

TEMAS PARCIAL 2:

- Patrones Arquitectonicos
- Vistas Arquitectonicas
- RNF

ACTIVIDADES DEL PROCESO DE DISEÑO DE LA ARQUITECTURA

- **Determinar Requerimientos Arquitectonicos:**

  Los requerimientos (funcionales y no funcionales) son una ENTRADA del Workflow de Diseño. Se van a analizar los RNF para poder determinar si son significativos para la arquitectura (si impactan o no impactan en las decisiones significativas para la arquitectura). Inicialemente, es suficiente ubicar los requerimientos en 3 categorias:

  - **Alto:** La aplicacion debe soportar este requerimiento. Se los debe atender desde la primer iteracion y no se pueden posponer. Conducen el diseño de la arquitectura.
  - **Medio:** Necesitara ser soportado en alguna etapa, pero no necesariamente en el primer release.
  - **Bajo:** Estos son parte de la lista de deseos. Las soluciones que los incluyen son deseables, pero no son conductores del diseño.

  La *priorizacion* es un concepto engañoso dado que los requeriemientos pueden entrar en conflicto entre si, ademas que algunos RNF se condicionan entre si, lo que implica que se deban implementar en la misma iteracion y que tengan la misma prioridad entre si. Dado un escenario de iteraciones, si determinamos que la prioridad de un RNF es alta, entonces esos RNF se deben atender en la primer iteracion del software.

- **Diseño Arquitectonico**: Se compone de 2 actividades:
  - **Elegir el Framework de Arquitectura:** Un framework esta compuesto por una serie de Patrones Arquitectonicos, que implican soluciones conocidad. Estas soluciones estan probadas y son validas por lo que nos permiten minimizar los riesgos. En base a los Requerimeintos significativos para la arquiotectura, se debe elegir el Framework adecuado de acuerdo a los patrones arquitectonicos que deseamos aplciar en el software.
  - **Distribuir Componentes:** Implica la definicion de la estructura y las responsabilidades de los componenetes que constituiran la arquitectura.
- **Validacion:** Implica un proceso de control para probar la arquitectura, comunmente recorriendo el diseño contra los requerimeintos existentes y cualquier requerimiento futuro, posible o conocido.

Patrones

Los **patrones arquitectonicos** son soluciones de alto nivel, validos y probados, que nos permiten resolver aquellos RNF que son significativos para la arquitectura de froma que minimizamos los riesgos y modelamos de forma correcta y consistente. Comunmente se aplican a partir de Frameworks.

Los patrones pueden ser:

- **Platonicos:** Es un patron idealizado, rara vez se aplica al codigo en forma exacta.
- **Embebidos:** Se lo ve en los sistemas reales, y a menudo rompen las restricciones estrictas de los patrones platonicos, generalemente a favor de una gran compensacion.

La distincion entre un *patron arquitectonico* y un *estilo arquitectonico* radica en que los estilos son de una jerarquia mayor que los patrones, de forma que multiples patrones pueden aparecer en un mismo diseño. Por otro lado, un sistema tiene usualmente un unico estilo arquitectonico dominante.

Los patrones que vamos a ver son:

- **Patron Layered:** La **arquitectura estratificada** o **en capas** implica la estratificacion de la arquitectura en una serie de capas para poder organizar los componenetes de software teneindo en cuenta el bajo acoplamiento y la alta cohesion, y donde cada una se encuentra en un nivel conceptual distinto. Es decir, cada capa funciona como un subsistema. Aplica a elementos de codigo y es parte del tipo de vista *modulo*. Las capas basicas son:
    - *Presentacion:* Aloja a todas las clases de tipo boundary, es decir, las pantallas e interfaces con las interactua el usuario. Es la mas cercana al usuario.
    - *Logica de Negocios:* Alberga a aquellos algoritmos y procesos que resuelven los RF y algunos de los RNF. Abarcaria a las clases de entidad y las de control.
    - *Administracion de Datos:* Aloja la base de datos. Es la mas alejada del usuario.

Podriamos adicionar una capa de *servicios web*, una de *persistencia*, etc.

- **Patron N-Tier:** Una **arquitectura cliente-servidor** diferencia una maquina de cliente que hace peticiones a un servidor atraves de la red, el cual atiene a las peticiones del cliente. Se puede combinar con la arquitectura en capas. Podemos aplicar hasta N niveles para tener una arquitectura de software distribuida. Posee comunicaicon *sincronica*.

- **Patron Publish-Suscribe:** Consiste en un conjunto de componenetes de software denominados *suscriptores*, que deben manifestar en interes de estar notificados sobre la ocurrencia de un cierto evento, y un componente *publicante* que frente a dicha ocurrencia informa sobre ello a traves de la creacion o publicacion de un *topico* al cual se suscribe cada suscriptor para ser notificado. Es una estructura muchos a muchos. Posee muy bajo acoplamiento (desconocimeinto entre suscriptores y publicantes). Es muy utilizado en los entornos mobile. Posee comunicacion *asincronica*.

- **Patron Broker:** Su motivacion es atender problemas de *compatibilidad de formatos* entre componentes, permitiendo a los *receptores* comprender la informacion proveniente de los *remitentes*

en un formato de entrada, brindando un formato de salida interpretable por el sistema. Utiliza un sistema de *ruteo* para poder enviar a los receptores correspodnientes aquella informacion proveniente del remitente indicado, mediante puertos de entrada y salida. Comunmente es utilizado cuando el sistema se comunica con sistemas externos para enviar o recibir informacion. Posee comunicacion *asincronica*.

CAPAS = Software NIVELES = Hardware

28/06/2021

- **Patron Messaging (Arquitectura Comunicando):** Funciona a partir de una estructura cliente - servidor, donde se implementa una cola (por lo general, FIFO) en la cual se acumulan los mensajes, de modo que funciona de modo *asincrono*. Se va a configurar la calidad de servicio, modificando la cantidad de intentos, los mensajes prioritarios, etc. Un tipo de sistema que utiliza mucho esta arquitectura son los *sistemas de afluencias*. Lo podemos observar principalemnete en los servicios de correo y los sistemas de afluencia.

Si se desea tener alta disponibilidad, se debe implementar *redundancia* en la cola de mensajes.

- **Patron Process Coordinator (Arquitectura Coordinador de Proceso):** Es un patron para dar soporte a procesos de negocios complejos en cuanto a la cantidad de pasos, o en cuanto a las reglas de negocio que hay que resolver, procesar y validar, o bien que posee una logica variable. Se implementa un modulo *coordinador de proceso*, quien es el responsable de atender la solicitud del negocio y entrega el resultado correspondiente, mediante la colaboracion con x servidores que lo asisten cada uno con un paso del proceso. Este coordinador posee la logica del negocio, lo cual puede ser ventajoso a la hora de la modularidad, pero puede ser desventajoso porque puede actuar como un cuello de botella para el rendimiento del sistema. Funciona de forma *asincrona*. Los servidores no se conocen (posee bajo acoplamiento).

- **Patron MVC (Model View Controller):** Consiste en una serie de *vistas* que manifiestan su interes en el estado de un *modelo*, el cual contiene la inforamcion sobre el estado de todos los objetos de interes. Es decir que las vistan son representaciones particulares del modelo que aportan multiples formas de ver e interactuar con los daots, y que se relacionan a traves de un objeto denominado *controlador*, que separa la presentacion e interaccion de los datos del sistema. Tiene la ventaja de permitir que los datos cambien de manera independiente de su presentacion y viceversa, y ofrece soporte a distintas representaciones de los mismos datos, y los cambios en una represetnacion se muestran en todos ellos (evitando asi inconsistencias). Por otro lado, tiene la desventaja de que puede implicar codigo adicional y complejidad de codigo cuando el modelo de datos y las interacciones son simples. Posee comunicacion *asincronica*. Separa la presentacion e interaccion de los datos del sistema (bajo acoplamiento).

Arquitecturas de Sistemas Distribuidos

Un *sistema distribuido* es un sistema de software que se ejecuta en un grupo de procesadores cooperativos integrados, conectados por una red. Poseen las siguientes caracteristicas:

- *Comparticion de Recursos:* Recursos de hardware y software asociados a una red.
- *Apertura:* Son sistemas abiertos, que se diseñan sobre protocolos estandar que combinan equipamiento y software de diferentes vendedores.
- *Concurrencia:* Permiten que varios procesos esten operando al mismo tiempo sobre diferentes computadoras de la red.
- *Tolerancia a Fallas:* Dada la alta disponibilidad de recursos y el potencial para reducir informacion se permite un cierto nivel de tolerancia a fallos.
- *Escalabilidad:* Pueden crecer incrementando recursos para cubrir nuevas demandas. Pueden crecer en *tamaño*, *distribucion* y *manejabilidad*.

Sus *desventajas* son:

- *Complejidad:* Son mas dificiles de comprender y probar.
- *Seguridad:* Se difivulta asegurar la integridad y la degradacion del servicio.
- *Manejabilidad:* Los defectos pueden propagarse de maquina a otra, implicando que es mas dificil de gestionar y administrar.
- *Impredecibilidad:* La respuesta depende de la carga total en el sistema, de la organizacion y de la red.

Identificamos una serie de arquitecturas para los sistemas distribuidos:

- **Arquitectura Maestro/Esclavo:** Fueron el primer nivel de distribucion de arquitectura, permitiendo multiples procesadores para un mismo hardware. Consiste en un procesador principal denominado *maestro* que delega y asigna acciones de procesameiunto a los procesadores secundarios (denominados *esclavos*). Deriva en la *arquitectura cliente/servidor*, donde ademas se comienza con la adicion de capas para lograr arquitecturas de n-capas que sirven a los sistemas distribuidos.
- **Arquitectura Peer-to-peer (Descentralizada):** Los componentes de software (nodos) funcionan todos con un mismo rol, pudiendo funcionar independientemente como cliente o como servidor. Su principal problema es la redundancia, de forma tal que las respuestas a peticiones pueden ser contestadas por multiples nodos. Posee alguna arquitecturas derivadas como la Peer-to-peer Semi Centralizada, que utiliza un servidor llamado *superpar*.

Encontramos tambien *arquitecturas de vista de distribucion* para poder atender a software de alta disponibilidad:

- **Arquitectura Espejada (Mirrored):** Implica la duplicacion o triplicacion de los elementos de hardware en linea y que corren en paralelo, segun los requerimeintos de disponibilidad. Permite mantener alto funcionamiento aun frente a fallas en algun equipo de hardware.
- **Arquitectura Rack:** Los servidores se acomodan el pilas para utilzar mejor el espacio, y todas se conectan a la misma red. Dicha red puede tener multiples conexiones a internet.
- **Arquitectura Granja de Servidores:** Implica la utilizacion de multiples racks en una misma habitacion, aportando un recurso masivo para alojar cualquier aplciacion. Es muy facilmente escalable.

Encontramos clientes *livianos* y clientes *pesados*:

- Liviano: No posee la capa de logica de negocios (se encuentra en el servidor).
- Pesado: Posee la capa de logica de negocios.

02/08/2021

Proceso de Diseño de la Arquitectura de Software

DESCRIPCION DEL PROCESO DE DISEÑO ARQUITECTONICO:

- *Determinar los requerimientos arquitectonicos:* Implica la creacion de una definicion o modelo de los requerimientos que conduciran el diseño arquitectonico y su priorizacion. Es decir, implican una decision del diseño arquitectonico.
- *Diseño Arquitectonico:* Implica la definicion de la estructura y las responsabilidades de los componenetes que constituiran la arquitectura.
- *Validacion:* Implica un proceso de control, para probar la arquitectura, comunmente recorriendo el diseño contra los requerimeintos existentes y cualquier requerimiento futuro, posible o conocido.

A partir de los *requeriemientos de la arquitectura* elegimos el *Framework de Arquitectura* apropiado, para luego distribuir los *componentes*. Permitien obtener como resultado vistas y documentos.

Tipos de Vistas

Un **modelo de diseño** es un conjunto o categoria de vistas que pueden ser facilmente conciliadas unas con otras. Las vistas que no pueden ser conciliadas perteneces a tipos de vistas diferentes.

Los tipos de vistas son:

- **Vistas de Modulo:** Contiene vistas de los elementos que se pueden ver en tiempo de compilacion. Definiciones de tiempo de componentes, puertos, conectores, clases e interfaces. Encontramos al Patron Layered.
- **Vistas de Ejecucion (Runtime):** Conteine vistas de los elementos que se pueden ver en tiempo de ejecucion. Incluye escenarios de funcionalidad, lista de responsabilidades y ensambles de compoennetes. Instancias de componentes, conectores y puertos (como objetos). Encontramos al Patron Cliente Servidor N-Tier.
- **Vistas de Distribucion:** Contiene vistas de elementos relacionados con la distribucion del software del hardware. Incluye al resto de los patrones (Messaging, Broker, Publish and Suscribe, Process Coordinator, Cliente Servidor N-Tier).
- **Vistas Spanning (Atrviesan):** Se utilizan para mostrar ciertos requerimientos con tal de que estas no colicionen. INVESTIGAR

Una **vista** es una proyeccion de un modelo/s para un involucrado o interesado en prticular, desde sus interes, perspectivas y necesidades. Por lo tanto, la vista es una abstraccion del modelo/s que se adecua para el interesado. El **punto de vista** es una definicion teorica que explica lo que la vista va a mostrar.

Al ser abstracciones, deben ser lo mas *minimas y pequeñas* posibles. Aproximadamente, solo del 10% al 15% de los CU son significativos para la arquitectura.

NO TODAS LAS VISTAS CON Arquitectonicas

Vistas Arquitectonicas (PUD)

Las vistas propuestas por el PUD son TODAS arquitectonicas, y poseen una parte estatica y una dinamica, usando diagramas de UML. Las vistas del PUD son:

- **Vista de Casos de Uso:** Es la primer vista que se observa en el PUD. Llamada tambien *vista de fuincionalidad*, muestra unicamente aqullos CU que son relevantes para resolver la arquitectura (es decir, aquellos CU significativos para la arquitectura, segun plantean las caracteristicas del PUD). Su parte *estatica* se modela mediante un Diagrama de Casos de Uso, y su parte *dinamica* mediante Diagramas de Colaboracion o de secuencia. Es artefacto del WF de Requeriemientos (parte estatica) y Analsis (parte dinamica).
- **Vista de Diseño:** Permite mostrar los elementos relevantes en terminos de subsistemas de componenetes, y las relaciones entre dichos elementos para satisfacer los requerimeintos significativos para la arquitectura. La parte *estatica* se construye mediante Diagrama de Clases o de Componenetes. Definimos a un *componente* como un tipo especial de clase, con la aprticualridad de que es fisico (es codigo). La parte *dinamica* se representa mediante Diagrama de Secuencia. Es artefacto del WF de Diseño.
- **Vista de Implementacion:** Implica el hecho de la codificacion del producto. Define como se van a configurar los entornos de desarrollo, y mantiene la integridad del codigo mediante la administracion de configuracion para no perder informacion relevante y mantener consistencia. Su parte *estatica* se mdoela mediante Diagrama de Componentes. La parte *dinamica* se representa mediante Diagrama de Secuencia. Es artefacto del WF de Diseño.
- **Vista de Proceso:** Hace foco en los procesos relacioandos a las *clases acticas*, que son las clases que poseen el hilo conductor del sistema. La parte *dinamica* se representa mediante Diagrama de Secuencia. La parte *estatica* se representa mediante Diagrama de Componenetes. Es artefacto de WF de Diseño.
- **Vista de Despliegue:** Muestra la distribucion del software en los nodos de hardware para que pueda ser desplegado, de forma tal que el hardware de soporte a la arquitectura del sistema. La parte *dinamica* se representa mediante Diagrama de Secuencia. Su parte *estatica* se modela mediante Diagrama de Nodos. Es artefacto del WF de Diseño.

Las vistas estaticas del PUD correspondel a las Vistas de Modulo, y las vistas dinamicas corresponden a las Vistas de Runtime.

No siempre son necesarias todas las vistas, asi como hay situaciones donde se peuden requerir vistas adicionales, siempre de acuerdo a los requerimientos y necesidades del producto de software.

Podriamos utilizar una *Vista de Datos* para completar el Modelo 4+1, ya que atiende el punto de mayor retardo en un sistema: Los accesos a BD relacionales.

23/08/2021

Los temas que SI O SI al *parcial practico* son:

- Vista Funcional
- Vista de Diseño / Subsistemas
- Vista de Despliegue / Nodos

Los temas del *parcial teorico* son:

- Definicion del Diseño
- Aspectos
- Estrategias de Prototipado y Ensamblaje de Componentes
- Diseño en el PUD, Trabajadores, Actividades (Cap. 9 del Libro del PUD)

A proceso de estrategias de prototipado se lo conoce tambien como **Ingenieria de Software basada en Componentes** o **ISBC**.

Estrategia de Prototipado

La *Estrategia de Prototipado* es una elección de modelo de proceso que se recomienda elegir a la hora de implementar un proyecto complejo, con dominio no familiar, que utilizará una tecnología desconocida; de ahí que surge la necesidad de requerirse el uso de prototipos en el diseño y la implementación, además de utilizarlos durante la validación de requerimientos.

Es un modo de desarrollo de software, implementando prototipos.

Un **prototipo** es una primera version de un nuevo tipo de producto, en el que se incorporan solo algunas de als caracteristicas del sistema final. Funciona como maqueta del sistema para facilitar la comprension del problema y entender sus posibles soluciones. La finalidad de los *prototipos* es probar varias suposiciones formuladas por analistas y usuarios respecto a las caracteristicas requeridas por el sistema. Se crean con rapidez, evolucionan a traves de un proceso interactivo y tienen un bajo costo de desarrollo.

Los prototipos poseen las siguientes caracteristicas:

- Poca fiabilidad
- Funcionalidad limitada
- Caracteristicas de operaciones pobres.

En general siemrpe es recomendable usar prototipos, pero es especia,ente ventajoso cuando:

- El area de aplicacion no esta bien definida.
- Hay un elevado costo de rechazo.
- Se utilizan nuevas tecnologias o tecnicas.
- Se desconocen los requerimientos. Hay elevados costos de inversion. Hay factores de riesgo asociados al proyecto.

Los *beneficios* que provee el uso de prototipos son:

1. Aumento de la productividad
2. Desarrollo planificado
3. Entusiasmo de los usuarios

Los *Tipos* de prototipos son:

- *Prototipado de Interfaz de Usuario:* Modelos de pantallas.
- *Prototipado Funcional:* Implementa algunas funciones y las corrije y refina.
- *Prototipos Arquitectonicos:* Permiten evlauar decisiones arquitectonicas de infraestructura, tecnologia e integracion.
- *Modelos de Rendimiento:* Evaluan el rendimiento de una aplciaicon critica.

Respecto a su *utilidad*, pueden ser:

- *Rapidos:* Se desechan luego de cumplir su proposito, que es el analisis y validacion de requisitos.
- *Evolutivos:* El prototipo va mutando, auemtnando a medida que se descubren nuevos requisitos hasta convertirse en el sistema requerido.

Respecto al *alcance*, pueden ser:

- *Vertical:* Desarrolla completamente alguna de las funciones.
- *Horizontal:* Desarrolla parcialemtne todas las funciones.

Las *etapas* del modelo de prototipos son:

1. Identificacion de los requerimientos conocidos.
2. Desarrollo de un modelo de trabajo.
3. Participacion del usuario.
4. Revision del prototipo.
5. Iteracion del proceso de refinamiento.

Estrategia de Ensamblado de Componentes

La *Estrategia de Ensamblado de Componentes* es una decisión arquitectónica y de diseño de la solución final del software a construir, que implica desde decidir implementar por componentes, definir la granularidad del componente hasta su ensamblando final y prueba de integración.

Es un modelo evolutivo para el desarrollo del software, con un enfoque iterativo.

Componente

Es una *pieza de software* (clase) independiente de software, que tiene el proposito de poder ser *reutilizada* con facilidad, aportando *modularidad* y *alta cohesion*. Encapsulan alguna funcionalidad expuesta mediante interfaces estandar. Debe diseñarse de forma tal que a la hora del ensamblamiento a otros componentes en el contexto del sistema, estos componenetes posean *bajo acoplamiento*.

Los *beneficios* de la reutilziacion es que permite poder reducir los tiempo de desarrollo drasticamente, asi como el coste de los proyectos. Esto deriva en un mayor indice de productividad.

Las *ventajas* del ISBC son:

- *Reutilizacion del Software*
- *Simplificacion de las pruebas*
- *Simplificacion del mantenimiento del sistema*
- *Mayor calidad del software*

Tipos de Composiciones

Pueden ser:

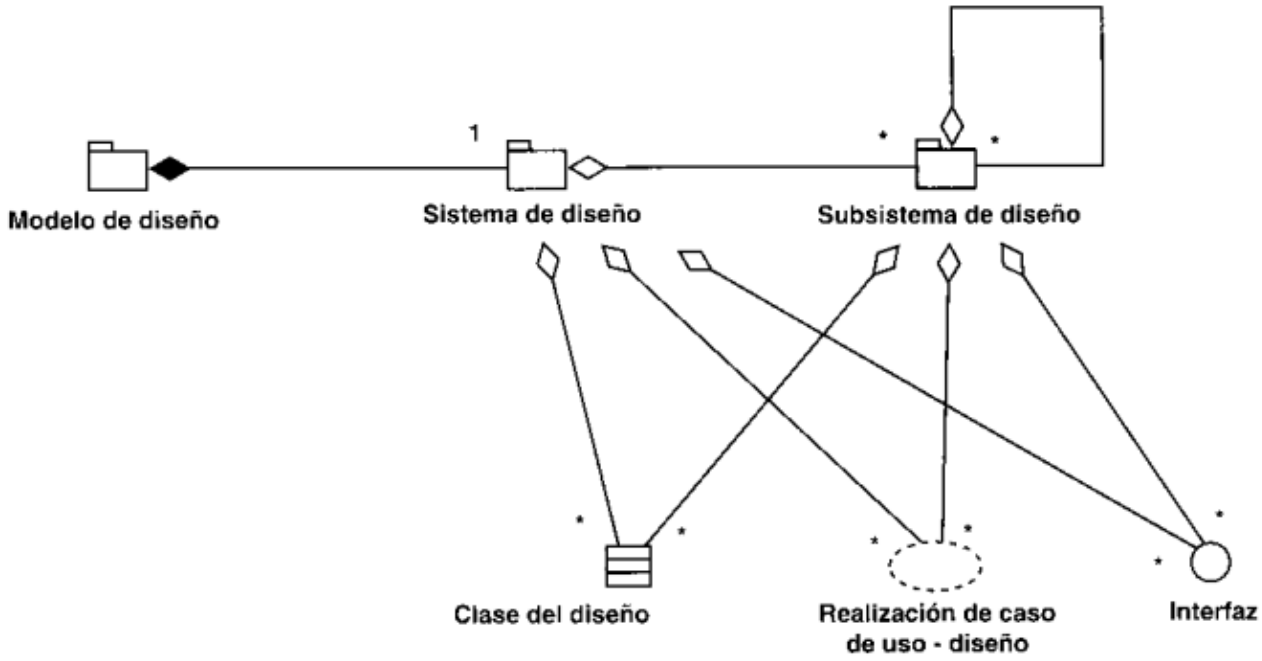
- **Secuencial:**
- **Jerarquica:**
- **Aditiva:**

Diseño en el PUD

El modelo de analisis es una entrada escencial del modelo de diseño, dado que impone una estructura del sistema que debemos esforzarnos para conservar lo mas fielmente posible.

El diseño debe ser mantenido durante todo el ciclo de vida del software. Su foco se da entre las ultimas iteraciones de la etapa de elebaoracion y las primeras de la fase de construccion, segun el PUD.

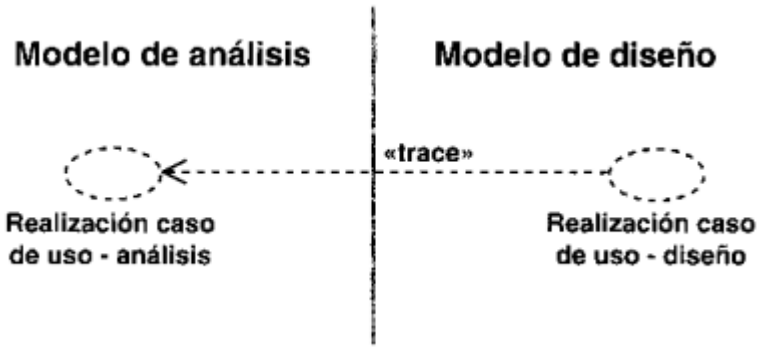
Artefactos del Diseño

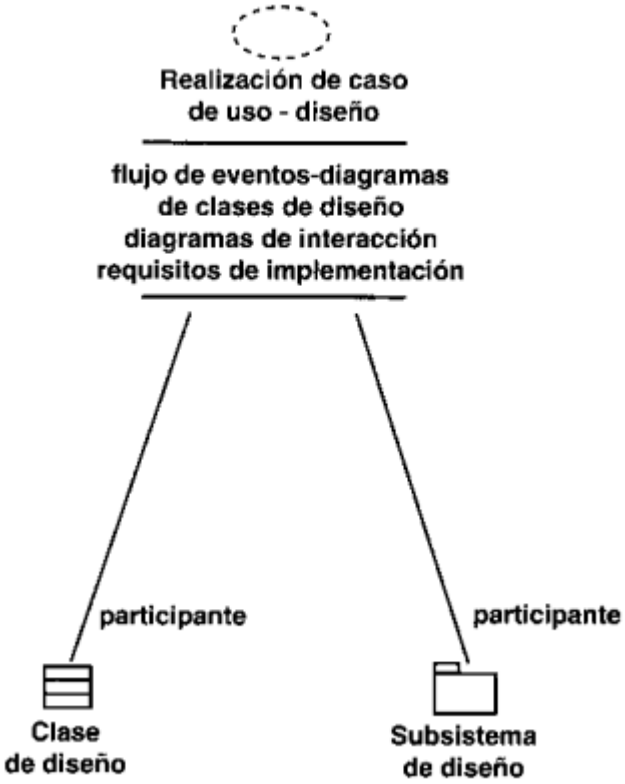


Los **artefactos** del Diseño son:

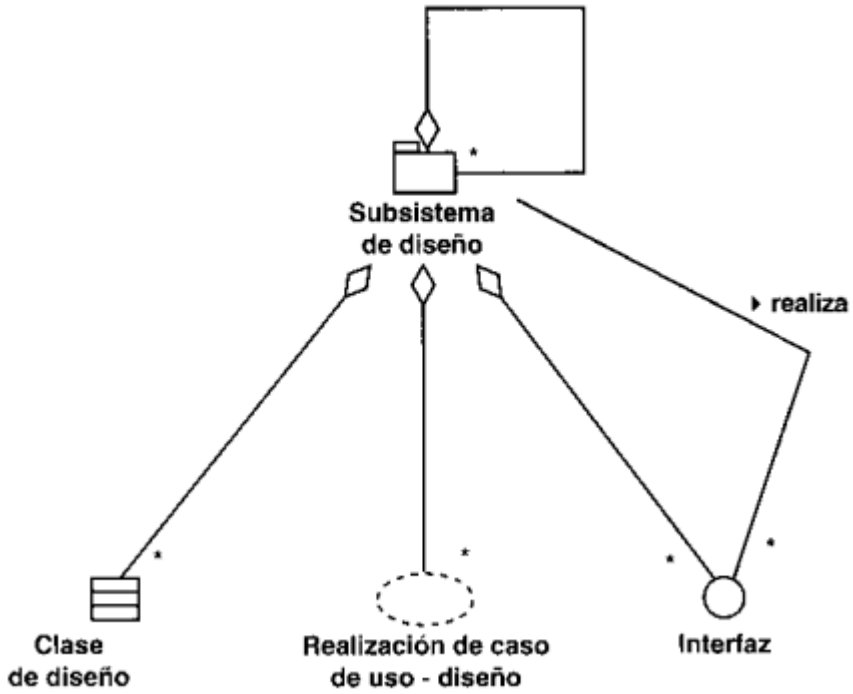
- *Modelo de Diseño:* Modelo de objetos que describe la realziacion fisica de los casos de uso, centrandose en como los RNF y otras restricciones tienen impacto en el sistema. Es un plano de la implementacion y por lo tanto es una entrada fundamental de las actividades de implementacion. Denota subsistemas (que son escencialmente abstracciones) que permiten organziar el modelo en porciones manejables, favoreciendo la alta cohesion y el bajo acoplamiento.
- *Clase de Diseño:* Es una abstraccion *sin costuras* de una clase o construccion similar en la implementacion del sistema. Ser sin costuras implica:
    - Se especifican en un lenguaje de programacion.
    - Su visibilidad es especifica de una frecuencia.
    - Pueden posponer el manejo de algunos requisitos para actividades subsiguientes de la implementacion.
    - Pueden proporcionar interfaces.

- *Realizacion de CU-Diseño:* Es una colaboracion en el modelo de diseño que describe como se realiza y ejecuta un CU especifico en terminos de las clases de diseño y sus objetos. Es decir, proporciona una realziacion fisica de la Realziacion de CU-Analisis trazada. Posee una parte estatica represetnada por diagramas de clases, y una parte dinamica represetnada con diagramas de interaccion.





- *Subsistema de Diseño:* Consiste en una pieza manejable dentro del diseño para poder organziar a los demas artefactos del diseño, represetnando una separacion de los aspectos del diseño. Esta conformado por clases de diseño, realizaciones de CU-Diseño, interfaces y otros subsistemas. Ademas, permite tambien exportar su funcionalidad en terminos de operaciones, mediante el uso de interfaces. Sus contenidos estan fuertemente asociados (alta cohesion) y sus dependendias con otros subsistemas so nminimas (bajo acoplamiento).

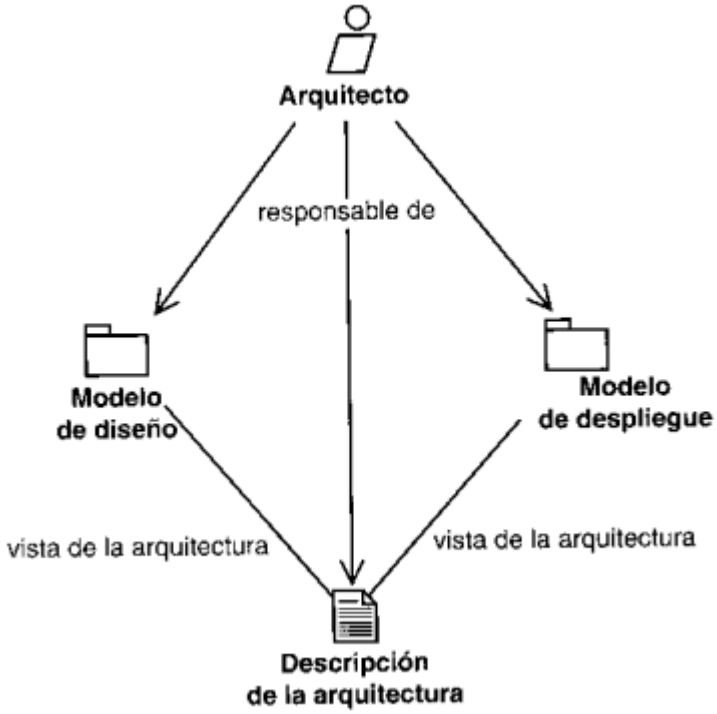


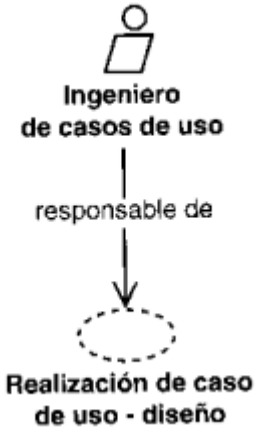
- *Interfaz:* Constituye una forma de separar la especificacion de la funcionalidad, de modo que permite especificar que oepraciones proporciona una clase o subsistema del diseño. Son escencialemtne *cascaras*. Definen las interacciones permitidas entre los susbsistemas.
- *Descripcion de la Arquitectura (Vista del Modelo de Diseño):* Contiene una vista de la arquitectura del modelo de diseño que meustra sus artefactos relevantes para la arquitectura. Los artefactos que suelen considerarse *significativos* para la arquitectura son:

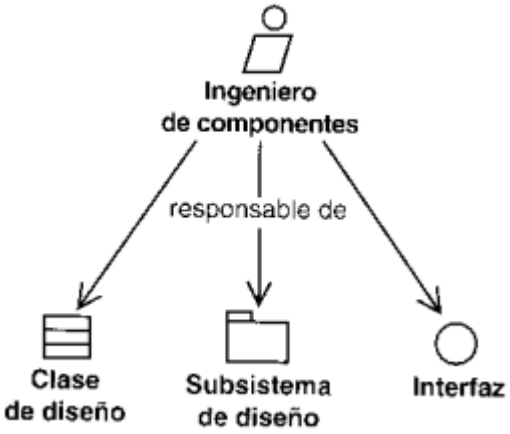
1. Los subsistemas, interfaces y dependencias entre ellos.
2. Las clases de diseño fundamentales.
3. Las Realziaciones de CU-Diseño que describen alguna funcionalidad importante y critica que debe desarrollarse pronto dentro del ciclo de vida del software.

- *Modelo de Despliegue:* Es un modelo de objetos que describe la distribucion fisica del sistema en terminos de como se distribuye la funcionalidad entre los nodos de computo. Cada nodo representa un recurso computacional (dispositivo o hardware similar) y poseen relaciones que denotan la comunicacion entre ellos
- *Descripcion de la Arquitectura (Vista del Modelo de Despliegue):* Contiene una vista de la arquitectura del modelo de despliegue que meustra sus artefactos relevantes para la arquitectura. Todos los aspectos del modelo de despleigue deberian mostrarse en la vista arquitectonica.

Trabajadores del Diseño



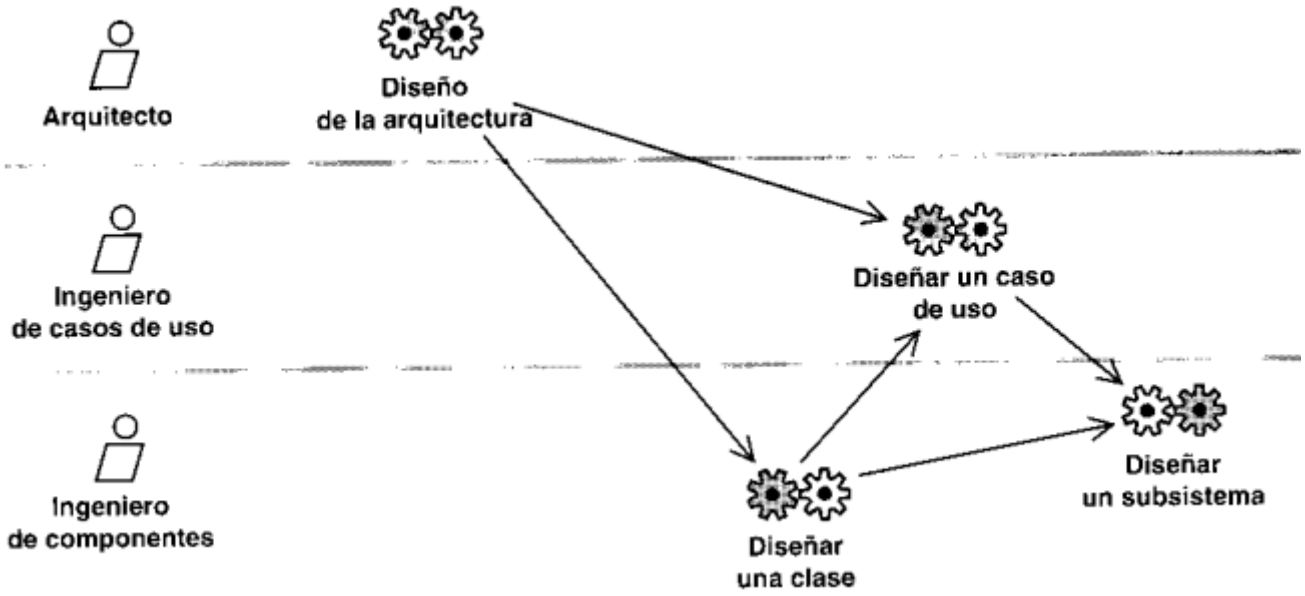




- *Arquitecto:* Es el responsable de la integridad de los modelos de diseño y despleigue, garantizadno que estos sean correctos, consistentes y legibles como un todo (que cumplan su funcionalidad esperada). Tambien es el responsable de la descripcion de la arquitectura de ambos modelos ya mencionados, de acuerdo a las vistas arquitectonicas.

- *Ingeniero de CU:* Es el responsable de la integridad de una o mas realziaciones de CU-Diseño, garantizadno que cumplen con los requisitos esperados de ellos y que cumplan con los comportameintos su correspondiente realziacion de CU-Analisis. No es responsable de las clases, subsistemas, interfaces y relaciones del diseño.
- *Ingeniero de Componentes:* Define y mantiene las operaciones, metodos, atributos, relaciones y requisitos de implementascion de las clases de diseño, garantizadno que cumplan con su funcionalidad esperada. A su vez es el responsable de mantener la integridad de uno o mas subsistemas y su contenido, lo cual incluye a las interfaces que estos proporcionan.

Flujo de trabajo



- *Diseño de la Arquitectura:* Tiene como objetivo esbozar los modelos de diseño y despliegue y su arquitectura mediante la identificacion de:
    - Nodos y sus configuraciones
    - Susbsistemas y sus interfaces
    - Clases de Diseño SPA
    - Mecanismos de diseño genericos que tratan requisitos comunes
- *Diseño de un Caso de Uso:* Sus objetivos son:
    - Identificar las clases del diseño participantes en las realizacion de CU-Diseño
    - Describir las interacciones de los objetos que interactuan en las realziaciones de CU-Diseño
    - Identificar los subsistemas e interfaces participantes en las realziaciones de CU-Diseño
    - Describir las interacciones entre los subsistemas
    - Capturar los requisitos de implementacion de los CU
- *Diseño de una Clase:* Su objetivo es crear clases de diseño que cumplan con su papel en las realziaciones de CU-Diseño y los RNF que se aplican a estos, incluido el mantenimiento de las clases. Para ello se debe:
    - Esbozar las calses de diseño
    - Identificar las operaciones
    - Identificar los atributos
    - Identificar asociaciones y agregaciones
    - Identificar las generalizaciones

    - Describir los metodos
    - Describir estados
    - Tratar los requisitos especiales no considerado anteriormente
- *Diseño de un Subsistema:* Sus propositos son:
    - Garantizar que el subsistema sea tan independiente como sea psoible de los demas y de sus interaces (bajo acoplamiento)
    - Garantizar que el subsistema proporcione las interfaces correctas
    - Garantizar el contenido y comportamiento de los subsistemas de acuerdo a las interfaces que proporcionan.