**Modelos de proceso**

* Proceso: colección de actividades de trabajo, acciones y tareas.
* Modelo del proceso: prescribe un flujo de trabajo. Busca generar estructura y orden en el desarrollo del SW.
* Etapas: Comunicación (especificación), planeación, modelado, construcción y despliegue. Mantenimiento.

**Modelos de proceso prescriptivo o tradicional**

Prescriben un conjunto de elementos del proceso: actividades estructurales, acciones de ingeniería de SW, tareas, productos del trabajo, aseguramiento de la calidad y mecanismos de control del cambio para cada proyecto.

**Modelo de Cascada**

* Se aplica cuando los requerimientos están bien definidos y tienen una estabilidad razonable.
* Cuando deben hacerse adaptaciones o mejoras bien definidas a un sistema ya existente.
* Enfoque sistemático y secuencial.

Variante de Cascada: Modelo en V

* Proporciona una forma de visualizar el modo de asociación de las acciones de verificación y validación con el trabajo de ingeniería inicial.

Problemas al aplicar Cascada:

* Es raro que los proyectos sigan un flujo secuencial.
* No se adapta a los cambios.
* Es difícil para el cliente enunciar en forma explícita todos los requerimientos.
* No se dispondrá de una versión funcional del sistema hasta que el proyecto esté muy avanzado.
* Un error grande sería desastroso.
* Llega a “estados de bloqueo” en los que ciertos miembros del equipo deben esperar a otros para realizar tareas.

**Modelo Incremental**

* Combina flujos de proceso lineal y paralelo. Aplica secuencias lineales de las actividades en forma escalonada.
* Cada secuencia lineal produce un incremento de SW (producto funcional).
* Se aplica cuando hay que dar rápidamente cierta funcionalidad, la cual se aumentará en las entregas posteriores.
* Es útil cuando no se dispone de personal para la implementación completa del proyecto en el plazo establecido.

**Modelo Evolutivo**

* Entregas evolutivas. Permite desarrollar versiones cada vez más completas del SW.
* Se utiliza cuando los requerimientos cambian a medida que avanza el desarrollo. Se comprenden bien los requerimientos básicos, pero los detalles o extensiones del sistema aún están por definirse.

Prototipo

* Técnica que puede usarse en cualquiera de los modelos. Ayuda a reducir riesgos.
* Ayuda a mejorar la comprensión de lo que hay que elaborar cuando los requerimientos no están claros.
* Se centra en la representación de los aspectos del SW que serán visibles para los usuarios finales.

*Problemas*

* No son funcionales. Se puede generar demasiada expectativa en los clientes.
* Por mejorar el diseño, el desarrollador se olvida de la performance y otros aspectos más importantes.

Espiral

* El primer circuito alrededor de la espiral da como resultado el desarrollo de una especificación del producto; las vueltas sucesivas se usan para desarrollar un prototipo y, luego, versiones cada vez más sofisticadas del SW.
* En cada paso evolutivo se marcan puntos de referencia puntuales: combinación de productos del trabajo y condiciones que se encuentran a lo largo de la trayectoria de la espiral.
* Permite aplicar prototipos en cualquier etapa de la evolución del producto.
* Es un enfoque realista para el desarrollo de SW a gran escala.
* El desarrollador y cliente comprenden y reaccionan mejor ante los riesgos en cada nivel de evolución.

**Modelo Concurrente**

* Permite elementos iterativos y concurrentes de cualquiera de los modelos.
* Define una red del proceso. Cada actividad existe simultáneamente con otras. Los eventos generados desencadenan transiciones entre los estados.
* Es apropiado para proyectos en los que se involucran varios equipos de trabajo.
* Proporciona un panorama apropiado del estado actual del proyecto.

**Modelos de proceso especializado**

Tienen muchas de las características de uno o más de los modelos tradicionales. Tienden a aplicarse cuando se elige un enfoque de ingeniería de SW especializado o definido muy específicamente.

Desarrollo basado en componentes

* Evolutivo e iterativo. Incorpora muchas de las características del modelo espiral.
* Se construye a partir de fragmentos de SW prefabricados. Lleva a la reutilización del SW.

Modelo de métodos formales

* Agrupa actividades que llevan a la especificación matemática formal del SW. Permite especificar, desarrollar y verificar un sistema muy complejo por medio del empleo de una notación matemática rigurosa.
* Lo ambiguo, incompleto e inconsistente se descubre y corrige con más facilidad.
* Para SW de primera calidad en seguridad y para los que sufrirían graves pérdidas económicas si ocurrieran errores.

*Problemas*:

* Consume mucho tiempo y es caro.
* Se requiere mucha capacitación.
* Es difícil utilizar los modelos como mecanismo de comunicación para clientes sin complejidad técnica.

Desarrollo de SW orientado a aspectos

* Consiste en definir “aspectos” que expresan preocupaciones del cliente que afectan múltiples funciones, características e información del sistema.

**Proceso Unificado de desarrollo de SW (PU)**

* Reconoce la importancia de la comunicación con el cliente y los métodos directos para describir su punto de vista del sistema.
* Hace énfasis en la importancia de la arquitectura del SW.
* Sugiere un flujo del proceso iterativo e incremental.
* Es genérico y debe ser “instanciado”.

Axiomas:

* Dirigido por requisitos y riesgo.
* Centrado en arquitectura: cuál es la forma o estructura que el SW debe tener.
* Iterativo e incremental.

Iteraciones:

* Cada iteración es un subproyecto con actividades secuenciales (planificación, análisis, diseño, construcción, integración, pruebas y versión del sistema).
* En cada iteración se genera una LB: versión del conjunto de artefactos generados en la iteración, revisados y aprobados. Cada LB proporciona una base para mayor revisión y desarrollo y se puede cambiar solamente por medio de procedimientos formales de configuración y gestión del cambio.

**Estructura**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fases** | **Comienzo** | **Elaboración** | **Construcción** | **Transición** |
| **Hitos** | Objetivos del proyecto | Arquitectura del sistema | Posibilidad operacional inicial | Versión del producto |

Las fases terminan cuando se cumplen los hitos. Cada fase puede tener varias iteraciones. Al finalizar, se obtiene el subsistema.

Comienzo

* Comunicación con el cliente y planeación.
* Identificar los requerimientos y se describen con CU preliminares.
* Proponer una arquitectura aproximada (subsistemas principales y la función y rasgos que tienen).
* Desarrollar un plan de proyecto. Identificar recursos, evaluar riesgos, definir actividades y establecer una base para las fases que se van a aplicar a medida que avanza el incremento del SW.
* Establecer la viabilidad técnica y económica.
* Crear un caso de negocio: recursos, alcance.
* Foco de atención: Workflows de Requisitos y Análisis.
* Entregables: Documento con requisitos, características y restricciones. Modelo inicial de CU. Documento de análisis de riegos. Prototipos. Documento inicial de arquitectura.

Elaboración

* Actividades de comunicación y modelado.
* Mejorar y ampliar los CU.
* Definir atributos de calidad (RNF).
* Mejorar la arquitectura: modelos de CU, de requerimientos, del diseño, de la implementación y del despliegue.
* Crear una LB ejecutable de la arquitectura. Demuestra la viabilidad de ésta, pero no proporciona todas las características y funciones que se requieren para usar el sistema.
* Revisar el plan para asegurar que el alcance, riesgos y fechas de entrega siguen siendo razonables.
* Realizar un plan detallado de la fase de construcción.
* Formular la oferta: recursos, tiempos, equipamientos, personas, costos.
* Foco de atención: Workflows de Requisitos, Análisis, Diseño para la arquitectura, Implementación y Prueba (sin generar SW) para formular la oferta.
* Entregables: Línea base ejecutable de la arquitectura. Modelo UML estático, dinámico y CU. Documentos actualizados y plan de proyecto.

Construcción

* Completar los modelos de requerimientos y diseño, a fin de que reflejen la versión final del incremento de SW.
* Implementar en código fuente todas las características y funciones.
* A medida que se implementan los componentes, diseñar y efectuar pruebas unitarias para cada uno.
* Realizar actividades de integración.
* Evolucionar la LB de la arquitectura.
* Foco de atención: Workflows de Implementación y Prueba.
* Entregables: producto de SW, modelos UML, suite de pruebas, manuales de usuario, descripción de versión y plan de proyecto.

Transición

* Entregar el SW a los usuarios finales para las pruebas beta, quienes reportan los defectos y los cambios necesarios.
* Generar la información de apoyo necesaria para el lanzamiento.
* Preparar los entornos de usuarios y adaptar el SW. Ofrecer una consultoría a usuario.
* Foco de atención: workflows de Implementación (incluye la puesta en funcionamiento) y Prueba (incluye verificar que el SW funcione en el entorno de los usuarios).
* Entregables: producto de SW, plan de soporte de usuario y manuales de usuarios actualizados.

**UML (Lenguaje Unificado de Modelado)**

* Modelar SW y sistemas como un conjunto de objetos que interactúan entre sí.
* Estructura estática: describe qué tipos de objetos son importantes para el modelado y cómo se relacionan.
* Comportamiento dinámico: describe los ciclos de vida de los objetos y cómo interactúan entre sí.

**Especificación de requisitos - Workflow de requisitos**

* Un requisito es una especificación de lo que debería hacer el sistema o lo que se debería implementar.
* La captura de requisitos es un factor crítico de éxito.
* Requisitos funcionales: lo que el sistema debería hacer.
* Requisitos no funcionales: restricciones y propiedades específicas no funcionales del sistema. Clasificación según rendimiento, capacidad, disponibilidad, seguridad y seguimiento de estándares.

**Modelado de CU**

* Forma de documentar requisitos. Es la entrada principal para el modelado de clases.
* Encontrar el límite del sistema candidato, encontrar los actores y encontrar los CU. Repetir esto hasta tener un modelo estable.

Componentes de un modelo de CU

**1. Límite del sistema** (Sujeto).

* Encierra todo lo que pertenece al sistema. Definir qué es parte del sistema y qué externo.
* Está definido por quién o qué utiliza el sistema.

**2. Actores**

* Especifican un rol que adopta una entidad externa cuando interactúa con el sistema.
* Son externos al sistema.
* Cada actor necesita un nombre breve y una descripción.

**3. CU**

* Es algo que el actor quiere que el sistema haga.
* Constituyen el comportamiento del sistema.
* Se inician por un actor.

**4. Glosario**

* Proporciona un diccionario de términos claves de negocio y definiciones.
  + *Sinónimos*: palabras diferentes que significan lo mismo.
  + *Homónimos*: la misma palabra que significa cosas diferentes a diferentes personas.

Realizaciones de CU

* Detallan el comportamiento del CU.
* Se buscan flujos alternativos: Alternativas posibles al flujo principal, errores o interrupciones.

**Modelado de CU**

**Extensiones de CU**

* Include (<<include>>):
  + Permite incluir el comportamiento de un CU en el flujo de otro CU.
  + El CU de inclusión proporciona su comportamiento a su CU base.
  + El CU base no está completo sin todos sus CU de inclusión.
  + Los CU de inclusión suelen no poder instanciarse por sí mismos.
* Extend (<<extend>>):
  + Permite que un CU extienda su comportamiento con otro CU.
  + El CU base está completo sin sus extensiones.

**Generalización**

* Generalización de Actor:
  + Cuando más de un actor se comunican con el mismo conjunto de CU de la misma forma.
  + Los actores descendentes heredan los roles y relaciones a CU albergados por el actor padre.
* Generalización de CU:
  + Se utiliza cuando tiene uno o más CU que son realmente especificaciones o un caso más general.
  + Los CU hijo representan formas más específicas del padre. Los hijos pueden heredar características, añadir nuevas y anular o cambiar las heredadas.

**Workflow de Análisis O.O.**

**Clases de análisis**

* Una clase representa una abstracción de un elemento específico del dominio del problema.
* Es crucial que cada clase tenga un conjunto cohesivo de responsabilidades que coincidan con el propósito de dicha clase y con el elemento del mundo real que representa.
* Todas las clases colaboran entre sí.

Técnicas para encontrar clases de análisis

* Análisis nombre/verbo: analizar los modelos y documentos para encontrar nombres y verbos.
* Análisis CRC (Clase, Responsabilidades y Colaboradores): se usan notas.
* Estereotipos RUP (proceso unificado racional).

Relaciones entre clases: Son conexiones semánticas entre elementos de modelado.

Vínculo: Conexión entre dos objetos que permite enviar mensajes. Si hay un vínculo, hay una asociación entre las clases.

Asociación: Relaciones entre clases. Una asociación entre clases significa que los objetos de una clase pueden albergar referencias a objetos de otra la clase.

Sintaxis de la asociación: Las asociaciones pueden tener un nombre de asociación o de rol, multiplicidad y navegabilidad

Asociaciones Reflexivas: Asociación de una clase consigo misma. Los objetos de la clase tienen vínculos a otros objetos de la misma clase.

Jerarquías de Objetos: Existe un objeto raíz y cualquier otro nodo en la jerarquía tiene exactamente un objeto directamente sobre él.

Redes de Objetos: Cada objeto puede tener muchos objetos directamente conectado a él.

**Clases de asociación**

Surge cuando hay una relación de muchos a muchos entre dos clases con atributos que no se pueden acomodar en ninguna de dichas clases.

**Realizaciones de un CU**

Las clases de análisis modelan la estructura estática de un sistema y las realizaciones de CU muestran cómo las instancias de esas clases interactúan para realizar la funcionalidad del sistema.

La realización de CU se centra en capturar atributos, operaciones y relaciones claves entre las clases de análisis.

Se convierte cada CU en diagramas de clase y diagramas de interacción.

En la realización se utilizan los siguientes elementos:

* Diagramas de clase de análisis: interacciones entre las clases para realizar el CU.
* Diagramas de interacción: interacciones entre las instancias que realizan el CU.
* Requisitos especiales: se pueden descubrir nuevos requisitos que se deben capturar.
* Mejora del CU: se puede descubrir nueva información, por lo que el CU original se debe actualizar.

La realización es un proceso iterativo y de mejora continua. Es probable descubrir nuevos requisitos o tener que realizar modificaciones. Se deben mantener los documentos existentes actualizados.

Interacciones

Son sencillas unidades de comportamiento de un clasificador de contexto. La interacción se muestra en análisis con diagramas de secuencias y de comunicación. En diseño con diagramas de visión de interacción y diagramas de tiempo.

**Diagramas de secuencia**

Enfatizan la secuencia ordenada en el tiempo de eventos entre líneas de vida.

Líneas de vida

* Representa cómo una instancia de un clasificador participa en la interacción.
* Elementos: nombre, tipo (clasificador representado) y selector (condición para seleccionar una instancia).
* Se comunican a través de mensajes.

La interacción describe cómo las instancias del clasificador interactúan de una forma general.

Mensajes

Representan un tipo específico de comunicación entre dos líneas de vida en una interacción.

Esta comunicación puede implicar:

* Invocar una operación mediante un mensaje de llamada.
* Crear o destruir una instancia mediante un mensaje de creación/destrucción.
* Enviar una señal.

Tipos de mensajes

* Envío de Mensaje:
  + Síncrono: el emisor espera a que el receptor acabe de ejecutar la operación solicitada.
  + Asíncrono: el emisor no espera, continúa inmediatamente con el siguiente paso.
* Retorno de mensaje.
* Creación de objeto.
* Destrucción de objeto: Se destruye la instancia de clasificador.

Activaciones

Cuando una línea de vida ejecuta un mensaje, tiene foco de control o activación.

Un objeto empieza con el foco e invoca una operación sobre otro objeto anidando el foco de control. Este objeto puede invocar una operación sobre otro objeto anidando aún más el foco de control.

**Flujo de control**: movimiento de la activación entre las líneas de vida a medida que progresa la interacción con el tiempo.

**Autodelegación**: una línea de vida se envía un mensaje a sí misma. Se crea una activación anidada.

Invariantes de estado y restricciones

Se usan invariantes de estado para mostrar los cambios de estado de una instancia.

Una restricción sobre la línea de vida indica algo que debe ser verdadero sobre instancias desde ese punto en adelante. Se puede indicar cualquier tipo de restricción.

Fragmentos combinados y operadores

* Los diagramas de secuencia se pueden dividir en áreas o fragmentos combinados.
* Todo fragmento combinado tiene un operador, uno o más operandos y cero o más condiciones de protección.
* El operador determina cómo se ejecutan sus operandos.
* Las condiciones de protección son expresiones booleanas que determinan si sus operandos se ejecutan.

**Diagramas de comunicación**

Enfatizan las relaciones estructurales entre objetos, es decir, cómo se conectan las líneas de vida.

Similar a los de secuencia, pero las líneas de vida son continuas y se conectan mediante vínculos que proporcionan canales de comunicación para los mensajes. Se enumeran los mensajes para indicar secuencia y anidamiento.

Iteración en diagramas de comunicación

Se pueden incluir iteraciones utilizando una expresión de iteración. Esto implica un especificador de iteración y una cláusula de iteración (opcional).

Ramificación

Se puede modelar ramificación al añadir condiciones de protección a los mensajes (solo se envía cuando la condición se cumple).

**Workflow de Diseño**

Ocurre al final de la fase de elaboración y principio de la fase de construcción.

Se representan o modelan los requerimientos para construir el SW. Agrupa el conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema de alta calidad.

Proporciona detalles sobre arquitectura del SW, estructuras de datos, interfaces y componentes para su implementación.

Los modelos de diseño constan de subsistemas, clases, interfaces, realizaciones de CU y un diagrama de despliegue.

Representaciones y refinamiento

El diseño comienza con una representación en un nivel alto de abstracción (arquitectura), y a medida que tienen lugar las iteraciones, las mejoras posteriores conducen a niveles menores de abstracción (código).

Un programa se elabora por medio del refinamiento sucesivo de los detalles del procedimiento.

Del Análisis al Diseño

* El modelo de diseño está basado en el de análisis y se puede considerar una mejora de ello ya que todos los artefactos están bien formados y deben incluir detalles de implementación.
* El trabajo de diseño es alimentado por el modelo de requerimientos, manifestado por elementos basados en el escenario, en la clase, orientados al flujo, y del comportamiento.
* El empleo de la notación y de los métodos de diseño produce diseños de los datos o clases, de la arquitectura, de la interfaz y de los componentes.

Metas de Diseño

* Debe implementar todos los requerimientos explícitos contenidos en el modelo de requerimientos y dar cabida a todos los requerimientos implícitos que desean los participantes.
* Debe ser una guía legible y comprensible para programadores, testers y soporte posterior.
* Debe proporcionar el panorama completo del SW, y abordar los dominios de los datos, las funciones y el comportamiento desde el punto de vista de implementación.

Lineamientos de calidad de Diseño

1. Debe exhibir una arquitectura.
2. Debe ser modular.
3. Debe contener distintas representaciones para los datos, arquitecturas, interfaces y componentes.
4. Debe conducir a estructuras de datos apropiadas para las clases.
5. Debe llevar a componentes funcionalmente independientes.
6. Debe llevar a interfaces simples.
7. Debe ser realizado utilizando un método repetible y motivado por la información obtenida durante el análisis.
8. Debe ser representado utilizando una notación que efectivamente comunique su significado.

Atributos de calidad

* **Funcionalidad**: características y capacidades del programa, generalidad de las funciones y seguridad general.
* **Usabilidad**: factores humanos, estética general, consistencia y documentación.
* **Confiabilidad**: frecuencia y gravedad de las fallas, exactitud de los resultados, tiempo medio de fallas, capacidad de recuperación.
* **Rendimiento**: velocidad de procesamiento, tiempo de respuesta, uso de recursos y eficiencia.
* **Mantenibilidad**: capacidad del programa para ser ampliable (extensibilidad), adaptable y servicial. Además, que pueda probarse, ser compatible y configurable y que cuente con la facilidad para instalarse en el sistema y para que se detecten los problemas.

Función, Forma y Desarrollo

* Un sistema de SW se construye basado en una especificación de su forma, la que satisface una función.
* **Forma:** especificación de diferentes atributos de calidad que satisfacen la función.
* **Función:** necesidad, propósito, utilidad o uso pretendido del sistema.
* **Diseño (arquitectónico):** une la función y la forma.

Forma y Diseño de Interacción

* **Diseño de interacción**: afecta directamente al usuario final.
* **Diseño de programa**: diseño de la aplicación.
* El diseño de interacción pobre contribuye a la ‘fricción cognitiva’: resistencia del usuario frente a un SW complejo.
* La interacción con aplicaciones de SW es alta en fricción cognitiva, debido a la gran cantidad de estados que el sistema puede tener, y en base a esto reaccionar.

Construcción

* La calidad de un sistema está basada en el diseño y en la elección de la tecnología utilizada en la implementación.
* El diseño de SW debe ser realizable:
  + Debe ser posible aplicar las prácticas y tecnologías de ingenierías disponibles.
  + Debe ser posible construir el sistema dada la disponibilidad de recursos: tiempo, personal, costos, sistemas existentes y componentes.

Problemas, Obstáculos y Soluciones

El diseño es una actividad de encontrar o crear soluciones a problemas dado un conjunto de obstáculos a sobrepasar.

**Conceptos de Diseño**

Abstracción

* Concentrarse en un problema al mismo nivel de generalización, independientemente de los detalles irrelevantes de bajo nivel. Permite trabajar con conceptos que son familiares al entorno del problema.
* **Abstracción de procedimiento:** instrucciones que tienen una función específica y limitada. Ejemplo: abrir.
* **Abstracción de datos:** conjunto de datos con nombre que describe a un objeto de datos. Ejemplo: ventana.

Arquitectura

Es la estructura general del SW y la forma en las que da integridad conceptual a un sistema.

Deben especificarse:

* **Propiedades estructurales**: Define los componentes de un sistema, su agrupación e interacción.
* **Propiedades extrafuncionales**: Descripción de la forma en la que la arquitectura satisface los requerimientos de desempeño, capacidad, confiabilidad, seguridad y adaptabilidad, así como otras características del sistema.
* **Familias de sistemas relacionados**: El diseño debe basarse en patrones repetibles. Debe tener la capacidad de reutilizar bloques de construcción arquitectónica.

Patrones de diseño

Describe una estructura de diseño que resuelve un problema particular dentro de un contexto específico.

Cada patrón tiene una descripción que permite determinar si es aplicable, si puede volverse a usar y si sirve como guía para desarrollar otro patrón.

División de problemas - Modularidad

* Cualquier problema complejo puede manejarse con más facilidad si se subdivide en elementos independientes. Lleva más tiempo y esfuerzo resolver un único problema complejo.
* Los módulos se integran para satisfacer los requerimientos del problema.
* El esfuerzo de desarrollar un módulo disminuye conforme aumenta el número total de módulos. Sin embargo, a medida que se incrementa el número de módulos, el esfuerzo asociado con su integración también aumenta.

Ocultamiento de Información

* Hacer módulos en los que la información contenida sea inaccesible para los que no necesiten de ella.
* Es menos probable que los errores inadvertidos introducidos durante la modificación se propaguen a distintos sitios dentro del SW.

Independencia Funcional

* Es resultado directo de la modularidad, abstracción y ocultamiento de información.
* Debe diseñarse el SW de manera que cada módulo resuelva un subconjunto específico de requerimientos y tenga una interfaz sencilla cuando se vea desde otras partes de la estructura del programa.
* Es clave para el buen diseño y calidad del SW.
* Se evalúa con el uso de dos criterios cualitativos:
  + **Cohesión**: Indicador de la funcionalidad de un módulo. Un módulo cohesivo debe ejecutar una sola tarea.
  + **Acoplamiento**: Indicación de la interconexión entre módulos. Depende de la complejidad de la interfaz, del grado en el que se referencia y de qué datos pasan a través de la interfaz.

Mínimo acoplamiento: SW que más fácil de entender y menos propenso a la propagación de errores.

Rediseño

Técnica de reorganización que simplifica el diseño o código de un componente sin cambiar su función o comportamiento.

Se examina el diseño existente en busca de redundancias, elementos de diseño no utilizados, algoritmos ineficientes o innecesarios, estructuras de datos mal construidas o inapropiadas y cualquier otra falla del diseño.

Resumen

* Una solución al problema de diseño de SW es la cuidadosa planificación y ejecución sistemática de un proceso de diseño y la producción de artefactos de diseño (modelos y especificaciones).
* **Diseño Funcional**: Diseño de la faceta externa o diseño de interacción. Aspecto percibido por los usuarios.
* **Diseño Estructural**: Diseño físico interno del SW. Los métodos de diseño se aplican para descubrir nuevas soluciones y evaluar soluciones existentes.
* **Diseño Detallado**: Involucra la especificación del SW para las actividades posteriores de implementación y testing. Debe realizarse en diferentes niveles de abstracción.
* El SW es inherentemente complejo.

**Atributos de Calidad de SW**

Son los RNF sobre la forma que debe tener el SW. Condicionan la manera en que las funciones serán llevadas a cabo.

La calidad debe ser considerada en todas las fases de diseño, implementación y distribución.

Categorías de calidades: calidad en tiempo de ejecución, calidades estáticas, de negocio y de diseño/arquitectura.

Interrelaciones entre las calidades

* Alcanzar los atributos de una categoría puede no decir nada acerca de los atributos de otra categoría.
* Alcanzar los atributos dentro de una categoría no dice nada acerca de otros atributos dentro de la misma categoría.
* Alcanzar un atributo de calidad nunca puede hacerse de forma aislada.
* Alcanzar cualquier atributo de calidad puede tener un efecto positivo o negativo sobre otros atributos de calidad.
  + Seguridad y tolerancia a fallas existen en un estado de tensión mutua.
  + Confiabilidad y performance.

Consideraciones:

* La arquitectura es crítica para alcanzar muchas de las cualidades de interés, y estas cualidades deben ser diseñadas y evaluadas a nivel arquitectónico.
* Algunas cualidades no son sensitivamente arquitectónicas, e intentar alcanzarlas a través de la arquitectura será infructuoso.

Atributos de Calidad - Cualidades en tiempo de ejecución

* **Funcionalidad:** Habilidad del sistema de hacer el trabajo para el cual fue creado.
  + Se alcanza a través de la interacción, cooperación y sincronización de los componentes.
  + Es ortogonal al diseño de SW (arquitectura).
  + El diseño del SW condiciona la asignación de funcionalidad a la estructura.
  + Es independiente de la arquitectura. Puede ser alcanzada a través de diferentes estructuras posibles.
* **Disponibilidad:** Medida del tiempo en que el sistema está operativo y ejecutando correctamente.
  + Analiza las fallas y defectos del sistema y sus consecuencias.
  + La falla es observable por el usuario. El defecto se encuentra en el código.
* **Modificabilidad**: Facilidad del SW para adecuarse a los cambios.
  + Cambio: agregación, modificación, eliminación, reestructuración.
  + Problema netamente arquitectónico: afectado por las características de los componentes de SW.
  + Cualquier aspecto del sistema puede ser cambiado.
  + Los cambios se realizan durante la implementación, compilación, construcción, configuración o ejecución.
  + Lo puede realizar un desarrollador, usuario final o administrador de sistema.
  + Una vez que el cambio ha sido especificado, debe ser diseñado, implementado, testeado y distribuido. Toma tiempo y dinero, lo que puede ser medido.
  + Portabilidad: Un tipo especial de modificabilidad. Habilidad del sistema para ejecutar en diferentes entornos de computación.
* **Performance**: Mide las respuestas del sistema.
  + Cantidad de transacciones por unidad de tiempo o tiempo requerido para completar una transacción.
  + Es una función de cuanta comunicación e interacción hay entre las componentes.
* **Usabilidad:** Qué tan fácil es para un usuario realizar una tarea (facilidad de uso) y qué soporte provee el sistema.
  + Aspectos de usabilidad: aprendizaje, eficiencia, impacto de errores, manejo de errores, adaptación del sistema a las necesidades del usuario, asistencia, confianza y satisfacción.

**Diagramas de actividad**

Son diagramas de flujo orientados a objetos que permiten modelar un proceso como una actividad que consta de una colección de nodos conectados por extremos.

Está centrado en comunicar un aspecto específico de un comportamiento dinámico de un sistema.

Las actividades se anexan a: CU, Clases, Interfaces, Componentes, Colaboraciones y Operaciones. También se pueden utilizar para modelar procesos de negocio y flujos de trabajo.

Actividades

* Son redes de nodos conectados por extremos. Categorías de nodos: nodos de acción, de control y de objeto.
* Los extremos o flechas representan flujo por la actividad. Existen 2 categorías: flujos de control y flujos de objeto.
* Las actividades pueden tener precondiciones y postcondiciones. Las acciones dentro de la actividad también pueden tener. Las actividades comienzan con un nodo inicial y pueden terminar con uno o más nodos finales.

Semántica de actividad

Los diagramas de actividad modelan comportamiento utilizando el flujo de tokens. Pueden representar: flujo de control, datos u objetos.

El estado del sistema en cualquier punto en el tiempo está determinado por la disposición de sus tokens.

Particiones de actividad

Se pueden dividir las actividades en particiones para hacer más legible el diagrama. Cada partición representa una agrupación de alto nivel de acciones relacionadas.

Nodos de acción

* Unidades de trabajo atómicas.
* Se ejecutan cuando existe un token simultáneamente en cada uno de sus extremos de entrada y se cumplen todas sus precondiciones locales.
* Después de ejecutarse, ofrecen tokens simultáneamente en todas sus salida cuyas postcondiciones se cumplen.

Tipos de acción

* Nodo de acción de llamada: invocar una actividad, comportamiento u operación. Puede enviar un nodo de acción de señal, nodo de acción de aceptar evento o nodo de acción de aceptar evento de tiempo.
* Enviar señal
* Nodo de acción de aceptar evento.
* Nodo de acción de aceptar evento de tiempo

Nodos de control

* Gestionan el flujo de control dentro de una actividad. El token de entrada se ofrece a todas las salidas.
* Cada salida está protegida por una condición de protección, las cuales deben ser mutuamente excluyentes.

Nodos de objeto

* Indican que las instancias de un clasificador especial están disponibles en un punto específico en la actividad.
* Representan instancias de un clasificador. Los extremos de entrada y salida son flujos de objeto. Los propios objetos se crean y consumen por los nodos de acción.
* Ofrece un token en todas sus salidas simultáneamente. El token no se replica en los extremos y uno sólo lo toma.
* Se pueden utilizar para recopilar objetos de múltiples flujos entrantes de objeto o para distribuir objetos a múltiples flujos salientes.
* Pueden seleccionar objetos de los extremos de entrada de acuerdo a algún criterio definido.
* Se pueden poner objetos sin duplicados.
* Pueden representar objetos en un estado determinado.

**Clases de Diseño**

Son clases cuyas especificaciones se han completado hasta tal nivel que se pueden implementar. Tienen todos sus atributos definidos, tipos, métodos, operaciones y parámetros.

Una clase está completa si está suficientemente detallada para servir como base para crear código fuente.

Clases de diseño bien creadas

* **Completa y suficiente**
  + Completitud: Proporcionar a otras clases lo que podrían esperar razonablemente.
  + Suficiencia: Todas las operaciones están centradas en realizar la finalidad de la clase.
* **Sencilla**
  + Las operaciones deben ofrecer un solo servicio.
  + No debe haber diferentes formas de hacer lo mismo.
  + Conjunto de operaciones los más sencillo y pequeño posible.
* **Alta cohesión**
  + Debe modelar un solo concepto abstracto y debería tener un conjunto de operaciones que soporten el propósito de la clase.
  + Permite que sean fáciles de entender, reutilizar y mantener.
  + Tiene un pequeño conjunto de responsabilidades que están íntimamente relacionadas.
* **Bajo acoplamiento**
  + Una clase debería estar asociada con suficientes clases para permitirle realizar sus responsabilidades, y solamente debería asociar clases si existe un verdadero vínculo semántico entre ellas.
  + El acoplamiento debe ser minimizado tanto como se pueda.
  + Cierto acoplamiento es bueno y deseable. Un alto acoplamiento dentro de un subsistema es generalmente aceptable ya que esto indica una alta cohesión dentro del componente.
  + Solamente compromete la arquitectura cuando el acoplamiento es entre subsistemas.

Herencia

La herencia tiene ciertas características no deseables:

* Es la forma mayor de acoplamiento posible entre dos o más clases.
* La encapsulación es débil dentro de una jerarquía de clase. Los cambios en la clase base provocan cambios en las subclases y tienen un impacto importante sobre otras clases en el sistema.
* Es un tipo muy inflexible de relación puesto que en tiempo de ejecución son relaciones fijas.

Herencia vs realización de interfaz

* **Herencia**
  + Mediante ésta obtiene:
    - Interfaz: las operaciones públicas de las clases base.
    - Implementación: los atributos, relaciones, operaciones protegidas y privadas de las clases base.
  + Utilizarla únicamente cuanto está preocupado por heredar algunos detalles de implementación.
* **Realización de la interfaz**
  + Mediante ésta obtiene:
    - Una interfaz: conjunto de operaciones, atributos y relaciones que no tienen implementación.
  + Es de utilidad siempre que desee definir un contrato, pero no heredar detalles de implementación.
  + No proporciona reutilización de código. Asegura que la implementación de clases se adapte a los mismos.
  + Mucho más flexible y robusta que la herencia.

**Diseño de relaciones**

Convertir las asociaciones en relaciones de agregación o composición.

Todas las asociaciones de diseño deben tener: navegabilidad, multiplicidad en ambos extremos y nombre o rol.

Agregación

* Relación todo-parte en la que el todo se compone de muchas partes. El todo utiliza los servicios de la parte. Si solamente tiene navegabilidad desde el todo a la parte, la parte ni siquiera sabe que es parte de un todo.
* El conjunto puede existir independientemente de las partes o no.
* Las partes pueden existir independientemente del todo pero el todo está incompleto si falta alguna de las partes.
* La agregación es transitiva y asimétrica (un objeto nunca puede ser parte de sí mismo).

Composición

* Relación todo-parte y es tanto transitiva como asimétrica.
* Las partes no tienen vida independientemente fuera del todo.
* Cada parte pertenece al menos a un y sólo a un todo. No existe la propiedad compartida de una parte.
* El todo tiene responsabilidad de creación y destrucción de todas sus partes.
* El conjunto puede liberar partes, siempre y cuando la responsabilidad para ellas se asuma por otro objeto.
* Si se destruye el conjunto, debe destruir todas sus partes o pasar la responsabilidad a algún otro objeto.

Multiplicidad de las asociaciones

* **Asociaciones uno a uno:** Casi siempre es composición. Ver si se pueden fusionar en una sola clase.
* **Asociaciones muchos a uno:** Muchos en el lado del todo y 1 en el lado de la parte. Usar agregación.
* **Asociaciones uno a muchos:** Colección de objetos en el lado parte de la relación. Para implementarlo se utilizan clases de colección.

Colecciones

Es una clase cuyas instancias se especializan en gestionar colecciones de otros objetos. Todas las clases de este tipo tienen operaciones para añadir y eliminar objetos a la colección, recuperar una referencia de un objeto y recorrer la colección.

Para representar la relación de las partes al todo (clase de colección) se puede utilizar composición o agregación.

Mapas (Diccionarios):

* Clases que funcionan como una tabla de BD con dos columnas: clave primaria y valor.
* Cada nodo apunta a dos objetos: el objeto clave y el objeto valor.
* Búsqueda rápida para un objeto clave específico.

Relaciones cosificadas

Relaciones de análisis que no son implementables directamente y se necesitan cosificar.

* Asociaciones muchos a muchos.
* Asociaciones bidireccionales.
* Clases de asociación.

**Fundamentos de orientación a objetos**

Objeto: instancia de una clase que define el conjunto común de características (atributos y operaciones) que se comparten por todas las instancias de esa clase.

Encapsulación: los datos de un objeto están ocultos y solamente se pueden manipular al invocar una de sus funciones.

Operaciones: especificaciones para funciones de objeto creadas en análisis.

Métodos: implementaciones para funciones de objeto creadas en implementación.

Los objetos colaboran para generar el comportamiento del sistema. La interacción implica objetos enviando mensajes. Cuando se recibe un mensaje se invoca la operación correspondiente.

**Interfaces y componentes – Diseño de subsistemas**

* Desglosar el sistema en partes que sean tan independientes como sea posible subsistemas.
* El diseño de subsistemas minimiza el acoplamiento y asegura que cada uno realiza correctamente el comportamiento especificado por sus interfaces.
* La comunicación entre subsistemas se hace a través de sus interfaces.

Interfaz

* Conjunto de características públicas. Define un servicio ofrecido por una clase, subsistema o componente.
* Separa la especificación de funcionalidad de su implementación.
* No se pueden instanciar. Solo definen un contrato. El realizador (clase o subsistema) implementa y acuerda el contrato definido por la interfaz.
* Los atributos y operaciones deben estar especificados completamente. Nunca implica implementación.
* Las clases no se relacionan, sino que se conectan a una interfaz.
* Notación: al estilo de “clase” completa (atributos y operaciones) o abreviada.

Interfaces proporcionadas y requeridas

* **Interfaces proporcionadas:** conjunto de interfaces realizadas por un clasificador.
* **Interfaces requeridas:** conjunto de interfaces requeridas por un clasificador para su operación.
  + La clase requirente, entiende y necesita el protocolo definido por la interfaz.

Interfaces vs herencia: Una solución basada en interfaz es más sencilla y tiene mejor semántica.

Puertos

* Agrupa un conjunto semánticamente cohesivo de interfaces proporcionadas y requeridas.
* Útiles para estructurar las interfaces proporcionadas y requeridas de un clasificador.
* Para conectar los puertos, sus interfaces proporcionadas y requeridas deben coincidir.
* Pueden ser públicos o privados.
* Multiplicidad: Indican la cantidad de instancias del puerto que puede tener un clasificador/componente.

**Desarrollo basado en componentes**

* Las interfaces son la clave para el desarrollo basado en componentes (construir SW a partir de partes).
* Las interfaces especifican un contrato que puede ser implementado por diferentes constructores.
* La colección de subsistemas e interfaces constituyen la arquitectura del SW.

Componente

* Un componente representa una parte modular de un sistema, que encapsula sus contenidos, y cuya manifestación se reemplaza dentro de su entorno.
* Es una parte modular, desplegable y sustituible de un sistema, que incluye la implantación y expone un conjunto de interfaces.
* Su comportamiento externo está definido por sus interfaces proporcionadas y requeridas. Debido a esto, un componente se puede reemplazar por otro que soporte el mismo protocolo.
* Un componente puede tener atributos y operaciones y puede participar en relaciones de asociación y generalización.
* El diagrama de componente puede mostrar componentes, dependencias entre componentes y la forma en la que los clasificadores están asignados a los componentes.

Componentes compuestos

Un componente puede tener una estructura interna. En este caso se delegarán las responsabilidades definidas por sus interfaces a una o más de sus partes internas.

Dependencias entre componentes: Los componentes pueden depender de otros componentes, siempre a través de interfaces.

Subsistemas: Es un componente o unidad lógica que se usa para descomponer sistemas grandes en bloques manejables.

Arquitectura en capas

* Organiza los subsistemas e interfaces en capas donde son semánticamente cohesivos en cada capa.
* Disminuye el acoplamiento entre subsistemas.
* Las dependencias entre capas están mediadas por interfaces.
* Una capa debería requerir interfaces de la capa inferior, y proveer interfaces a la capa superior.
* El patrón básico es una división entre las capas de presentación, la lógica de negocio y las capas de utilidad. Es bastante común subdividir aún más la capa de lógica de negocio.

Ventajas de las Interfaces

* Diseñar para contratos libera el modelo de dependencias de implementación, lo que aumenta su **flexibilidad** y **extensibilidad**.
* Reduce el número de dependencias entre clases, componentes y subsistemas. Controla y disminuye el **acoplamiento**.

Desventajas de las Interfaces

* La flexibilidad se logra con **complejidad**.

**Máquinas de Estado**

* Modelan el comportamiento dinámico de un sistema o de sus clasificadores. Se utilizan para modelar la historia del ciclo de vida de un solo objeto reactivo.
* Número finito de estados. Transiciones de estados en respuesta a eventos.

Elementos:

* **Estado**: condición durante la vida de un objeto durante el cual se cumple una condición, se realiza alguna actividad o se espera la ocurrencia de algún evento.
  + Solo considerar los estados relevantes. Debe existir una diferencia semántica entre los estados.
  + Las acciones de entrada y salida son instantáneas, no se pueden interrumpir.
  + Transiciones internas: permiten capturar el hecho de que algo que merece modelarse ha ocurrido, pero no provoca una transición a un nuevo estado. Se pueden interrumpir.
* **Evento**: especificación de una ocurrencia que tiene ubicación en tiempo y espacio. Activan transiciones.
  + Evento de llamada: Petición de una operación específica a invocarse en una instancia de la clase de contexto.
  + Evento de señal: Paquete de información que se envía asíncronamente entre objetos.
  + Evento de cambio: La acción asociada al evento se realiza cuando una expresión booleana es verdadera.
  + Evento de tiempo: Se indican normalmente por medio de las palabras claves ‘cuando’ y ‘después’.
* **Transición**: movimiento de un estado a otro en respuesta a un evento.
  + Tiene tres elementos opcionales:
    - Eventos: especifican ocurrencias externas o internas que pueden activar la transición.
    - Condición de protección: expresión booleana que debe evaluar en verdadero antes de que pueda ocurrir la transición.
    - Acciones: parte del trabajo asociado con la transición. Ocurre cuando se activa la misma.

Objeto reactivo: es un objeto que proporciona el contexto para una máquina de estado.

* Responden a eventos externos. Pueden generar y responder a eventos internos.
* Tienen un ciclo de vida modelado como una progresión de estados, transiciones y eventos.
* Pueden tener un comportamiento que depende de comportamiento pasado.

Pseudo estado de unión: representan puntos donde se unen o ramifican las transiciones. Cada salida debe protegerse con una condición de protección mutuamente excluyente.

Pseudo estado de opción: Permite dirigir el flujo de acuerdo a condiciones que especifica en sus transiciones de salida. Las condiciones en las transiciones de salida deben ser mutuamente excluyentes.

**Implementación**

* Generar un modelo de Implementación: Transformar un modelo de diseño en código ejecutable.
* Modelo de implementación: Especifica cómo los elementos de diseño se muestran como artefactos y cómo estos se despliegan en nodos.
* **Artefactos:** representan las especificaciones de los elementos del mundo real o elementos que tienen una representación concreta en SW. Se despliegan en nodos.
* **Nodos:** representan recursos de HW o entornos de ejecución en las que se despliegan los artefactos para su ejecución.

Diagrama de Despliegue

* Muestra cómo el SW se desplegará sobre HW y cómo se conecta ese HW. Mapea la arquitectura de SW con la arquitectura física que lo ejecuta
* **Despliegue:** Proceso de asignar artefactos a nodos o instancias de artefactos a instancias de nodos.
  + **Forma de descriptor**: contiene nodos, relaciones entre nodos y artefactos.
  + **Forma de instancia:** contiene instancias de nodos, relaciones entre instancias de nodo e instancias de artefacto.
* Nodos anidados:
  + Formato Descriptor: son buenos para modelar un tipo de arquitectura física.
  + Formato Instancia: son buenos para modelar un despliegue real de esa arquitectura en un sitio determinado.
* Diagramas de despliegue estereotipados
  + Se utilizar símbolos. Es útil para ver la infraestructura computacional.

**Análisis Estructurado**

* Es una técnica de modelado, los cuales reflejan el flujo y contenido de la información.
* Se parte el sistema funcionalmente.
* Se establece la esencia del problema.

Modelo de flujo de la información

* **Modelo**: Entidad externa 🡪 Información de entrada 🡪 Sistema basado en computadoras 🡪 Información de salida 🡪 Entidad externa.
* **Objetivo:** visión funcional o de proceso del sistema.
* Se puede crear un modelo de flujo para cualquier sistema de computadora, independientemente del tamaño y de la complejidad.
* Los principales modelos que se generan son los Diagramas de Flujo de Datos y Diagrama Entidad Relación.

Proceso de Análisis Estructurado

Relevamiento 🡪 Modelo ambiental 🡪 Modelo preliminar de Comportamiento: Especificación de proceso y Diccionario de datos.

Modelo Ambiental

* La labor más difícil es determinar qué es parte del sistema y qué no.
* Primer modelo: definir las interfaces entre el sistema y el resto del ambiente.
* Elementos para definir el ambiente:
  + Declaración de propósitos: declaración textual, breve y concisa del propósito del sistema.
    - Explicitando lo que NO se va a hacer. Este es un aspecto importante desde el punto de vista ‘contractual’ y de definición de ‘alcances’.
  + Diagrama de Contexto: caso especial del diagrama de flujo de datos, en donde una sola burbuja representa todo el sistema.

Enfatiza varias características importantes del sistema:

* + - Las personas, organizaciones y sistemas con los que se comunica el sistema. (Entidades Externas)
    - Los datos que el sistema recibe del mundo exterior y que deben procesarse. (Flujos de Entrada)
    - Los datos que el sistema produce y que se envían al mundo exterior. (Flujos de Salida)
    - La frontera entre el sistema y el resto del mundo.
  + Lista de Acontecimientos: lista narrativa de los “estímulos” que ocurren en el mundo exterior a los cuales el sistema debe responder.

Modelo Preliminar de Comportamiento

* Es el modelo del comportamiento final que el sistema debe tener para manejar con éxito el ambiente.
* Dibujar un DFD con un proceso para la respuesta del sistema ante cada evento de ‘estímulo’.

Modelo Ambiental – Complementos

* Especificación de Procesos
* Diccionario de Datos

Modelo Esencial

Definición: es un modelo de lo que el sistema debe hacer para satisfacer los requerimientos del usuario, diciendo lo mínimo posible (de preferencia nada) acerca de cómo se implantará.

* ¿Qué tipo de modelo debemos construir? ¿Debemos modelar completamente el sistema actual? ¿Debemos modelar una propuesta para el sistema?
* Premisa: “Si estamos estudiando/analizando un sistema es porque el mismo presenta problemas, y en consecuencia debe ser modificado”.
* Recomendación:
  + El analista evite modelar el sistema actual de ser posible.
  + Comenzar tan pronto como sea posible, a desarrollar un modelo del nuevo sistema que el usuario desea.

**Diagrama de Flujo de Datos (DFD)**

* Herramienta de modelado de procesos.
* Representa el flujo de la información y las transformaciones que se aplican a los datos al moverse desde la entrada hasta la salida.
* Permite visualizar un sistema como una red de procesos funcionales conectados entre sí por “conductos” y “tanques de almacenamientos” de datos.
* A medida que la información se mueve a través del software, es modificada por una serie de transformaciones.

Elementos

* Entidad externa: productor o consumidor de información. Reside fuera de los límites del sistema.
* Proceso: transformador de información. Reside dentro de los límites del sistema.
* Elemento de datos o una colección de elementos de datos en movimiento (Flujo).
* Depósito de datos que se guardan para ser usados por uno o más procesos.

Sincronismo

* Los procesos en los DFDs no deberían relacionarse directamente para respetar el 'asincronismo' requerido.
* Se comunican a través de los almacenes: Un proceso escribe, otro lee.
* De ser necesario, se indica el sincronismo (Normalmente en los niveles detallados de DFD)

Terminadores/Entidades Externas

* Son externos al sistema; los flujos que los conectan representa la interfaz.
* No se puede cambiar el contenido de un terminador o la manera en la que trabaja.
* Las relaciones que existan entre los terminadores no se muestran en el modelo de DFD.

Todo DFD:

* Comienza con la llegada de un ‘evento/flujo’, anunciado por una ‘entidad externa’.
* Termina con un ‘flujo de datos’ hacia una ‘entidad externa’ o almacén de datos.

**Diseño Estructurado**

Diseño Modular

* Descomposición funcional sistematizada.
* Los sistemas constituidos por módulos son más manejables. cada módulo es independiente de manera que pueden sacarse del sistema, cambiarse, y reponerse sin afectar el resto del sistema.
* La tarea del diseñador es formar los módulos y diseñar sus interconexiones para minimizar la interconexión entre estos, y a la vez reducir la posibilidad la propagación de errores.
* Un diseño modular reduce la complejidad, facilita los cambios, y produce una implementación más sencilla, permitiendo el desarrollo paralelo.

Fases del Diseño Estructurado

* Representar el sistema como un Diagrama de Flujos de Datos (DFD).
* Estructurar el sistema como jerarquías de procesos (DE).
* Realizar Análisis transformacional.
* Realizar Análisis transaccional.
* Verificar proyecto y reestructurar: cohesión, acoplamiento, tamaño de interfaces, etc.
* Descomposición de procesos: refinamiento sucesivo, factorización.
* Preparar para la implementación.

Diagramas Estructurados (DE)

Describen una arquitectura de programas a través de una jerarquía de llamadas a módulos.

* Módulo: representación de una unidad de software, que ejecutará alguna función/proceso.
* Comunicación entre módulos: la estructura es de invocación de módulos.
* Los módulos pueden intercambiar datos o señales.
* Reglas de control para un DE:
  + Hay sólo un módulo raíz.
  + El control es pasado hacia abajo nivel por nivel a los otros módulos.
  + El control siempre es devuelto al módulo invocante. Cuando la ejecución del programa finaliza, el control regresa a la raíz.
  + Hay a lo sumo una relación de control entre dos módulos. (Si A invoca a B, B no puede invocar a A).
* Secuencia de ejecución de los módulos: de arriba hacia abajo y de izquierda a derecha.
* Los DE no muestran ni iteración ni selección por defecto, pero pueden incorporarse simbologías para estos casos.

Derivación de DFD a DE

1. Establecer el tipo de flujo de información.
2. Determinar los límites del flujo.
3. Convertir el DFD en la estructura del programa.
4. Definir la jerarquía de control mediante factorización.
5. Refinar la estructura resultante usando medidas y heurísticas de diseño (acoplamiento y cohesión).

**Reingeniería de negocios: el camino del cambio**

* Es la revisión fundamental y el rediseño radical de procesos para alcanzar mejoras espectaculares en medidas críticas y contemporáneas de rendimiento, tales como costos, calidad, servicio y rapidez.
* Significa abandonar procedimientos establecidos hace mucho tiempo y examinar otra vez desprevenidamente el trabajo que se requiere para crear el producto o servicio de una compañía y entregarle algo de valor al cliente.
* Tiene como objetivo mejorar el rendimiento de la organización de forma drástica.
* Es el rediseño rápido y radical de los procesos estratégicos de valor agregado y de los sistemas, las políticas y las estructuras organizacionales que los sustentan, para optimizar los flujos del trabajo y la productividad de una organización.
* Se buscan avances trascendentales. Al rediseñar los procesos, es posible que se deban romper viejas tradiciones.

Palabras clave:

* Fundamental: La ingeniería determina primero qué debe hacer una compañía; luego, cómo debe hacerlo. No da nada por sentado. Se olvida por completo de lo que es y se concentra en lo que debe ser.
* Radical: rediseñar radicalmente significa descartar todas las estructuras y los procedimientos existentes e inventar maneras enteramente nuevas de realizar el trabajo.
* Espectacular: la reingeniería se trata de dar saltos gigantescos en rendimiento. Se debe apelar a la misma cuando exista la necesidad de volar todo: la mejora espectacular exige volar lo viejo y cambiarlo por algo nuevo.

Etapas de la implantación de la reingeniería

* Inducción a la Reingeniería.
* Identificación y análisis de oportunidades de rediseño.
* Rediseño de procesos.
* Implantación de los procesos rediseñados.

Factores críticos para la implantación

* Compromiso de la alta dirección
* Liderazgo
* Comunicación
* Participación de los empleados
* Equipos

Qué no es la Reingeniería

* No es lo mismo que la automatización.
* No es lo mismo que la reingeniería de software, que significa reconstruir sistemas obsoletos de información con tecnología más moderna.
* Rediseñar no es lo mismo que reorganizar una compañía, reducir el número de niveles o hacerla más plana.

**Reconstrucción de los Procesos**

Proceso: Un proceso es una serie de actividades relacionadas entre sí que convierten insumos en productos.

Flujo de Proceso: Es la ruta que debe seguir un proceso para poder efectuarse.

Importancia de los procesos

* Los procesos pueden ser eficientes o ineficientes. La Reingeniería de procesos se encarga de detectar los puntos más vulnerables dentro de la organización para fortalecerlos.
* Primero es necesario descomponer los procesos en Estratégicos y No estratégicos.
* Los procesos estratégicos son aquellos que proporcionan los medios para alcanzar los objetivos de una compañía.
* El rediseño está basado en darle a las actividades valor agregado y eliminando los excesos y sobrantes. El cambio debe ser rápido y en forma radical.

Metodología

* Es una manera sistemática o claramente definida de alcanzar un fin.
* Se diseña para guiar la reingeniería de procesos ya que tiene el objetivo de obtener resultados óptimos en corto tiempo.

Metodología Rápida Re

* Preparación: Relevamiento de las metas y objetivos.
* Identificación: Desarrollar un modelo del negocio con procesos orientados al cliente; identificar los procesos estratégicos y críticos.
* Visión: Buscar oportunidades de avance decisivo en los procesos; analizarlos y estructurarlos como visiones de cambio radical.
* Solución: Diseño técnico y diseño social.
* Transformación: Realizar las visiones de proceso, lanzando versiones piloto y de plena producción de los nuevos procesos.

Características comunes en procesos de negocios rediseñados

* Varios oficios se combinan en uno.
* Los trabajadores toman decisiones.
* Los pasos del proceso se ejecutan en orden natural.
* Los procesos tienen múltiples versiones para diferentes situaciones.
* El trabajo se realiza en el sitio razonable.
* Se reducen las verificaciones y los controles.
* La conciliación se minimiza.
* Un gerente de caso ofrece un solo punto de contacto.
* Prevalecen operaciones híbridas centralizadas-descentralizadas.

Roles que llevan a cabo la reingeniería

* Líder, dueño del proceso, equipo de reingeniería, comité directivo, zar de reingeniería.
* El líder nombra al dueño del proceso, quien reúne el equipo de reingeniería para rediseñar el proceso con ayuda del zar y bajo los auspicios del comité directivo.