CAPÍTULO 3
DISEÑO DE SOFTWARE

ACRÓNIMOS

ADL Lenguajes de Descripción de Arquitecturas.
CRC Tarjeta Clase-Responsabilidades-Colaboradores
ERD Diagrama Entidad-Relación
IDL Lenguaje de Descripción de Interfaz
DFD Diagrama de Flujo de DatosData Flow Diagram
PDL Pseudo-código y lenguaje de Diseño de Programa
CBD Diseño Basado en Componentes

INTRODUCCIÓN

El diseño se define en [IEEE610.12-90] como "el proceso para definir la arquitectura, los componentes, los interfaces, y otras características de un sistema o un componente" y "el resultado de este proceso." Visto como proceso, el diseño del software es la actividad del ciclo de vida de la cual los requisitos del software se analizan para producir una descripción de la estructura interna del software que servirá como la base para su construcción.

Más exacto, un diseño del software (el resultado) debe describir la arquitectura del software, en cómo la descomposición del software, la organización de los componentes, y los interfaces entre los mismos componentes. Debe también describir los componentes en un nivel de detalle que permita su construcción.

El diseño del software desempeña un papel importante en el desarrollo de software: permite que la Ingeniería del software produzca los diversos modelos para la solución que se pondrá en desarrollo. Podemos analizar y evaluar estos modelos para determinar si o no permitirán que se satisfaga los requisitos.

Podemos también examinar y evaluar varias soluciones alternativas y compensaciones. Finalmente, podemos utilizar los modelos que resultan para planear las actividades subsecuentes del desarrollo, además de usarlas como entrada o punto de partida de la construcción y prueba en un listado estándar de los procesos del ciclo de vida del software, tales como, procesos del ciclo de vida del software de IEEE/EIA 12207 [IEEE12207.0-96], diseño del software consiste en dos actividades que quepan entre el análisis de requisitos del software y la construcción del software:

- Diseño de la arquitectura del software (a veces llamado diseño de nivel superior): describiendo la estructura del software y organización e identificar a nivel superior los varios componentes
- Diseño detallado software del: describiendo cada componente suficientemente para tener en cuenta su construcción.

Referente al alcance del área del conocimiento del diseño del software (KA), la descripción actual de KA no discute todos los asuntos del nombre del cual contenga la palabra "diseño." En la terminología de Tom DeMarco (DeM99), el KA discutido en este capítulo trata principalmente del D-diseño (diseño de la descomposición, traza del software en partes de componentes). Sin embargo, debido a su importancia en el campo cada vez mayor de la arquitectura del software, también trataremos el diseño desde el punto de congelación (el diseño del patrón de familia, cual meta es establecer concordancias explotables en una familia del software). Por el contrario, el KA del diseño del software no trata el I-Diseño (el diseño de la Innovación, realizado generalmente durante el proceso de los requisitos del software con el objetivo de conceptuar y de especificar el software para satisfacer las necesidades y los requisitos), puesto que este asunto se debe considerar parte del análisis y la especificación de requisitos la descripción de KA del diseño del software se relaciona específicamente con los requisitos del software, la construcción del software, la gerencia de la ingeniería del software, la calidad del software, y las disciplinas relacionadas con la ingeniería del software

INTERRUPCIÓN DE LOS ASUNTOS PARA EL DISEÑO DEL SOFTWARE

1. Fundamentos del diseño del software

Los conceptos, las nociones, y la terminología introducida aquí forman una base subyacente para entender el papel y el alcance del diseño del software.

1.1. Conceptos generales de diseño

El software no es el único campo donde está implicado el diseño. En el sentido amplio, podemos ver diseño como forma de solucionar un problema. [Bud03: c1] Por ejemplo, el concepto de un problema travieso del problema-uno sin definitivo solución-es interesante en términos de entender los límites del diseño. [Bud04: c1] Un número de otras nociones y conceptos están también de interés en diseño el entender en su sentido general: metas, apremios, alternativas, representaciones, y soluciones. [Smi93]

1.2. Contexto del diseño del software

Para entender el papel del diseño del software, es importante entender el contexto, el ciclo de vida de la tecnología de dotación lógica. Así, es importante entender las características principales del análisis de requisitos del software contra diseño del software contra la construcción del software contra la prueba del software. [IEEE12207.0-96]; Lis01: c11; 2 Mar; Pfl01: c2; Pre04: c2]

1.3. *Proceso del diseño del software*

El diseño del software generalmente se considera un proceso de dos etapas: [Bas03; Dor02: v1c4s2; Fre83: I; IEEE12207.0-96]; Lis01: c13; 2 Mar: D]

1.3.1. Diseño arquitectónico

El diseño arquitectónico describe cómo el software se descompone y se organiza en los componentes (la arquitectura) del software [IEEEP1471-00]

1.3.2. Diseño detallado

El diseño detallado describe el comportamiento específico de estos componentes.

La salida de este proceso es un sistema de modelos y los artefactos que registran las decisiones principales que se han tomado. [Bud04: c2; IEE1016-98; Lis01: c13; Pre04: c9]

1.4. Permitir técnicas

Según el diccionario del inglés de Oxford, un principio es "una verdad básica o una ley general… que se utiliza como una base del razonamiento o guía de la acción." Los principios del diseño del software, también llamados técnicas permisibles [Bus96], son nociones dominantes que consideran fundamental a los diversos acercamientos y conceptos del diseño del software. Las técnicas que lo permiten son las siguientes: [Bas98: c6; Bus96: c6; IEEE1016-98; Jal97: c5, c6; Lis01: c1, c3; Pfl01: c5; Pre04: c9]

1.4.1. Abstracción

La abstracción es "el proceso de olvidarse de la información para poder tratar las cosas que son diferentes como si fueran iguales." *Lis01+ En el contexto del diseño del software, dos mecanismos dominantes de la abstracción son parametrización y especificación. La abstracción por la especificación conduce a tres clases importantes de abstracción: abstracción procesal, abstracción de los datos, y abstracción del control (iteración). [Bas98: c6; Jal97: c5, c6; Lis01: c1, c2, c5, c6; Pre04: c1]

1.4.2. Acoplador y cohesión

El acoplador se define como la fuerza de las relaciones entre los módulos, mientras que la cohesión es definida por cómo los elementos que componen un módulo son relacionados. [Bas98: c6; Jal97: c5; Pfl01: c5; Pre04: c9]

1.4.3. Descomposición y modularización

Descomposición y modularización del software en partes mas pequeñas e independientes, generalmente con la meta de poner diversas funcionalidades o responsabilidades en diversos componentes. [Bas98: c6; Bus96: c6; Jal97: c5; Pfl01: c5; Pre04: c9]

1.4.4. Encapsulación/el ocultar de la información

Medios que ocultan de la encapsulación/de la información que agrupan y que empaquetan los elementos y los detalles internos de una abstracción y que hacen esos detalles inaccesibles. [Bas98: c6; Bus96: c6; Jal97: c5; Pfl01: c5; Pre04: c9]

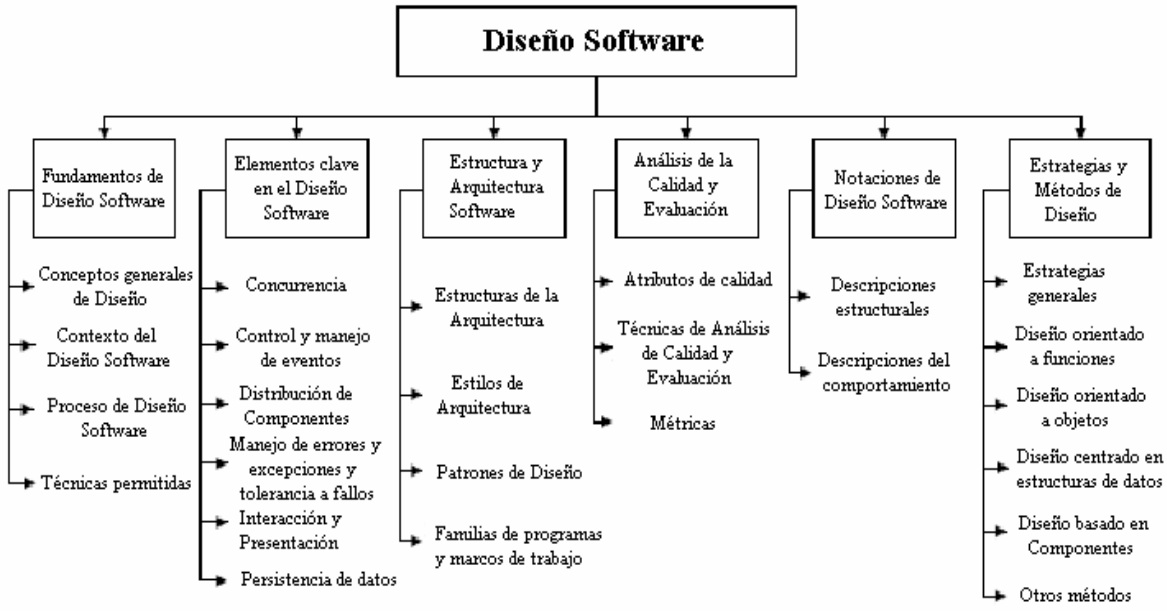

Figura1 Descomposición de los temas del KA de Diseño Software

1.4.5. Separación del interfaz y de la puesta en práctica

La separación del interfaz y de la puesta en práctica implica el definir de un componente especificando un interfaz público, a parte de los detalles de cómo se observa el componente. [Bas98: c6; Bos00: c10; Lis01: c1, c9]

1.4.6. Desahogo, lo completo y deshaciéndose de lo primitivo

Alcanzando desahogo, lo completo, y medios no primitivos se asegura que un componente de software captura todas las características importantes de una abstracción, y nada más. [Bus96: c6; Lis01: c5].

2. Cuestiones claves en diseño del software

Se tiene que tener en cuenta a la hora de diseñar software una serie de principios claves. Algunos son preocupaciones que todo el software debe tratar -por ejemplo, funcionamiento de la calidad. Otra edición importante es cómo se descomponer, se organiza, y constituyen los paquetes de software. Esto es tan fundamental que todos los acercamientos del diseño deben tratarlo de un modo u otro (véase las técnicas del asunto 1.4 y la subarea 6, los métodos que permiten el diseño del software). En cambio, otras ediciones "se ocupan de un cierto aspecto del comportamiento del software que no está en el dominio del uso, pero que trata algunos de los dominios de soporte." *Bos00+ Tales ediciones, que interseccionaron a menudo la funcionalidad del sistema, se han referido como aspectos: "*aspectos+ tender para no ser unidades de la descomposición funcional del software, sino algo para ser las características que afectan el funcionamiento o la semántica de los componentes de manera sistemáticas" (Kic97). Un número éstos llave, ediciones del cruz-corte son los siguientes (presentado en orden alfabético):

2.1 Concurrencia

Cómo descomponer el software en procesos, tareas, e hilos y reparto con eficacia relacionada, atomicidad, la sincronización, y ediciones programar. [Bos00: c5; 2 Mar: CSD; Mey97: c30; Pre04: c9]

2.2 Control y dirección de acontecimientos

Cómo organizar datos y controlar flujo, cómo manejar acontecimientos reactivos y temporales a través de varios mecanismos tales como invocación y servicios repetidos implícitos. [Bas98: c5; Mey97: c32; Pfl01: c5]

2.3 Distribución de componentes

Cómo distribuir el software a través del hardware, cómo los componentes se comunican, cómo el middleware se puede utilizar para ocuparse de software heterogéneo. [Bas03: c16; Bos00: c5; Bus96: c el 2 Mar de 94: DD; Mey97: c30; Pre04: c30]

2.1. Dirección del error y de excepción y tolerancia de fallos

Cómo prevenir y tolerar averías y ocuparse de condiciones excepcionales. [Lis01: c4; Mey97: c12; Pfl01: c5]

2.1. Interacción y presentación

Cómo estructurar y organizar las interacciones con los usuarios y la presentación de la información (por ejemplo, separación de la presentación y de la lógica del negocio usando el acercamiento del Modelo-Vista-Regulador). [Bas98: c6; Bos00: c5; Bus96: c2; Lis01: c13; Mey97: c32] Debe ser observado que este asunto no está sobre especificar los detalles del interfaz utilizador, que es la tarea del diseño del interfaz utilizador (una parte de ergonómica del software); ver las disciplinas relacionadas de la tecnología de dotación lógica.

2.1. Persistencia de los datos

Cómo los datos duraderos deben ser dirigidos. [Bos00: c5; Mey97: c31];

3. Estructura y arquitectura del software

En su sentido terminante, una arquitectura del software es "una descripción de los subsistemas y de los componentes de un sistema de software y de las relaciones entre ellas." (Bus96: c6) La arquitectura procura así definir la estructura interna - según el diccionario del inglés de Oxford, "la manera de la cual se construye o se organiza algo" - del software que resulta. Durante los mid-1990s, sin embargo, la arquitectura del software comenzó a emerger como disciplina más amplia que implicaba el estudio de las estructuras y de las arquitecturas del software en una manera más genérica [Sha96]. Esto dio lugar a un número de ideas interesantes sobre diseño del software en diversos niveles de la abstracción. Algunos de estos conceptos pueden ser útiles durante el diseño arquitectónico (por ejemplo, estilo arquitectónico) del software específico, así como durante su diseño detallado (por ejemplo, patrones de

nivel inferior del diseño). Pero pueden también ser útiles para diseñar sistemas genéricos, conduciendo al diseño de familias de los programas (también conocidos como líneas de productos). Interesante, la

3.1. Estructuras y puntos de vista arquitectónicos

Diversas facetas de alto nivel de una poder del diseño del software y deben ser descritas y ser documentadas. Estas facetas a menudo se llaman las opiniones: "Una visión representa un aspecto parcial de una arquitectura del software que demuestre características específicas de un sistema de software" *Bus96: c6]. Estas visiones distintas pertenecen a las ediciones distintas asociadas a diseño del software - por ejemplo, la visión lógica (que satisface los requisitos funcionales) contra la visión de proceso (ediciones de la concurrencia) contra la visión física (ediciones de la distribución) contra la opinión del desarrollo (cómo el diseño se analiza en unidades de la puesta en práctica). Otros autores utilizan diversas terminologías, como del comportamiento contra funcional contra estructural contra los datos que modelan opiniones. Resumiendo, un diseño del software es un artefacto múltiple producido por el proceso del diseño e integrado generalmente por visiones relativamente independientes y ortogonal. [Bas03: c2; Boo99: c31; Bud04: c5; Bus96: c6; IEEE1016-98; IEEE1471-00] Estilos arquitectónicos (patrones arquitectónicos macro)

Un estilo arquitectónico es "un sistema de apremios en una arquitectura *que+ define un sistema o una familia de arquitecturas que las satisfagan" *Bas03: c2+. Un estilo arquitectónico se puede considerar así mientras que un meta-modelo que pueda proporcionar la organización de alto nivel del software (su arquitectura macro). Los varios autores han identificado un número de estilos arquitectónicos importantes. [Bas03: c5; Boo99: c28; Bos00: c6; Bus96: c1, c6; Pfl01: c5]

- Estructura general (por ejemplo, capas, pipas, y filtros, pizarra)
- Sistemas distribuidos (por ejemplo, servidor de cliente, tres gradas, corredor)
- Sistemas interactivos (por ejemplo, regulador de la Modelo-Vista, Presentación-Abstracción-Control)
- Sistemas adaptables (por ejemplo, micro-núcleo, reflexión)
- Oros (por ejemplo, hornada, intérpretes, control, basados en las reglas de proceso).

3.2. Patrones del diseño (patrones arquitectónicos micro).

Resumido brevemente, un patrón es "una solución común a un problema común en un contexto dado." (Jac99) Mientras que los estilos arquitectónicos se pueden ver como patrones que describen la mayor parte de estos conceptos se pueden considerar como tentativas de describir, y de reutilizar así, conocimiento genérico del diseño.

organización de un nivel alto del software (su arquitectura macro); otros patrones del diseño se pueden utilizar para describir los detalles en un nivel más bajo, más local (su arquitectura micro). [Bas98: c13; Boo99: c28; Bus96: c1; 2 Mar: DP]

- Patrones de creación (por ejemplo, builder, factory, prototipo, y singleton)
- Patrones estructurales (por ejemplo, adapter, bridge, composite, decorator, façade, flyweight, and proxy)
- Patrones del comportamiento (por ejemplo, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor)

3.3 Familias de programas y de marcos.

Una posible opción para permitir la reutilización de los diseños y de los componentes del software es diseñar las familias del software, también conocidas como líneas del producto de software. Estas pueden ser hechas identificando las concordancias entre los miembros de tales familias y por los componentes reutilizables y customizables entre miembros de la familia. [Bos00: c7, c10; Bas98: c15; Pre04: c30] En programación orientada a objetos, una clave relacionada es la del marco: un subsistema parcialmente completo del software que puede ser ampliado apropiadamente instalando los plug-ins específicos (también conocidos como puntos calientes). [Bos00: c11; Boo99: c28; Bus96: c6]

4. Análisis y evaluación de la calidad del diseño del software

Esta sección incluye generalidades de la calidad y evaluación que se relacionen específicamente con el diseño del software. La mayoría se cubren de una manera general en Software Quality KA

4.1. Cualidades de los atributos

Varias atributos son generalmente importantes para obtener un diseño del software de buenos calidad - varios "ilities" (capacidad de mantenimiento, portabilidad, testeo, trazabilidad), los varios "nesses" (corrección, robustez), incluyendo la "aptitud del propósito." *Bos00: c5; Bud04: c4; Bus96: c6; ISO9126.1-01; ISO15026-98; Mar de 94: D; Mey97: c3; Pfl01: c5] Una distinción interesante es la que está entre las cualidades de la calidad discernible en el tiempo de ejecución (funcionamiento, seguridad, disponibilidad, funcionalidad, utilidad), ésas no discernibles en el tiempo de ejecución (modificabilidad, portabilidad, reutilidad, integridad, y testeabilidad), y ésas relacionadas con las calidades intrínsecas de la arquitectura (integridad, corrección,

y lo completo, capacidad conceptuales de la estructura). [Bas03: c4]

4.2 Técnicas de evaluación y calidad del análisis.

Varias técnicas pueden ayudar a asegurar la calidad de un diseño del software:

- Revisiones de diseño del software: informal o semiformal, a menudo basado en grupo, las técnicas para verificar y para asegurar la calidad de los artefactos del diseño (por ejemplo, revisiones de la arquitectura [Bas03: c11], revisiones de diseño, e inspecciones [Bud04: c4; Fre83: VIII; IEEE1028-97; Jal97: c5, c7; Lis01: c14; Pfl01: c5], técnicas basadas en panorama [Bas98: c9; Bos00: c5], y la toma de los requisitos [Dor02: v1c4s2; Pfl01:])
  - Análisis estático: análisis estático formal o semiformal (ningún ejecutable) que se puede utilizar para evaluar un diseño (por ejemplo, el análisis o el cross-checking automatizado) [Jal97 del fault-tree: c5; Pfl01:]
  - Simulación y prototipado: técnicas dinámicas para evaluar un diseño (por ejemplo, simulación o prototipo de la viabilidad [Bas98 del funcionamiento: c10; Bos00: c5; Bud04: c4; Pfl01: c5])

4.3 Medidas.

Las medidas se pueden utilizar para determinar o para estimar cuantitativamente varios aspectos del tamaño, de la estructura, o de la calidad de un diseño del software. La mayoría de las medidas se han propuesto que dependen generalmente del acercamiento usado para producir el diseño. Estas medidas se clasifican en dos amplias categorías:

- Diseño de medidas orientada a función (estructuradas): Estructura del diseño, obtenida sobre todo con la descomposición funcional; representado generalmente como una carta de estructura (a veces llamada un diagrama jerárquico) en la cual varias medidas pueden ser computadas [Jal97: c5, c7, Pre04:]
- Diseño de medidas orientada a objetos: La estructura total del diseño se representa a menudo como diagrama de la clase, en el cual varias medidas pueden ser computadas. Las medidas en las características del contenido interno de cada clase pueden también ser computadas [Jal97: c6, c7; Pre04: c15]

5 Notaciones del diseño del software:

Muchas notaciones e idiomas existen para representar los artefactos del diseño del software. Algunos se utilizan principalmente para describir la organización estructural de un diseño, otras para representar comportamiento del software. Ciertas notaciones se utilizan sobre todo durante el diseño arquitectónico y otros principalmente durante el diseño detallado, aunque algunas notaciones se pueden utilizar en ambos pasos. Además, algunas notaciones se utilizan sobre todo en el contexto de métodos específicos (véase el *Software Design Strategies and Methods* subarea). Aquí, se categorizan en las notaciones para describir la opinión (estática) estructural contra la visión (dinámica) del comportamiento.

5.1 Descripción estructural (vista estática):

Las siguientes notaciones, sobre todo (pero no siempre) gráficas, describen y representan los aspectos estructurales del diseño de software – las cuales, describen los componentes principales y cómo se interconectan (visión estática):

- Lenguajes descriptivos de la arquitectura: textuales, a menudo formal, los lenguajes describían una arquitectura del software en términos de componentes y conectadores [Bas03: c12]
- Diagramas de la clase y objeto: usados para representar un sistema de clases (y de objetos) y de sus correlaciones [Boo99: c8, c14; Jal97:]
- Diagramas de componentes: usados para representar un sistema de componentes (“parte física y reemplazable de un sistema al cual conforma y proporciona la realización de un sistema de interfaces” *Boo99+) y de sus correlaciones *Boo99: +
- Tarjetas del colaborador de la responsabilidad de la clase (CRCs): denotan los nombres de los componentes (clases), de sus responsabilidades, y nombres de sus componentes de colaboración’ [Boo99: c4;]
- Diagramas de despliegue: representar un sistema de nodos (físico) y de sus correlaciones, y, así, modelaban los aspectos físicos de un sistema [Boo99:]
- Diagramas de la Entidad-relación (ERDs): representan modelos conceptuales de los datos almacenados en los sistemas de información [Bud04: c6; Dor02: v1c5; 2]
- Lenguaje descriptivo de la interfaz (IDLS): programación como lenguajes usados para definir los interfaces (nombres y tipos de operaciones exportadas) de los componentes de software [Bas98: c8; Boo99:]
- Diagramas de la estructura de Jackson: Usados para describir las estructuras de datos en términos de secuencia, selección, e iteración [Bud04: c6; 2 Mar:
- Estructura de cartas: Usados para describir la estructura que llamaba de los programas (el módulo llama, y es llamado por otro

módulo) [Bud04: c6; Jal97: c5; 2 Mar: Dr; Pre04: c10]

5.2 Descripciones del comportamiento (visión dinámica):

Las siguientes notaciones y lenguajes, algunos gráficos y otros textuales, se utilizan para describir el comportamiento dinámico del software y de los componentes. Muchas de estas notaciones son útiles sobre todo, pero no exclusivamente, durante el diseño detallado.

- Diagramas de actividad: Muestran el flujo del control de la actividad ("ejecución no-atómica en curso dentro de una máquina del estado") a la actividad *Boo99: +
- Diagramas de colaboración: Muestran las interacciones que ocurren entre un grupo de objetos, donde está el énfasis en los objetos, sus acoplamientos, y los mensajes que intercambian en estos acoplamientos [Boo99:]
- Organigramas de datos: Muestran los flujos de datos entre un sistema y los procesos [Bud04: c6; 2 Mar: Dr; Pre04:]
- Tablas y diagramas de decisión: representan combinaciones complejas de las condiciones y de las acciones [Pre04:]
- Organigramas y organigramas estructurados: Representan el control de flujo y de las acciones asociadas que se realizarán [Fre83: VII; 2 Mar: Dr; Pre04:]
- Diagramas de secuencia: Muestran las interacciones entre un grupo de objetos, con énfasis sobre el tiempo de ordenación de mensajes [Boo99:]
- Transición de estado y diagramas de carta de estado: demostraban el control de flujo de estado a estado en una máquina de estados[Boo99: c24; Bud04: c6; 2 Mar: Dr; Jal97:]
- Lenguajes formales de especificación: Lenguajes textuales que utilizan nociones básicas de matemáticas (por ejemplo, lógica, sistema, secuencia), para obtener de forma rigurosa y abstracta, definir interfaces y comportamientos del componente de software, a menudo en términos de pre y post-condiciones [Bud04: c18; Dor02: v1c6s5; Mey97:]
- Lenguajes del diseño de pseudo código del programa (PDLs): Programa estructurado como los lenguajes usados para describir, generalmente en la etapa detallada del diseño, el comportamiento de un procedimiento o el método [Bud04: c6; Fre83: VII; Jal97: c7; Pre04: c8, c11]

6 Estrategias y métodos del diseño de software:

Existen varias estrategias generales para ayudar a dirigir el proceso de diseño. [Bud04: c9, 2 Mar: D] Al contrario que en las estrategias generales, los métodos son más específicos, sugieren y proporcionan generalmente un sistema de notaciones que se utilizarán con el método, una descripción del proceso que se utilizará después del método y un sistema de pautas al usar el método. [Bud04: c8] Tales métodos son útiles como medios de transferir conocimiento y como marco común para los equipos de los ingenieros de software. [Bud03: c8] Ver también Herramientas y metodos KA de Ingeniería de software.

6.1 Estrategias generales

Algunos de los ejemplos citados de las estrategias generales útiles en el proceso del diseño son dividir-y-conquistar y el refinamiento[Bud04: c12; Fre83: V], de arriba hacia abajo contra las estrategias bottom-up [Jal97: c5; Lis01: c13], abstracción de los datos y el ocultar de la información [Fre83: V], uso de la heurística [Bud04: c8], uso de patrones y lenguajes de patrónes [Bud04: c10; Bus96: c5], uso de un acercamiento iterativo e incremental. [Pfl01: c2]

6.2 Diseño (estructurado) orientado a función:

[Bud04: c14; Dor02: v1c6s4; Fre83: V; Jal97: c5; Pre04: está uno c9, c10]
Esto es uno de los métodos clásicos del diseño de software, donde los centros de descomposición identifican las funciones del software y después elaboran y refinan de una manera de arriba hacia abajo. El diseño estructurado se utiliza generalmente después de análisis estructurado, produciendo así, entre otras cosas, organigramas de datos y de descripciones de proceso asociados. Los investigadores han propuesto varias estrategias (por ejemplo, análisis de la transformación, análisis de la transacción) y la heurística (por ejemplo, fan-in/fan-out, alcance del efecto contra el alcance del control) para transformar un DFD en una arquitectura del software representada generalmente como carta de estructura.

6.3 Diseño orientado a objeto

[Bud0: c16; Dor02: v1: c6s2, s3; Fre83: VI; Jal97: c6; 2 Mar: D; Pre04: c9]
Numerosos métodos de diseño de software basados en objetos han sido propuestos. El campo se ha desarrollado basado en el diseño objeto de los mediados de los años ochenta (sustantivo = objeto; verbo = método; adjetivo = cualidad) con el diseño orientado a objetos, donde la herencia y el polimorfismo desempeñan un papel importante, el campo del diseño del componente-basado, donde la meta información puede ser definida y ser alcanzada (con la reflexión, por ejemplo). Aunque las raíces del

diseño Orientado a Objetos provienen del concepto de la abstracción de los datos, el diseño responsabilidad-conducido también se ha propuesto como alternativo al diseño Orientado a Objetos.

6.4 Diseño Dato-Estructura-Centrado

[Bud04: c15; Fre83: III, VII; 2 Mar02:D]
El diseño Dato-estructura-centrado (por ejemplo, Jackson, Warnier-Orr) comienza desde las estructuras de datos que un programa manipula más bien que desde las funciones que realiza. La Ingeniería de software primero describe las estructuras de datos de entrada y de salida (que usan los diagramas de la estructura de Jackson, por ejemplo) y en seguida desarrolla la estructura de control del programa basada en estos diagramas de estructura de datos. La varia heurística se ha propuesto para tratar como caso especial, cuando hay una unión mal hecha entre la entrada y las estructuras de la salida.

6.5 Diseño basado en componente (CBD):

Un componente de software es una unidad independiente, teniendo bien definidos los interfaces y dependencias que se pueden componer y desplegar independientemente. El diseño basado en componente trata las ediciones relacionadas con el abastecimiento, desarrollo, e integración de tales componentes para mejorar la reutilización. [Bud04: c11]

6.6 Otros métodos

Otros interesantes pero menos aprovechados también existen: métodos formales y rigurosos [Bud04: c18; Dor02: c5; Fre83; Mey97: c11; Pre04: c29] y métodos transformacionales. [Pfl98: c2]

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[iEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal 97]	[Li s01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
1.Fundamentos del Diseño de Software																c11s1				
1.1Conceptos Generales de Diseño				c1												c13s1, c13s2				
1.2El contexto de Diseño Software											*					c1s1, c13s2, c3s1-c3s3,c125-128, c9s1-c9s3	D		c2s2	c2
1.3 El Proceso de Diseño Software	c2s1			c2		v1c4s2	2-22	*		*	*						D			c9
1.4 Técnicas Permitidas	{c6s1}		c10s3		c6s3			*						c5s1, c5s2, c6s2					c5s2, c5s5	c9
2.Elementos clave en el Diseño Software																				
2.1 Concurrencia			c5s4.1														CSD	c30		c9
2.2 Control y manejo de eventos	{c5s2}																{DD}	c32s4, c32s5	c5s3	
2.3 Distribución de componenetes	c16s3, c16s4		c5s4.1		c2s3													c30		c30
2.4 Manejo de errores y excepciones y tolerancia a fallos																c4s3-c4s5		c12	c5s5	
2.5 Interacción y presentación	{c6s2}		c5s4.1		c2s4											c13s3		c32s2		
2.6 Persistencia de Datos			c5s4.1															c31		

* ver siguiente sección

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[iEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal 97]	[Li s01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
3. Estructura y Arquitectura Software		c31																		
3.1 Estructuras de la Arquitectura	c2s5	c28		c5	c6s1			*		*										
3.2 Estilos de Arquitectura	c5s9	c28	c6s3.1		c1s1-c1s3, c6s2													c2s3		
3.3 Patrones de Diseño	{c13s3}	c28			c1s1-c1s3											DP				
3.4 Familias de programas y Marcos de Trabajo	{c15s1, c15s3}		c7s1, c7s2, c10s2-c10s4, c11s2 –c11s4		c6s2														c30	
4. Análisis de la Calidad y Evaluación del Diseño de Software																				
4.1 Atributos de Calidad	c4s2		c5s2.3	c4	c6s4							*	*			{D}	c3	c5s5		
4.2 Técnicas de Análisis de Calidad y Evaluación	c11s3, [c9s1, c9s2, c10s2, c10s3}		c5s2.1, c5s2.2, c5s3, c5s4	c4		v1c4s2	542 - 576		*					c5s5, c7s3	c14s1			c5s6, c5s7, c11s5		
4.3 Métricas														c5s6, c6s5, c7s4					c15	

	[Bas03] {Bas98}	[Boo99]	[Bos00]	[Bud03]	[Bus96]	[Dor02]	[Fre83]	[IEEE1016-98]	[iEEE1028-97]	[IEEE1471-00]	[IEEE12207.0-96]	[ISO9126-01]	[ISO15026-98]	[Jal 97]	[Li s01]	[Mar02]* {Mar94}	[Mey97]	[Pfl01]	[Pre04]	[Smi93]
5. Notaciones de Diseño de Software																				
5.1 Descripciones estructurales (Vista Estática	{c8s4} c12s1, c12s2	c4, c8 c11, c12, c14, c30, c31		c6	429									c5s3, c6s3		DR				
5.2 Descripciones del Comportamiento (Vista Dinámica)				c6, c18		v1c5	485-490, 506-513							c7s2		DR			c10	
6. Métodos y Estrategias en Diseño de Software																	c11		c8,c11	
6.1 Estrategias generales				c8, c10, c12	c5s1-c5s4		304-320, 533-539							c5s1.4	c13s13					
6.2 Diseño Orientado a Funciones (Estructurado)							328-352							c5s4				c2s2		
6.3 Diseño Orientado a Objetos							420-436							c6s4		D			c9,c10	
6.4 Diseño centrado en Estructuras de Datos							201-120, 514-532									D			c9	
6.5 Diseño Basado en Componentes (CBD)																				
6.6 Otros						181-192											c11	c2s2	c29	

Referencias Recomendadas Para el Diseño de Software

[Bas98] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.[Bas03] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, second ed., Addison-Wesley, 2003.

[Boo99] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[Bos00] J. Bosch, Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach, first ed., ACM Press, 2000.

[Bud04] D. Budgen, *Software Design*, second ed., Addison-Wesley, 2004.

[Bus96] F. Buschmann et al., Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.

[Dor02] M. Dorfman and R.H. Thayer, eds., *Software Engineering* (Vol. 1 & Vol. 2), IEEE Computer Society Press, 2002.

[Fre83] P. Freeman and A.I. Wasserman, *Tutorial on Software Design Techniques*, fourth ed., IEEE Computer Society Press, 1983.

[IEEE610.12-90] IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

[IEEE1016-98] IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions, IEEE, 1998.

[IEEE1028-97] IEEE Std 1028-1997 (R2002), *IEEE Standard for Software Reviews*, IEEE, 1997.

[IEEE1471-00] IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Description of Software Intensive Systems*, Architecture Working Group of the Software Engineering Standards Committee, 2000.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, IEEE, 1996.

[ISO9126-01] ISO/IEC 9126-1:2001, Software Engineering Product Quality—Part 1: Quality Model, ISO and IEC, 2001.

[ISO15026-98] ISO/IEC 15026-1998, Information Technology — System and Software Integrity Levels, ISO and IEC, 1998.

[Jal97] P. Jalote, An Integrated Approach to Software Engineering, second ed., Springer-Verlag, 1997.

[Lis01] B. Liskov and J. Guttag, Program Development in Java: Abstraction, Specification, and Object-Oriented Design, Addison-Wesley, 2001.

[Mar94] J.J. Marciniak, Encyclopedia of Software Engineering, J. Wiley & Sons, 1994. The references to the Encyclopedia are as follows:

CBD = Component-Based Design
D = Design
DD = Design of the Distributed System
DR = Design Representation

[Mar02] J.J. Marciniak, Encyclopedia of Software Engineering, second ed., J. Wiley & Sons, 2002.

[Mey97] B. Meyer, Object-Oriented Software Construction, second ed., Prentice-Hall, 1997.

[Pfl01] S.L. Pfleeger, Software Engineering: Theory and Practice, second ed., Prentice-Hall, 2001.

[Pre04] R.S. Pressman, Software Engineering: A Practitioner's Approach, sixth ed., McGraw-Hill, 2004.

[Smi93] G. Smith and G. Browne, "Conceptual Foundations of Design Problem-Solving," IEEE Transactions on Systems, Man and Cybernetics, vol. 23, iss. 5, 1209-1219, Sep.-Oct. 1993.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES

(Boo94a) G. Booch, Object Oriented Analysis and Design with Applications, second ed., The Benjamin/Cummings Publishing Company, 1994.

(Coa91) P. Coad and E. Yourdon, Object-Oriented Design, Yourdon Press, 1991.

(Cro84) N. Cross, Developments in Design Methodology, John Wiley & Sons, 1984.

(DSo99) D.F. D'Souza and A.C. Wills, Objects, Components, and Frameworks with UML — The Catálisis Approach, Addison-Wesley, 1999.

(Dem99) T. DeMarco, "The Paradox of Software Architecture and Design," Stevens Prize Lecture, Aug. 1999.

(Fen98) N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, second ed., Internacional Thomson Computer Press, 1998.

(Fow99) M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

(Fow03) M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003.

(Gam95) E. Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

(Hut94) A.T.F. Hutt, Object Analysis and Design — Comparison of Methods. Object Analysis and Design — Description of Methods, John Wiley & Sons, 1994.

(Jac99) I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

(Kic97) G. Kiczales et al., "Aspect-Oriented Programming," presented at ECOOP '97 — Object-Oriented Programming, 1997.

(Kru95) P. B. Kruchten, "The 4+1 View Model of Architecture," IEEE Software, vol. 12, iss. 6, 42-50, 1995

(Lar98) C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice-Hall, 1998.

(McC93) S. McConnell, Code Complete: A Practical Handbook of Software Construction, Microsoft Press, 1993.

(Pag00) M. Page-Jones, Fundamentals of Object-Oriented Design in UML, Addison-Wesley, 2000.

(Pet92) H. Petroski, To Engineer Is Human: The Role of Failure in Successful Design, Vintage Books, 1992.

(Pre95) W. Pree, Design Patterns for Object-Oriented Software Development, Addison-Wesley and ACM Press, 1995.

(Rie96) A.J. Riel, Object-Oriented Design Heuristics, Addison-Wesley, 1996.

(Rum91) J. Rumbaugh et al., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

(Sha96) M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996.

(Som05) I. Sommerville, Software Engineering, seventh ed., Addison-Wesley, 2005.

(Wie98) R. Wieringa, "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," ACM Computing Surveys, vol. 30, iss. 4, 1998, pp. 459-527.

(Wir90) R. Wirfs-Brock, B. Wilkerson, and L. Wiener, Designing Object-Oriented Software, Prentice-Hall, 1990.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

(IEEE1016-98) IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions, IEEE, 1998.

(IEEE1028-97) IEEE Std 1028-1997 (R2002), IEEE Standard for Software Reviews, IEEE, 1997.

(IEEE1471-00) IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Descriptions of Software-Intensive Systems, Architecture Working Group of the Software Engineering Standards Committee, 2000.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, vol. IEEE, 1996.

(ISO9126-01) ISO/IEC 9126-1:2001, Software Engineering-Product Quality-Part 1: Quality Model, ISO and IEC, 2001.

(ISO15026-98) ISO/IEC 15026-1998 Information Technology — System and Software Integrity Levels, ISO and IEC, 1998.