DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Diseño de sistemas de información

© Fernando Berzal, berzal@acm.org

Diseño

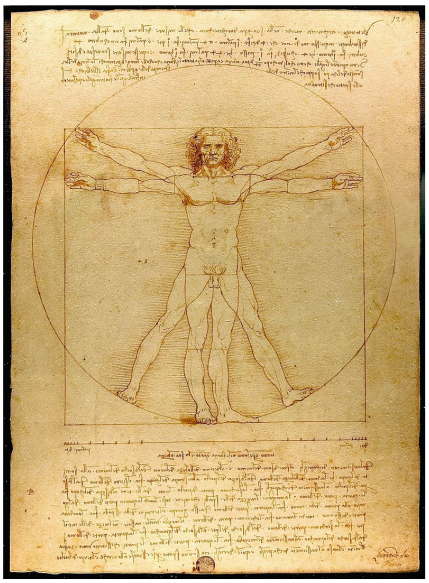
- Diseño de software.
- El proceso de diseño:
  - Revisiones técnicas.
  - Calidad del diseño: FURPS.
- Técnicas de diseño:
  - Evolución.
- Caso práctico: Aplicaciones de gestión.

Diseño

"Software design manifesto"

Mitch Kapor (creador del Lotus 1-2-3)

Dr. Dobbs Journal 16(1):62-67, January 1991



"What is design? It's where you stand with a foot in two worlds—the world of technology and the world of people and human purposes—and you try to bring the two together…

The Roman architecture critic Vitruvius advanced the notion that well-designed buildings were those which exhibited firmness, commodity, and delight."

Diseño

Puerta de Europa
Torres KIO, 114m, 1996
(Phillip Johnson & John Burgee)

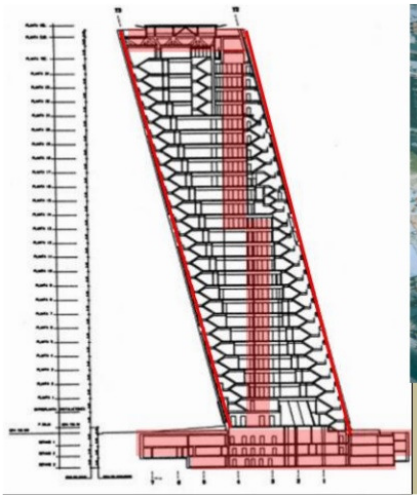
Castellana 81
Torre BBVA, 107m, 1981
(Francisco Javier Sáenz de Oiza)

Diferencia entre un buen y un mal edificio…

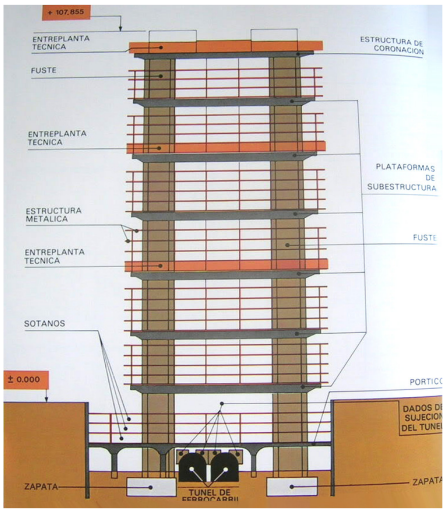
Pedro Torrijos, Twitter, junio de 2021

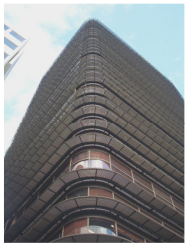
https://twitter.com/Pedro_Torrijos/status/1401463897429876737

Diseño



Debido a la inclinación de las torres, la **cimentación** se diseñó con un **contrapeso** de 15.000 toneladas para contrarrestar el vuelco. A dicho contrapeso van anclados los cables de postensado que lo solidarizan a la estructura.





Diferencia entre un buen y un mal edificio…
Pedro Torrijos, Twitter, junio de 2021
https://twitter.com/Pedro_Torrijos/status/1401463897429876737

Diseño

"Software design manifesto"

Mitch Kapor (creador del Lotus 1-2-3)

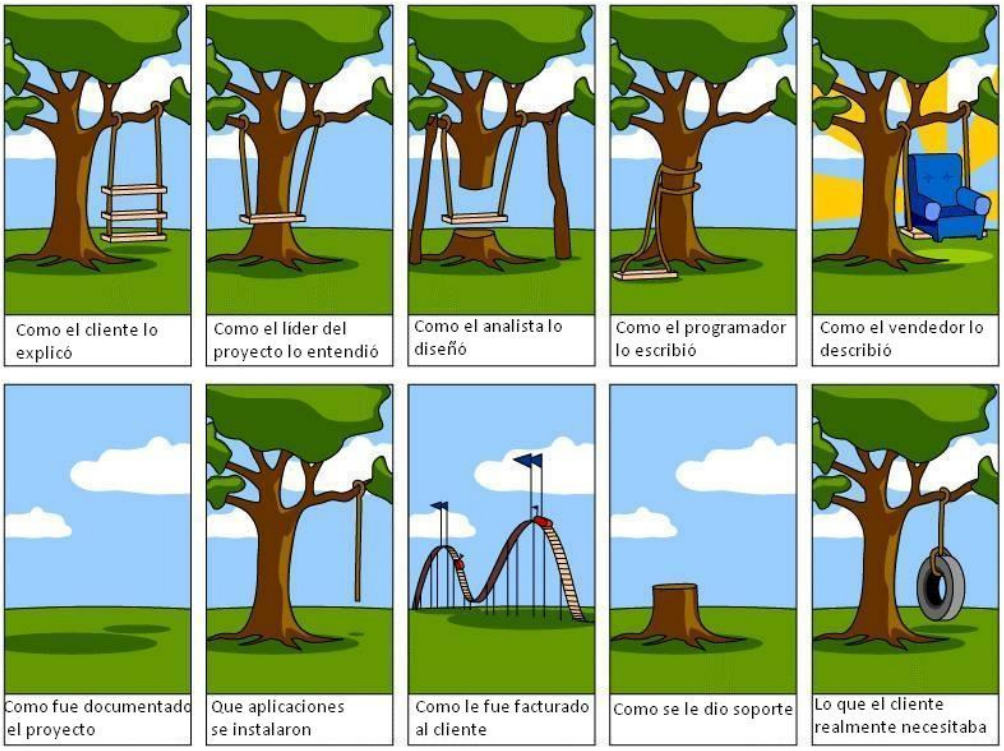
Dr. Dobbs Journal 16(1):62-67, January 1991

"… The same might be said of good software.

- **Firmness**: A program should not have any bugs that inhibit its function.
- **Commodity**: A program should be suitable for the purposes for which it was intended.
- **Delight**: The experience of using the program should be a pleasurable one.

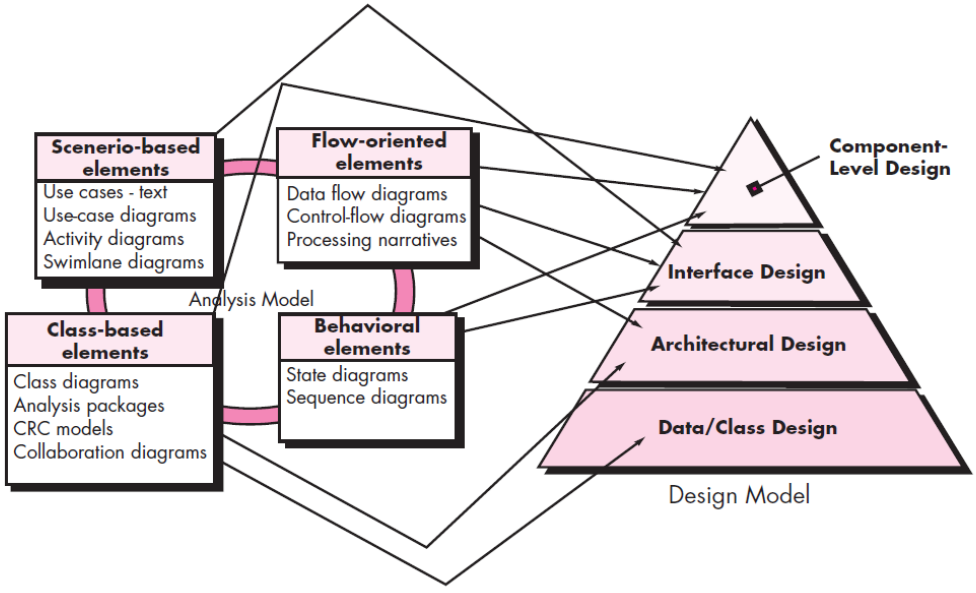
Here we have the beginnings of a theory of design for software."

Diseño



Diseño

Del análisis de requisitos al diseño de un sistema:



Roger S. Pressman:
Software Engineering: A Practitioner's Approach, 7/e, 2009

Diseño

Consejos de Pressman

- El diseño de software siempre debería empezar considerando los datos – los cimientos para todos los demás elementos del diseño.
- Una vez que los cimientos están en su sitio, se ha de derivar una arquitectura adecuada.
- Sólo después de establecer la arquitectura del sistema se deben realizar las demás tareas de diseño (interfaces y componentes)

Diseño

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

-- C. A. R. Hoare

El proceso de diseño

Proceso iterativo mediante el cual se trasladan los requisitos a un "plano" [blueprint] para la construcción del software:

- El diseño se representa a un nivel de abstracción alto (trazabilidad con los objetivos del sistema y sus requisitos más detallados).
- Conforme avanzan las iteraciones de diseño, los refinamientos del diseño conducen a representaciones de menor nivel de abstracción (en las que la conexión con los requisitos del sistema es más sutil).

El proceso de diseño

Revisiones técnicas

Evaluación de la calidad del diseño

R. McGlaughlin: "Some Notes on Program Design," Software Engineering Notes, 16(4):53-54, October 1991

- El diseño debe soportar todos los requisitos explícitos del sistema (los que aparecen en el documento de especificación de requisitos) y acomodar los requisitos implícitos deseados por los diferentes "stakeholders".
- El diseño debe servir de guía comprensible para los encargados de su desarrollo, QA y soporte.
- El diseño debe proporcionar una vista completa del sistema desde el punto de vista de su implementación.

El proceso de diseño

La calidad del diseño

Criterios técnicos para evaluar un diseño:

- Arquitectura creada utilizando patrones y estilos arquitectónicos reconocibles.
- Diseño modular (sistema dividido en elementos o subsistemas de forma lógica): componentes.
- Arquitectura implementable iterativamente (para facilitar implementación y pruebas).
- Representación de los datos, de la arquitectura, de las interfaces del sistema y de sus componentes (utilizando una notación adecuada).

El proceso de diseño

La calidad del diseño

Criterios técnicos para evaluar un diseño:

- Datos:
  Estructuras de datos apropiadas.
- Componentes:
  Cohesión & acoplamiento.
- Interfaces:
  Reducción de la complejidad de las conexiones entre componentes (y con sistemas externos).

El proceso de diseño

La calidad del diseño: FURPS

Atributos de calidad (Hewlett-Packard)

- **F**unctionality [funcionalidad]
- **U**sability [usabilidad]
- **R**eliability [fiabilidad]
- **P**erformance [rendimiento]
- **S**upportability [soporte]: maintainability (extensibility, adaptability, serviceability), testability, compatibility, configurability…

R.B. Grady & D.L. Caswell:
Software Metrics: Establishing a Company-Wide Program, 1987.

El proceso de diseño

"A designer knows that he has achieved perfection
not when there is nothing left to add,
but when there is nothing left to take away."

-- Antoine de Saint-Exupéry

Técnicas de diseño

Evolución de las técnicas de diseño de software

Orígenes

- Criterios para el desarrollo de programas modulares.
  David L. Parnas:
  "On the Criteria to Be Used in Decomposing Systems into Modules,"
  CACM 15(12):1053-1058, December 1972

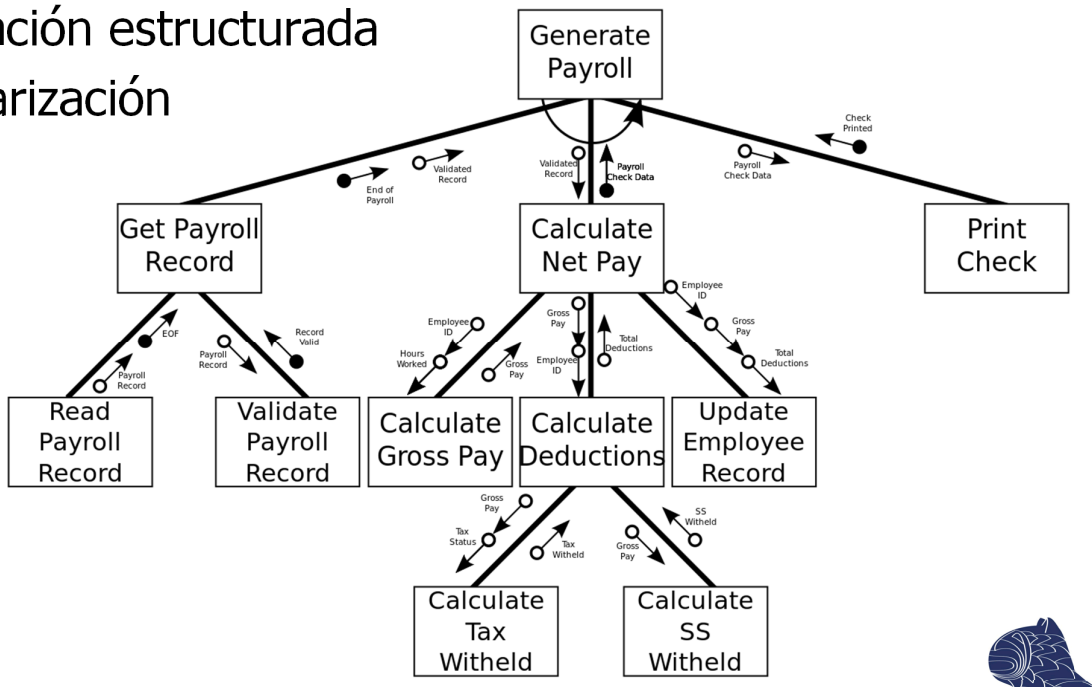
- Métodos de refinamiento: Diseño descendente.
  Niklaus Wirth:
  "Program Development by Stepwise Refinement," CACM 14(4):221-227, April 1971

Técnicas de diseño

Evolución de las técnicas de diseño de software

Programación estructurada

- Modularización



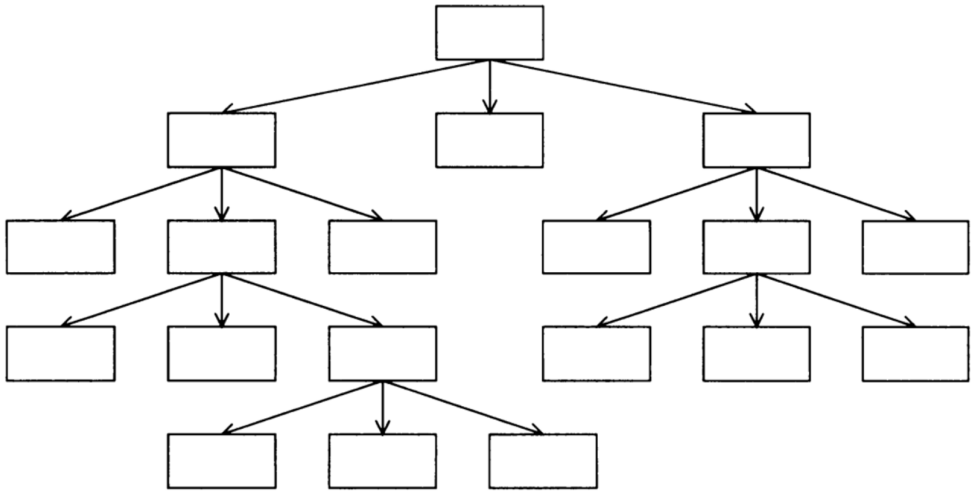
https://en.wikipedia.org/wiki/Structure_chart

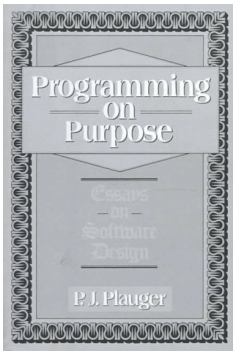
Técnicas de diseño

Evolución de las técnicas de diseño de software

Programación estructurada

Carta de estructura, a.k.a. HIPO chart (IBM)





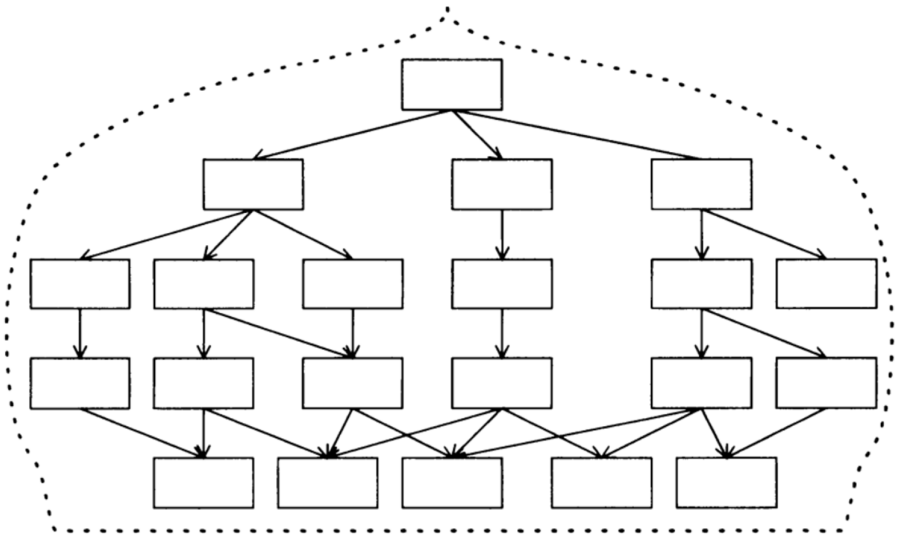
HIPO = Hierarchy with Input, Process, and Output

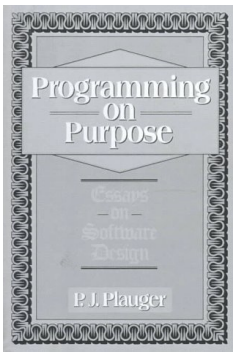
Técnicas de diseño

Evolución de las técnicas de diseño de software

Programación estructurada

Carta de estructura, a.k.a. HIPO chart (IBM)



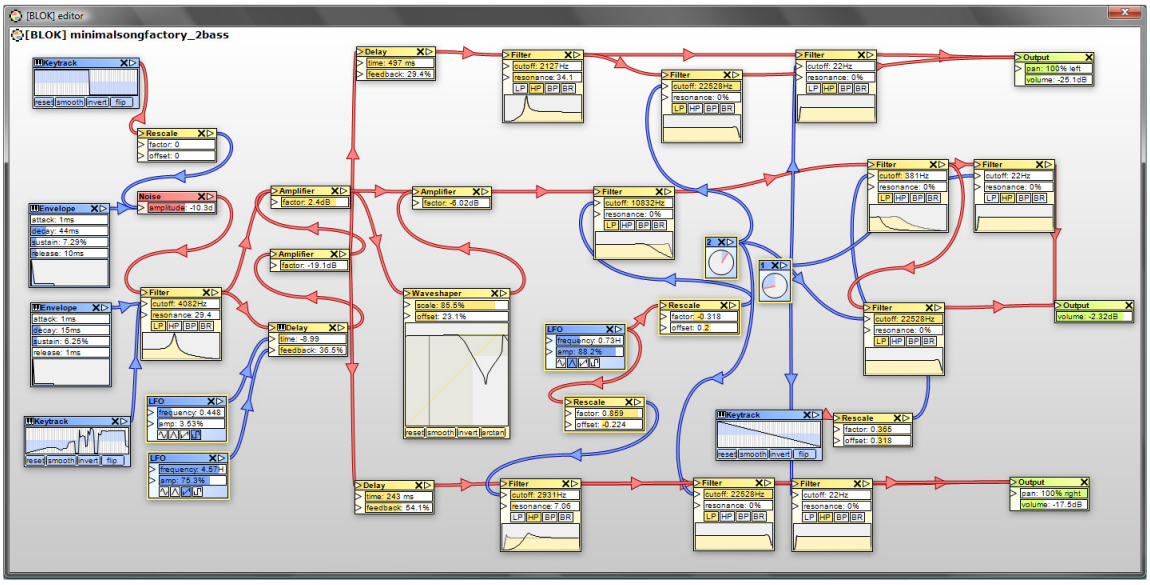


Una estructura más realista

Técnicas de diseño

Evolución de las técnicas de diseño de software

Dataflow programming [flujos de datos]



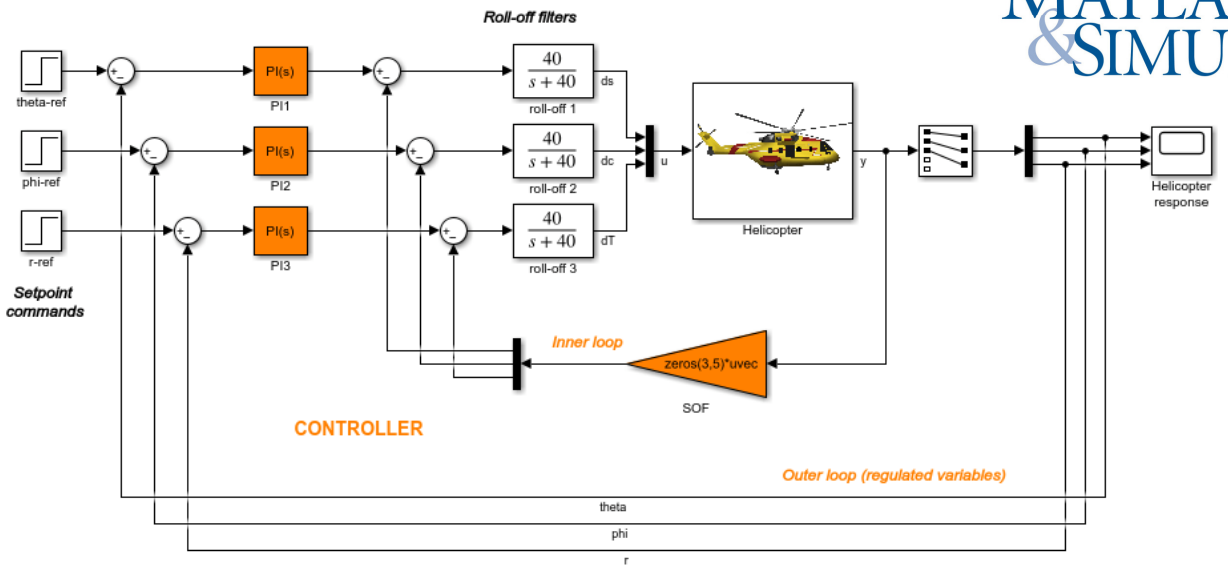
https://en.wikipedia.org/wiki/Dataflow_programming

Técnicas de diseño

Evolución de las técnicas de diseño de software

Dataflow programming [flujos de datos]

MATLAB® & SIMULINK®

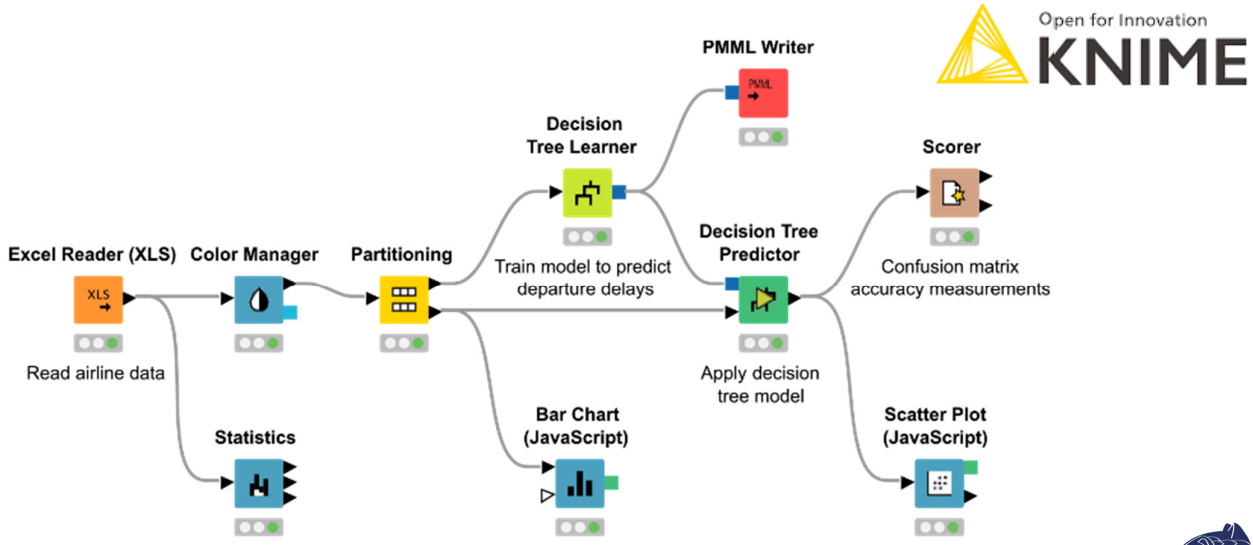


Copyright 2015 The MathWorks, Inc.

Técnicas de diseño

Evolución de las técnicas de diseño de software

Dataflow programming [flujos de datos]



Técnicas de diseño

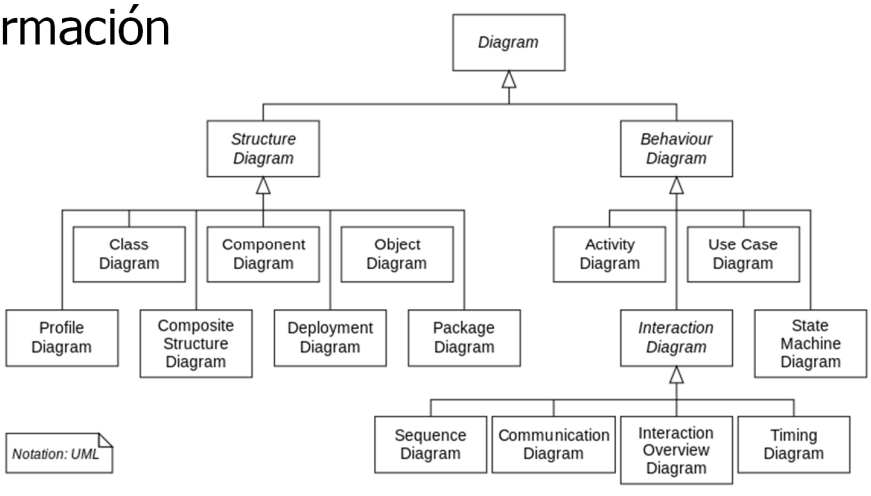
Evolución de las técnicas de diseño de software

Orientación a objetos

- Ocultación de información

Class Name	
Responsibilities	Collaborators

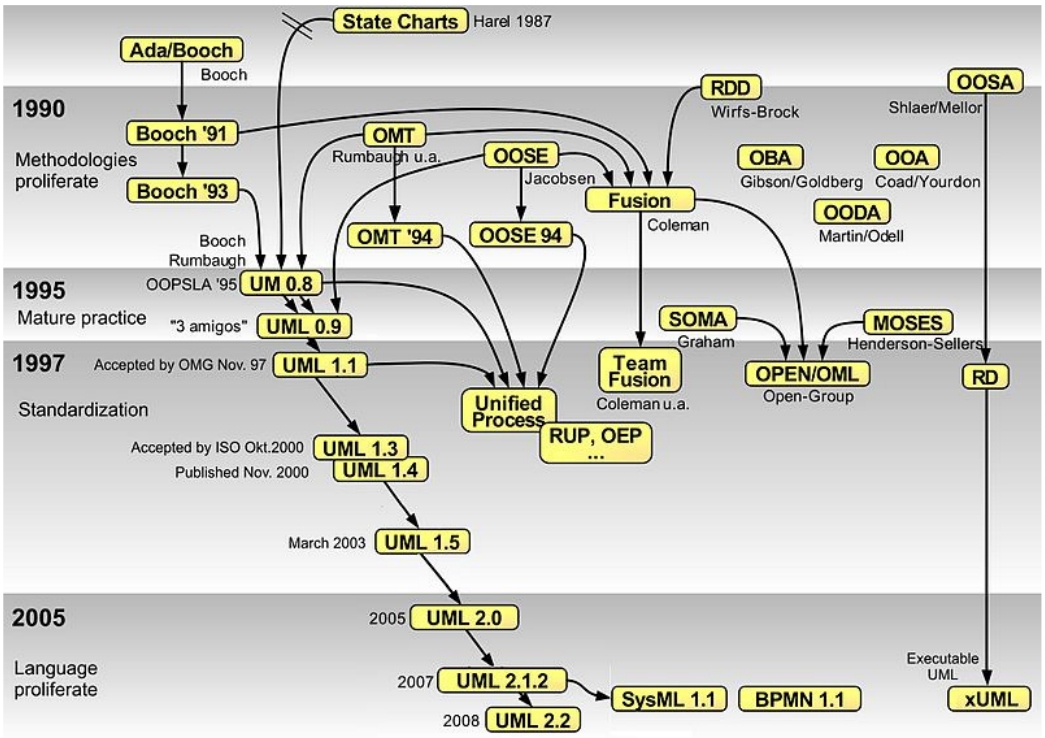
Tarjeta CRC



UML: Unified Modeling Language

Técnicas de diseño

Modelado OO

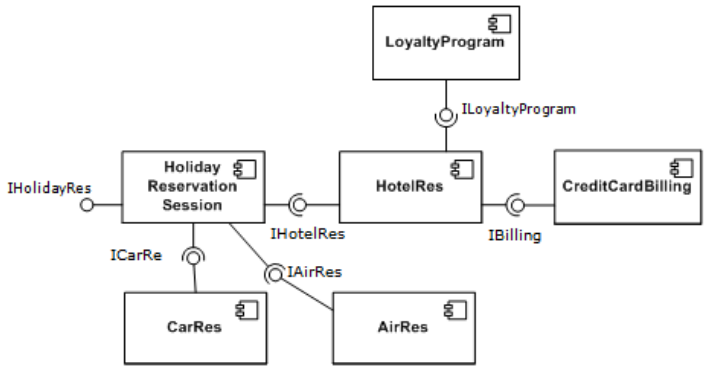


Técnicas de diseño

Evolución de las técnicas de diseño de software

Diseño basado en componentes [CBD o CBSE]

- Separation of Concerns [SoC]
- Middleware: EJB, COM, CORBA, SOA…



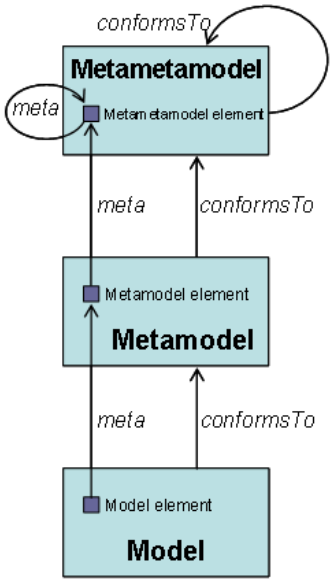
Componentes en UML 2.0

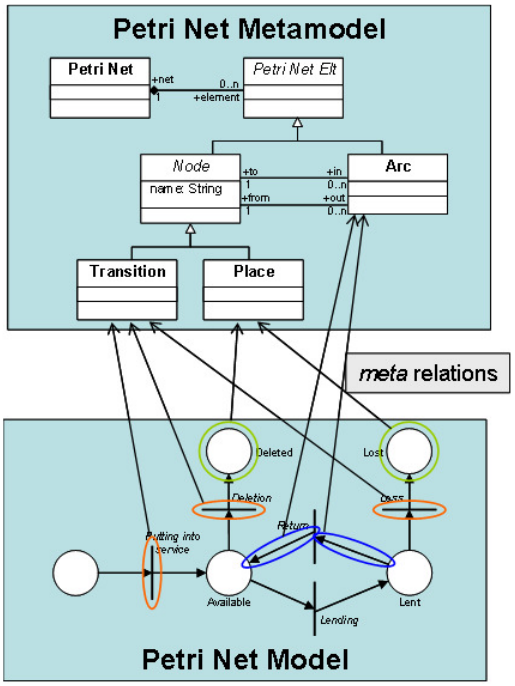
https://en.wikipedia.org/wiki/Component-based_software_engineering

Técnicas de diseño

Evolución de las técnicas de diseño de software

Model-driven software development [MDSD]



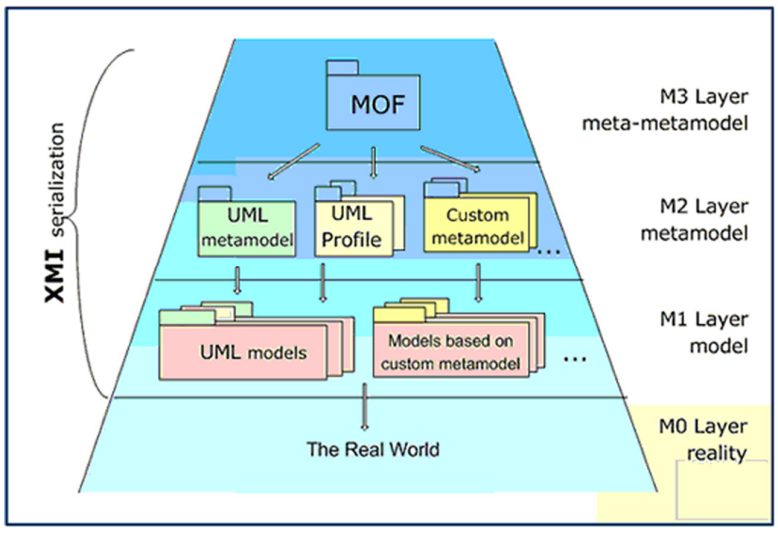


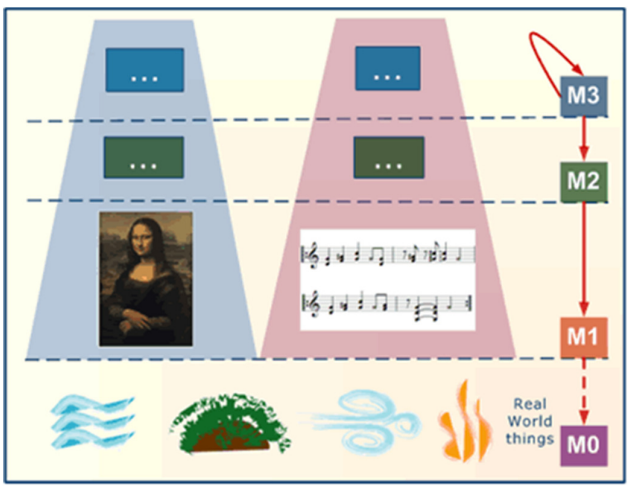
Metamodelos

Técnicas de diseño

Evolución de las técnicas de diseño de software

Model-driven software development [MDSD]



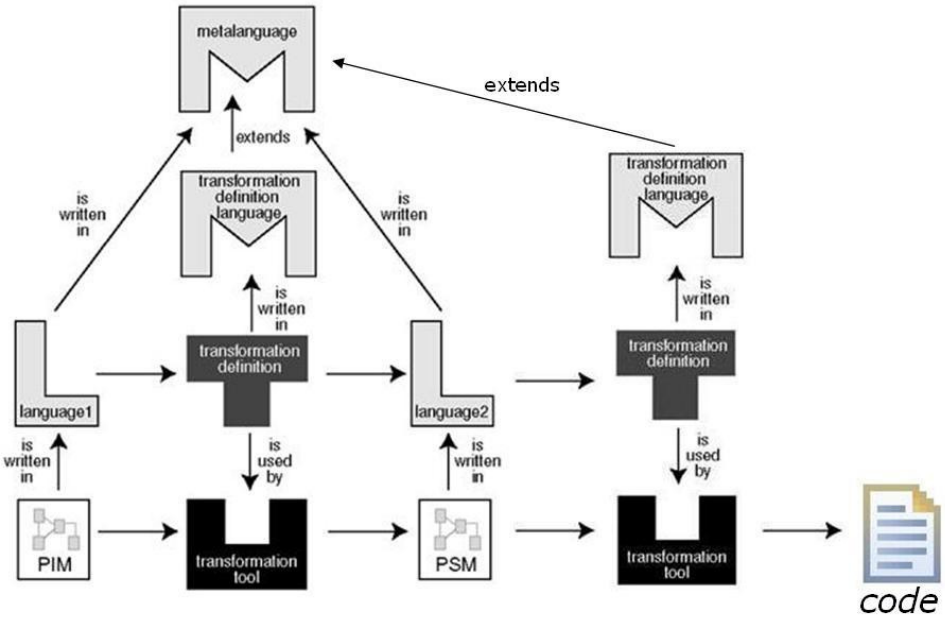


Metamodelos: OMG Meta-Object Facility [MOF]

Técnicas de diseño

Evolución de las técnicas de diseño de software

Model-driven software development [MDSD]

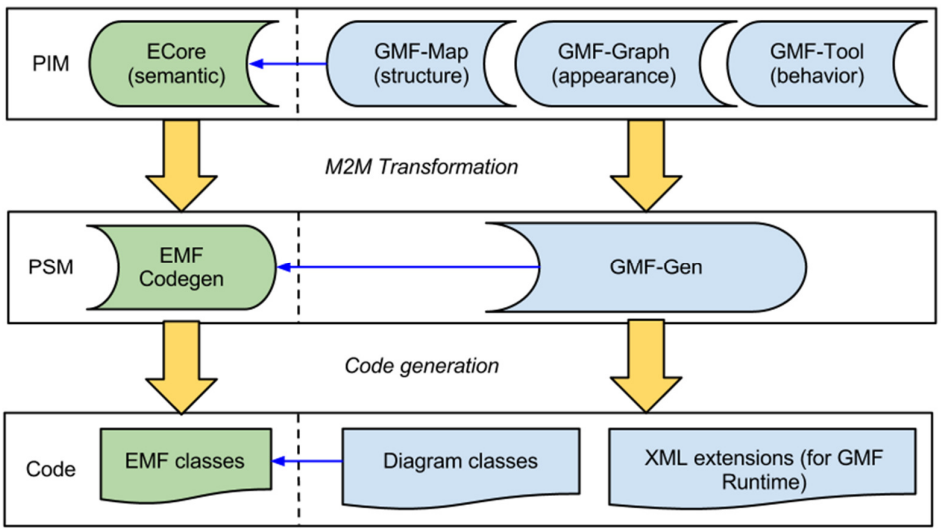


Ejemplo: OMG Model-Driven Architecture

Técnicas de diseño

Evolución de las técnicas de diseño de software

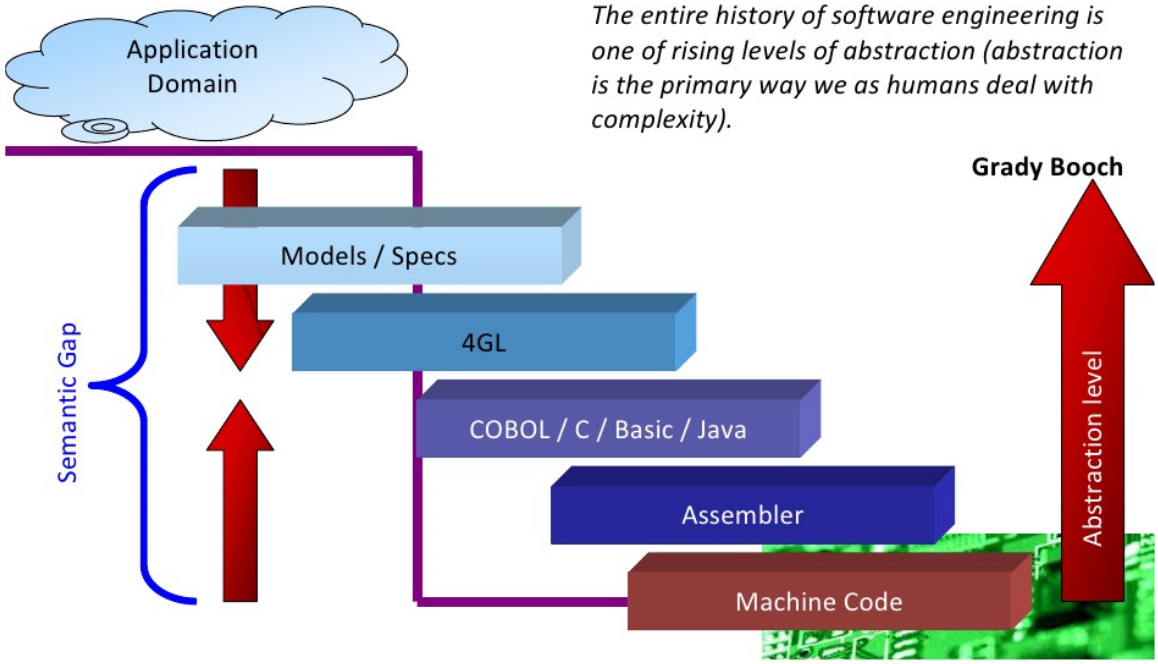
Model-driven software development [MDSD]



Ejemplo: Eclipse Modeling Framework (EMF)

Técnicas de diseño

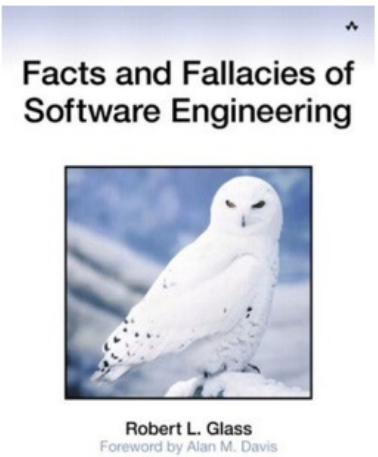
Evolución de las técnicas de diseño de software



The entire history of software engineering is one of rising levels of abstraction (abstraction is the primary way we as humans deal with complexity).

Grady Booch

Técnicas de diseño



"Most software tool and technique improvements account for about a 5- to 30-percent increase in productivity and quality. But at one time or another, most of these improvements have been claimed by someone to have "order of magnitude" (factor of 10) benefits. Hype is the plague on the house of software."

— **Robert L. Glass**:
"Facts and Fallacies of Software Engineering," 2003.

Técnicas de diseño

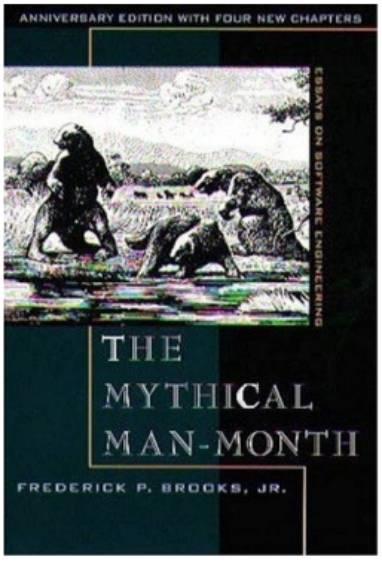
"The most important factor in attacking complexity is not the tools and techniques that programmers use but rather the quality of the programmers themselves."

— **Robert L. Glass**:
"Facts and Fallacies of Software Engineering", 2003.

"Flon's axiom: There does not now, nor will there ever, exist a programming language in which it is the least bit hard to write bad programs."

— **Lawrence Flon**: "On Research in Structured Programming", ACM SIGPLAN Notices, October 1975

Técnicas de diseño



Frederick P. Brooks Jr.:
"No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer, April 1987

Caso práctico

Aplicaciones de gestión [enterprise applications]

Requisitos funcionales

- Datos persistentes:
  Grandes volúmenes de datos organizados en bases de datos (relacionales, multidimensionales, NoSQL…).
- Acceso concurrente:
  Múltiples usuarios accediendo al sistema.
- Interfaz de usuario:
  GUI basada en ventanas, web y dispositivos móviles.
- Integración con otras aplicaciones y sistemas:
  Uso de middleware e intercambio de datos.

Caso práctico

Aplicaciones de gestión [enterprise applications]

Requisitos no funcionales

- Tiempo de respuesta.
- "Responsividad" [responsiveness].
- Latencia.
- Throughput (trabajo/tiempo).
- Capacidad (carga de trabajo máxima).
- Escalabilidad:
  - Vertical (servidor más potente), a.k.a. scaling up.
  - Horizontal (más servidores), a.k.a. scaling out.

Caso práctico

Aplicaciones de gestión [enterprise applications]
Evolución

IBM System z9 (2005)

Mainframes (1960s-)
a.k.a. "big iron" computers

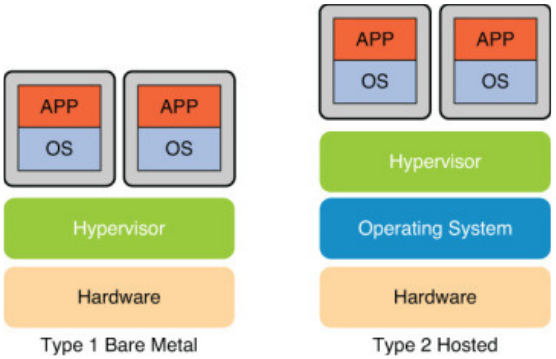
- Procesamiento por lotes y aplicaciones críticas.
- Compatibilidad hacia atrás (software antiguo)
- Redundancia ➔ ↑ fiabilidad
- Sistema de E/S ➔ ↑ throughput
- Virtualización ➔ ↑ utilización del hardware

Caso práctico

Aplicaciones de gestión [enterprise applications]
Evolución

Virtualización

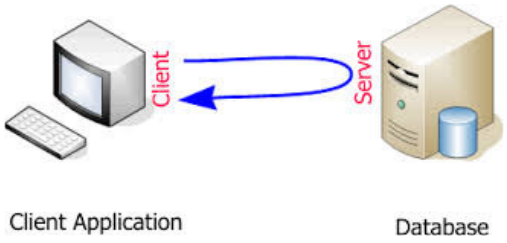
- Máquinas virtuales.
- Instantáneas.
- Migración & failover.



Caso práctico

Aplicaciones de gestión [enterprise applications]
Evolución

Arquitecturas cliente/servidor [C/S]



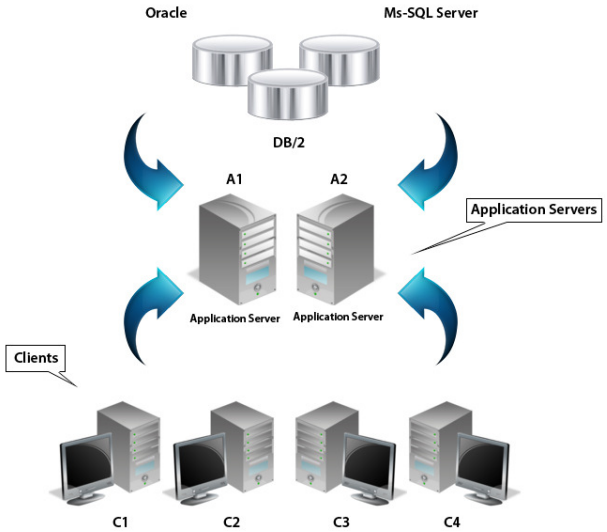
Caso práctico

Aplicaciones de gestión [enterprise applications]
Evolución

Arquitecturas multicapa

Distribución física:

3-tier architecture



- Clientes (GUI)
- Servidores de aplicaciones
- Almacenamiento de datos (p.ej. RDBMS)

Caso práctico

Aplicaciones de gestión [enterprise applications]

Evolución

Arquitecturas multicapa

Distribución lógica

3-layer architecture

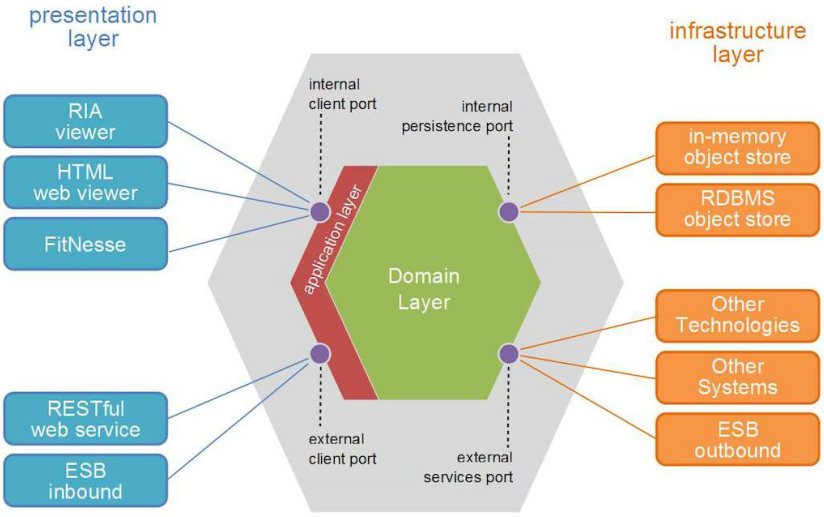
- Capa de presentación (GUI & servicios)
- Capa de aplicación (a.k.a. dominio)
- Capa de acceso a los datos (DB & middleware)

Caso práctico

Otra forma de verlo:

Arquitectura hexagonal

a.k.a. Ports & Adapters architecture



http://alistair.cockburn.us/Hexagonal+architecture

Caso práctico

¿Dónde se implementa la lógica de la aplicación?

No es una buena idea implementarla en la capa de presentación (incrementa el coste de mantenimiento).

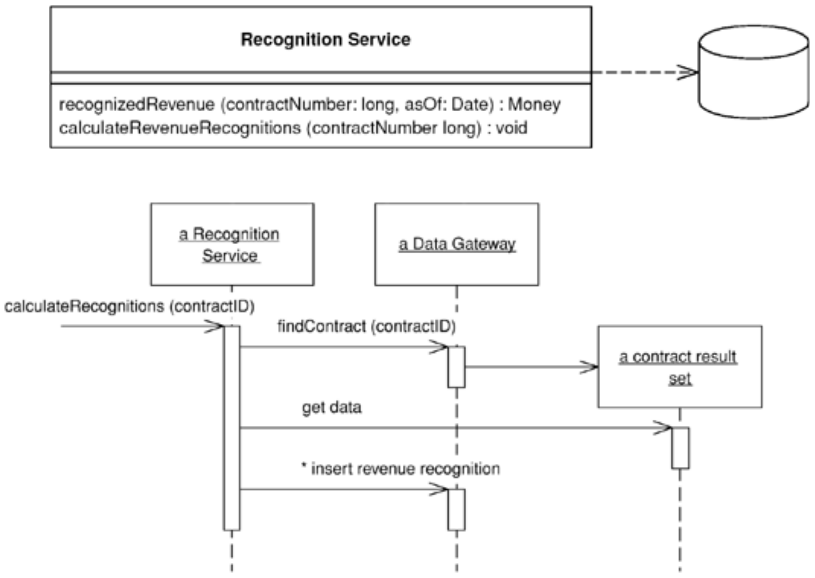
Alternativas habituales:

- **Rutinas**, a.k.a. transaction scripts.
- **Módulos de datos**, p.ej. programación visual
- **Modelo del dominio** (solución orientada a objetos).
- **Capa de servicio**

Caso práctico

Rutinas [transaction scripts]

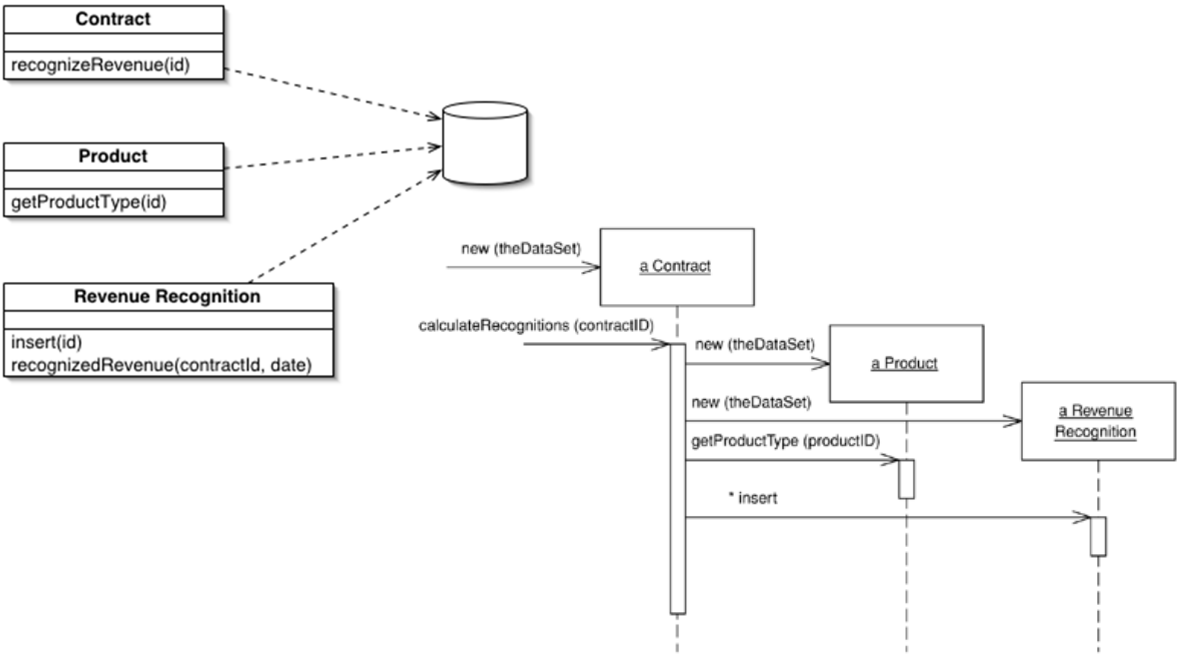
p.ej. Procedimientos almacenados (RDBMS)



Caso práctico

Módulos de datos [a.k.a. table modules]

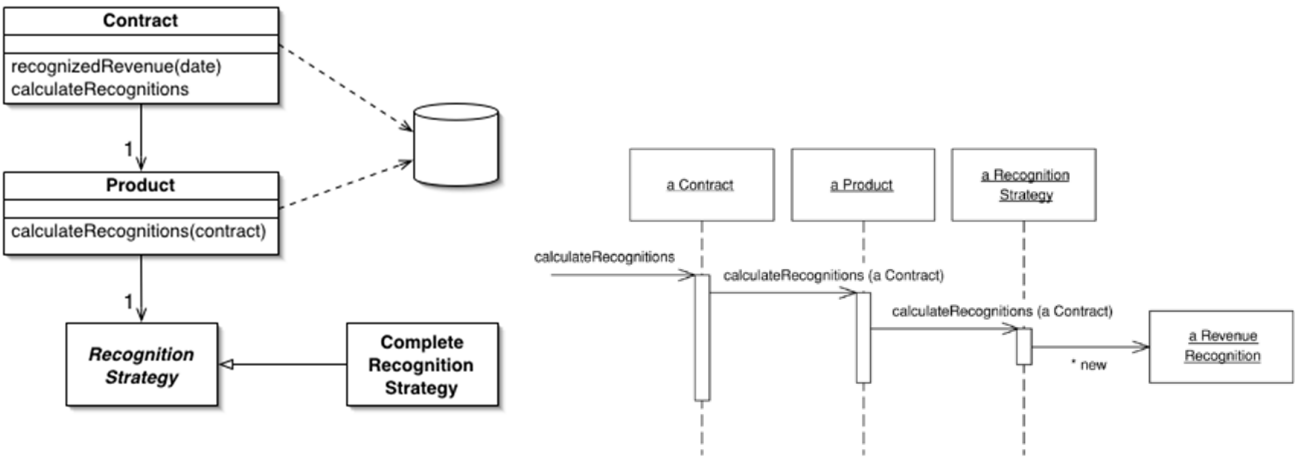
p.ej. Herramientas de programación visual



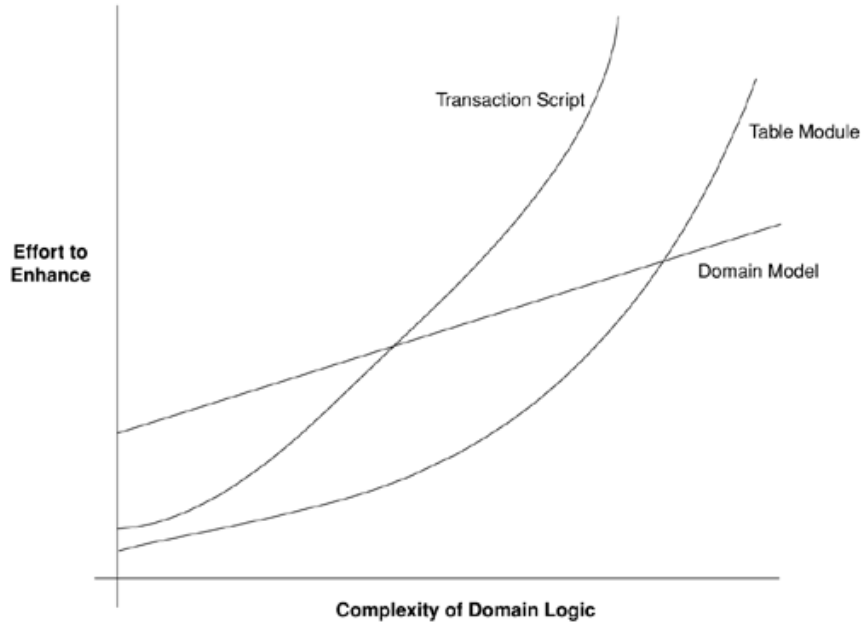
Caso práctico

Modelo del dominio [domain model]

Soluciones orientadas a objetos & MDSD



Caso práctico

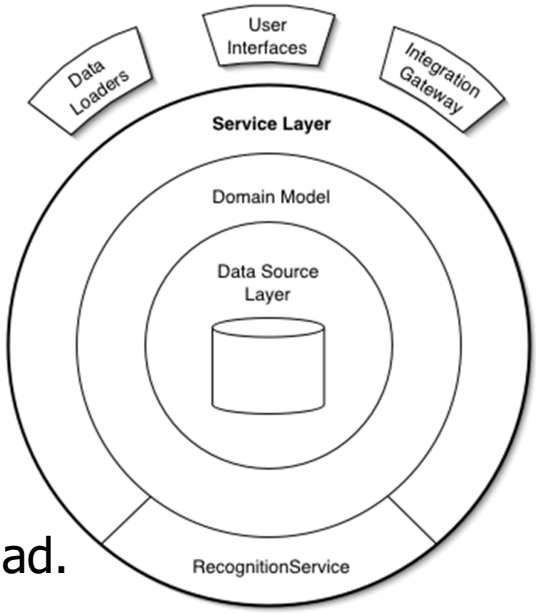


Complejidad y esfuerzo
asociados a cada estrategia de implementación

Caso práctico

Capa de servicio [service layer]

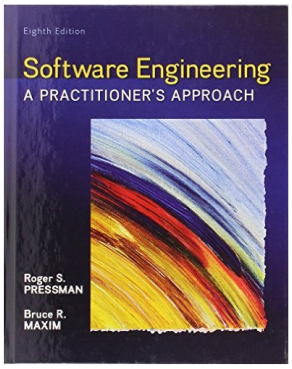
Capa intermedia entre la capa de presentación y el modelo del dominio:



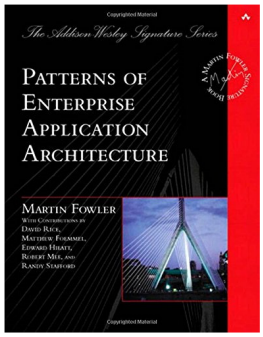
- Provisión de un API para distintas aplicaciones.
- Control de transacciones.
- Comprobaciones de seguridad.

Bibliografía recomendada

- Roger S. Pressman:
  Software Engineering:
  A Practitioner's Approach
  McGraw-Hill, 8th edition, 2014.
  ISBN 0078022126



- Martin Fowler:
  Patterns of Enterprise
  Application Architecture
  Addison-Wesley, 2003.
  ISBN 0321127420
  http://martinfowler.com/eaaCatalog/



Bibliografía adicional

Patrones de diseño

- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad & Michael Stal:
  Pattern-Oriented Software Architecture. Volume 1: A System of Patterns
  Wiley, 1996. ISBN 0471958697
- Gregor Hohpe & Bobby Woolf:
  Enterprise Integration Patterns.
  Addison-Wesley, 2003.
  ISBN 0321200683
- Paul Dyson & Andrew Longshaw:
  Architecting Enterprise Solutions: Patterns for High-Capability Internet-based Systems.
  Wiley, 2004. ISBN 0470856122
- Clifton Nock: **Data Access Patterns.**
  Addison-Wesley, 2003. ISBN 0321555627