Arquitetura e Padrões de Software

Dos princípios do design à nova visão sobre os objetos

Hugo Paredes
hparedes@utad.pt

Leonel Morgado
Leonel.Morgado@uab.pt

MESTRADO EM ENGENHARIA INFORMÁTICA E TECNOLOGIA WEB

utad

UNIVERSIDADE ABERTA
www.uab.pt

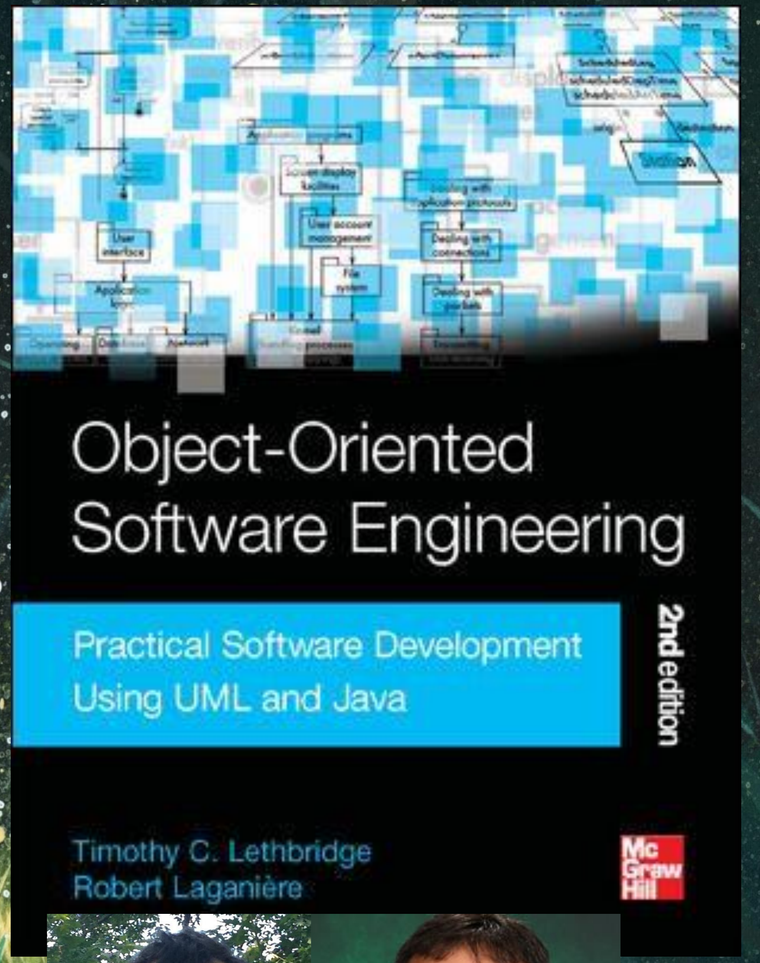
Alguns exemplos de princípios de design de software (de vários autores)

Decomposição

"***1.º Princípio do Design:***
dividir para reinar

Tentar lidar com algo grande tudo de uma vez é geralmente muito mais difícil do que lidar com uma variedade de coisas mais pequenas.

- *Pessoas diferentes podem trabalhar em cada peça.*
- *Um engenheiro de software pode especializar-se.*
- *Cada componente individual é mais pequeno, logo mais fácil de compreender.*
- *As peças podem ser substituídas ou alteradas sem ter de substituir ou mudar grandemente outras peças.*
- *Surgem oportunidades para reutilizar componentes.*"

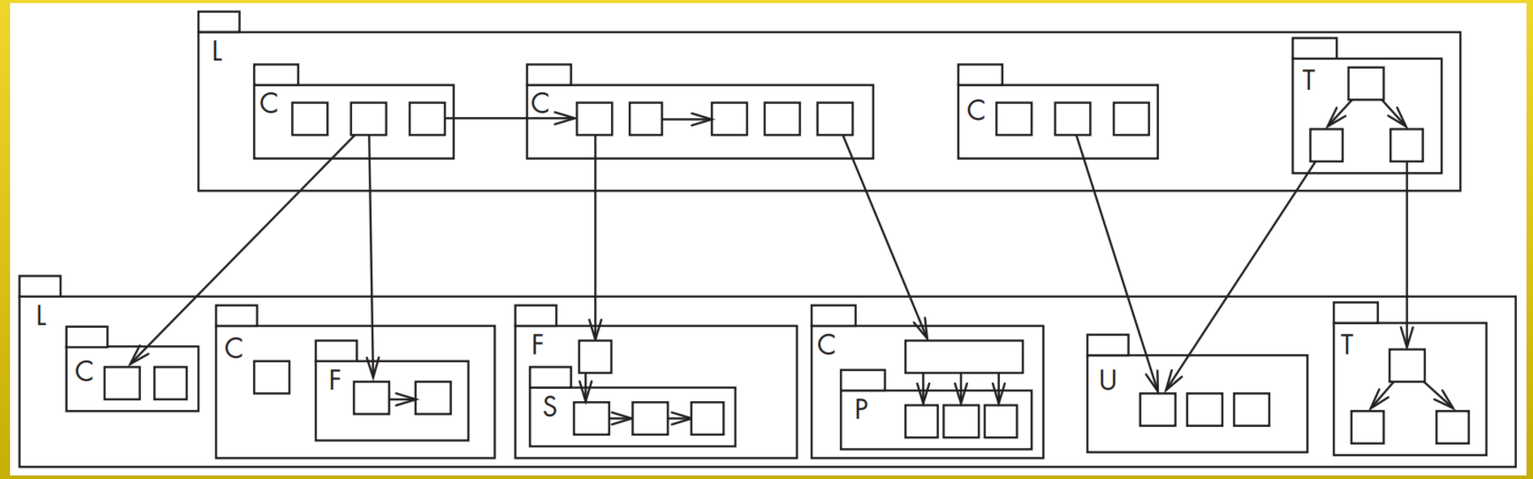


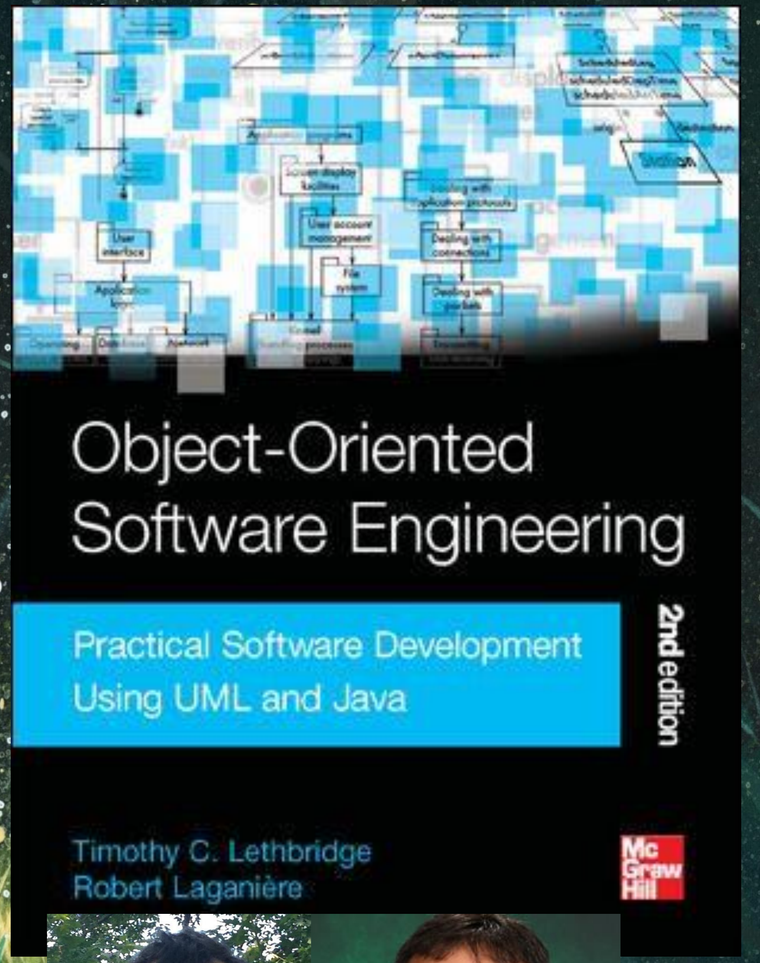
Coesão

"***2.º Princípio do Design:***
aumentar a coesão sempre que possível

"A Coesão diz para se fazer isso [da decomposição] *com inteligência: sim, dividam-se as coisas, mas mantenha--se junto o que deve ficar junto.*

Um subsistema ou módulo tem coesão alta se mantiver juntas coisas relacionadas entre si, e não mantiver outras coisas. Isto faz com que o sistema em geral seja mais fácil de compreender e de alterar."



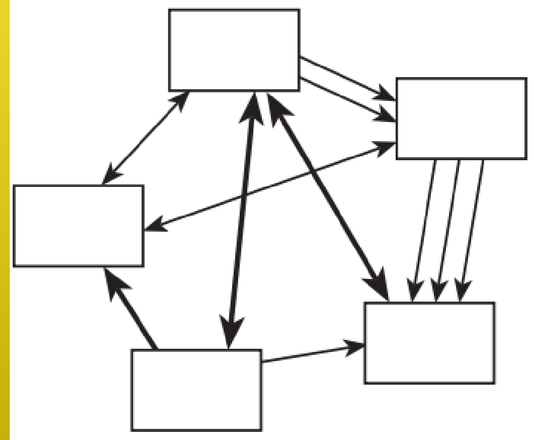


Acoplamento

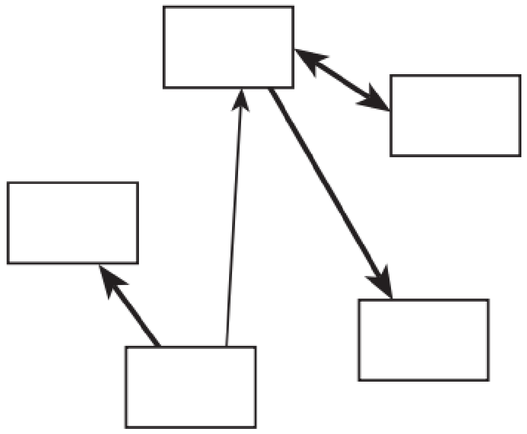
*"**3.º Princípio do Design:***
reduzir o acoplamento onde for possível

"Ocorre acoplamento quando há interdependências entre módulos. (…) Em geral, quanto mais fortemente acoplado estiver um conjunto de módulos, mais difícil é compreendê-lo e, consequentemente, mais difícil é mudar o sistema."

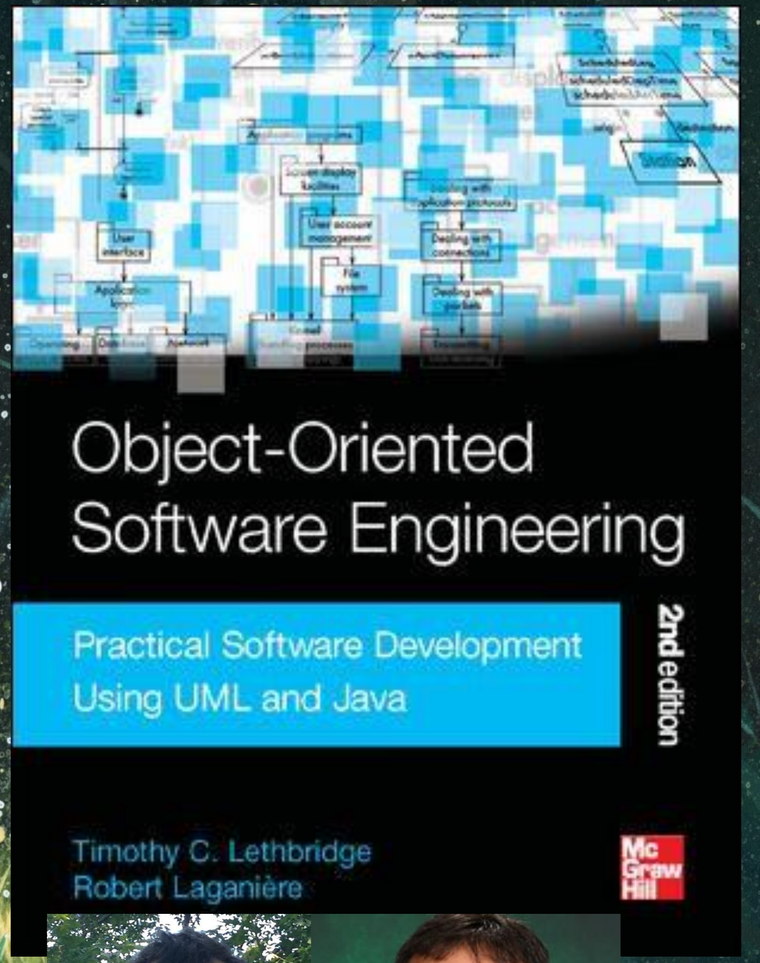
Forte (indesejável)



Fraco (desejável)



Força do acoplamento proporcional à espessura:



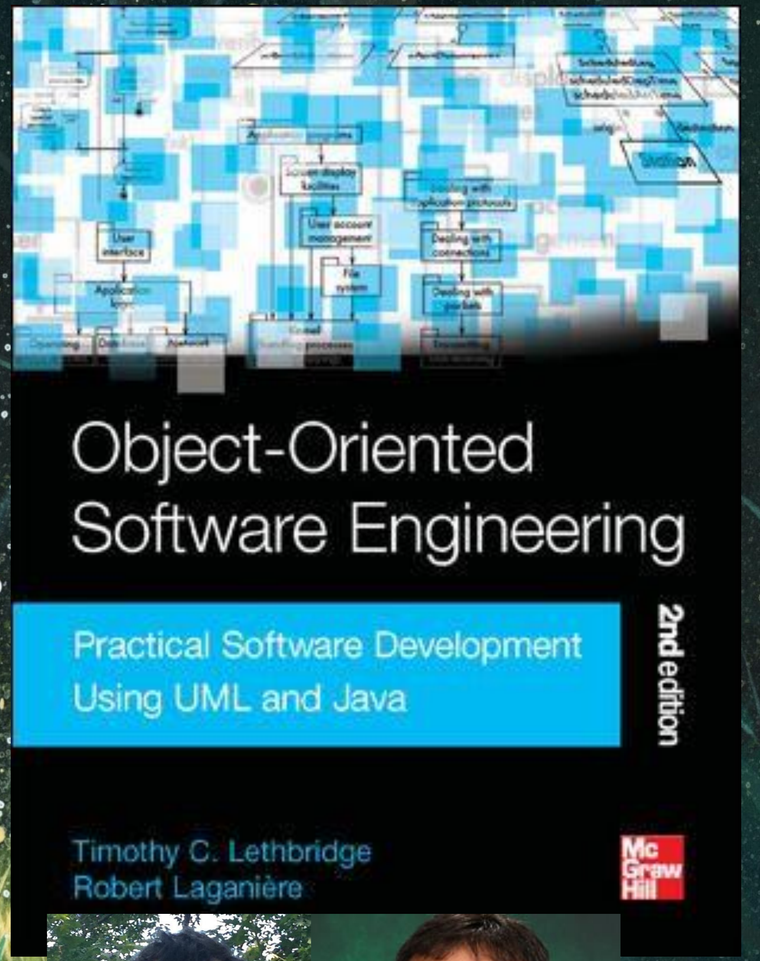
Abstração

"***4.º Princípio do Design:***
manter o nível de abstração tão alto quanto seja possível

"Deve-se assegurar que (…) é possível ocultar ou adiar as preocupações com os detalhes, reduzindo assim a complexidade."

"Algumas abstrações, como classes e métodos, são suportadas diretamente pela linguagem de programação. Outras, como as associações, existem apenas nos modelos usados pelo designer."

"As abstrações funcionam, permitindo compreender a essência de algo e tomar decisões importantes sem conhecer detalhes desnecessários."

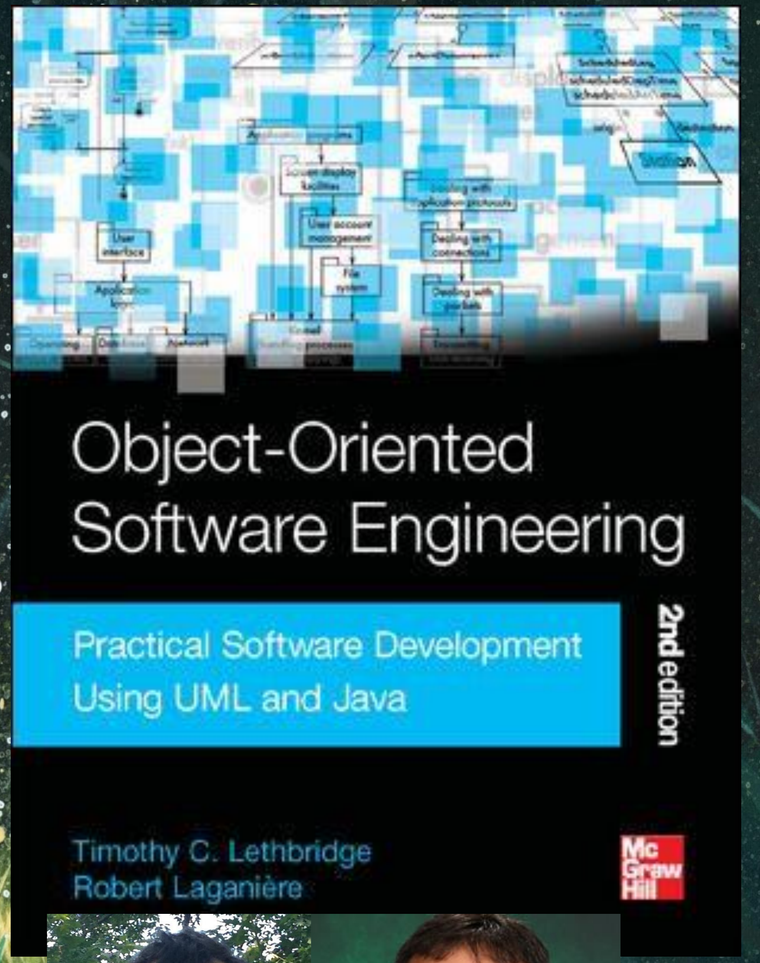


Reutilizabilidade

"***5.º Princípio do Design:***
aumentar a reutilizabilidade se possível

"O design para a reutilizabilidade significa que se concebem vários aspetos do sistema para poderem voltar a ser utilizados noutros contextos, tanto no próprio sistema como noutros (…):

- *Generalize o design tanto quanto possível*
- *Siga os três princípios de design anteriores. Aumentar a coesão (…) Reduzir o acoplamento (…) Aumentar a abstração (…)*
- *Conceba o sistema com ganchos (…)* [que são pontos intencionalmente] *para outros designers acrescentarem funcionalidades.*
- *Simplifique o design tanto quanto possível*

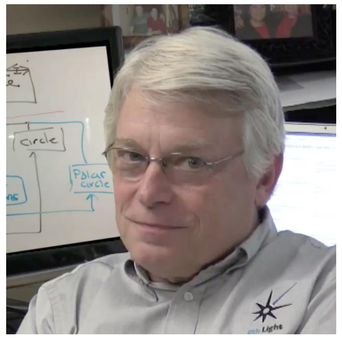


Dependências acíclicas

www.objectmentor.com 1

Design Principles and Design Patterns

Robert C. Martin
www.objectmentor.com

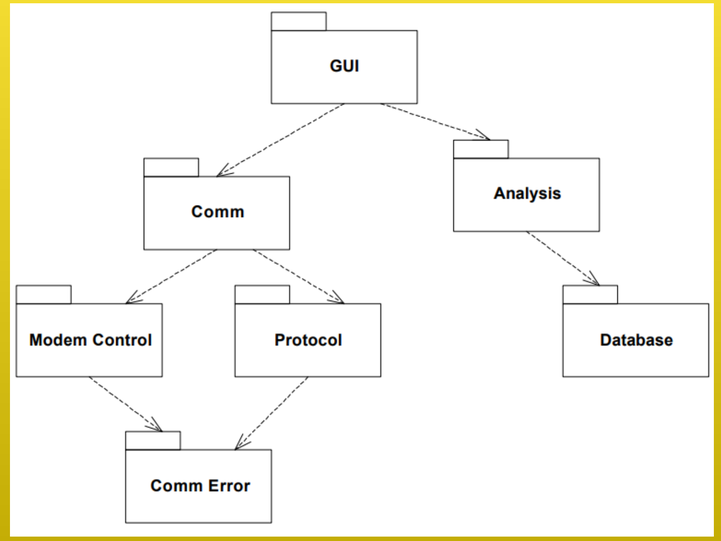


Robert Cecil Martin
(vg. “Uncle Bob”)

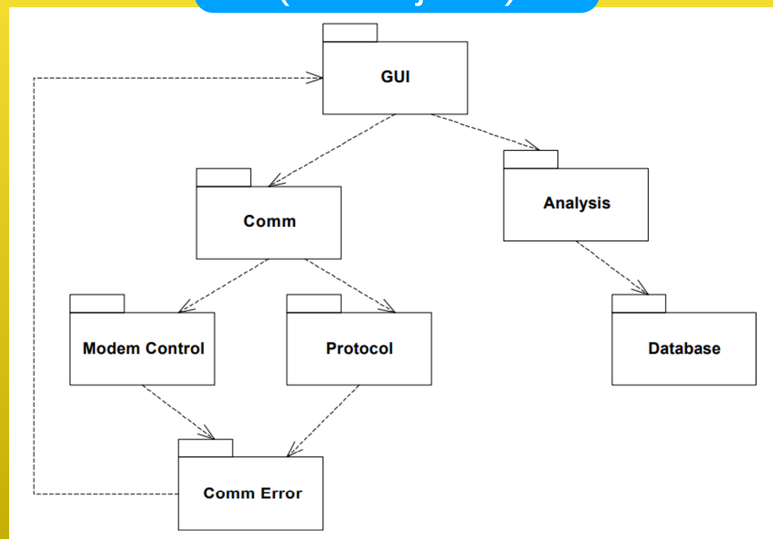
“***Princípio das dependências acíclicas***

As dependências entre componentes não devem formar ciclos.”

Acíclicas (desejável)



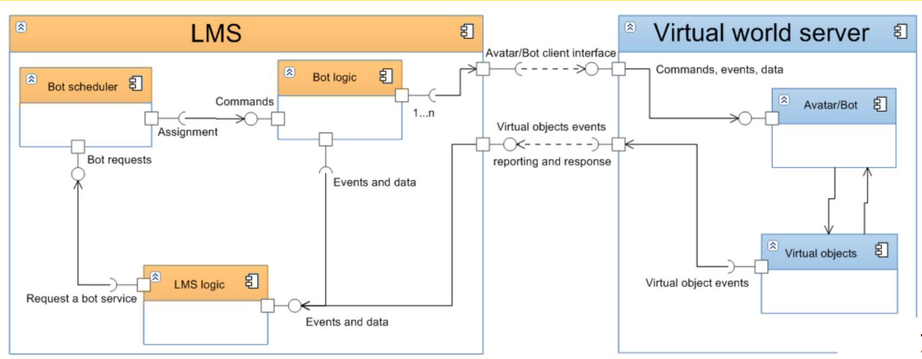
Com um ciclo (indesejável)



Separação de responsabilidades

“separation of concerns”

“A separação de responsabilidades é um tipo de estratégia de dividir para reinar usada por engenheiros de software. (…) modularização de código e design orientado a objetos”



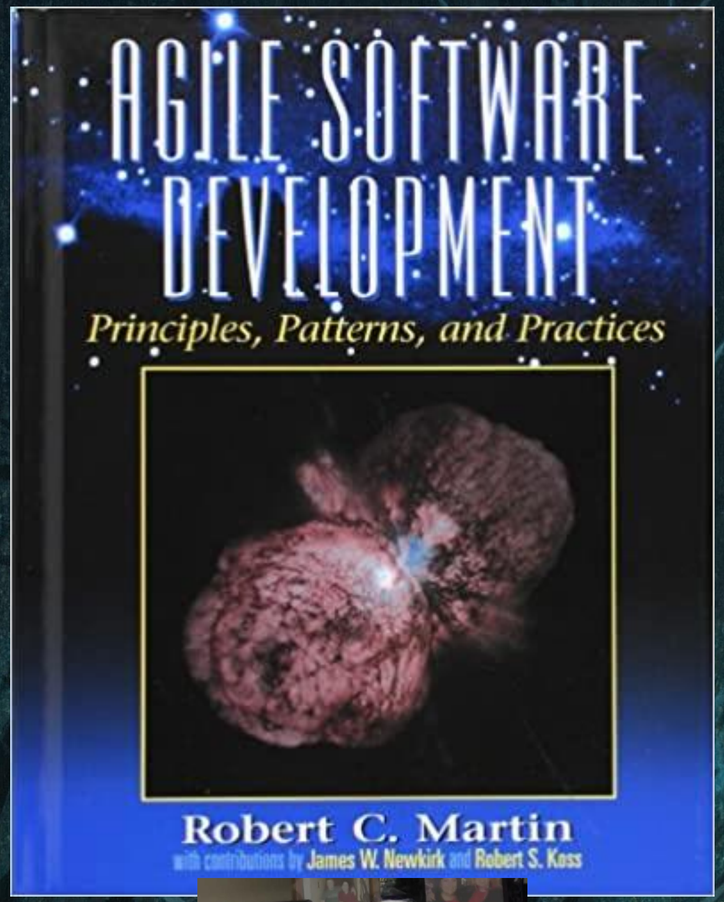
Exemplo:

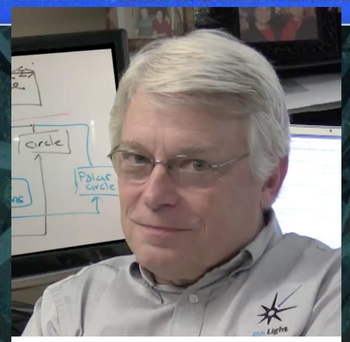
Journal of Universal Computer Science, vol. 22, no. 2 (2016), 271-297
submitted: 2/1/15, accepted: 29/1/16, appeared: 1/2/16 © J.UCS

A Bot Spooler Architecture to Integrate Virtual Worlds with E-learning Management Systems for Corporate Training

«(…) o desenvolvimento (…) tem de separar responsabilidades e ter maior independência entre o conteúdo (…) e a plataforma (…) onde este é fornecido. A arquitetura MULTIS constitui um passo nessa direção (…)»

Aberto-Fechado



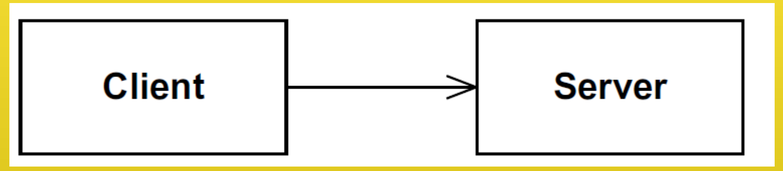


Robert Cecil Martin
(vg. “Uncle Bob”)

“***Princípio de ser Aberto-Fechado***

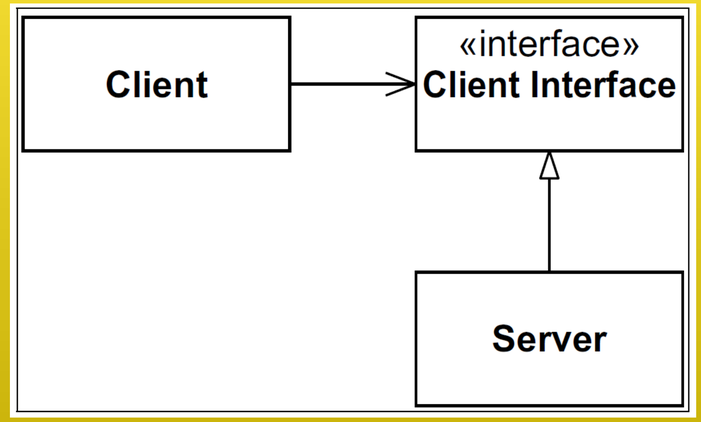
As entidades de software (classes, módulos, funções, etc.) devem ser abertas à extensão, mas fechadas à modificação.”

Design rígido (indesejável)



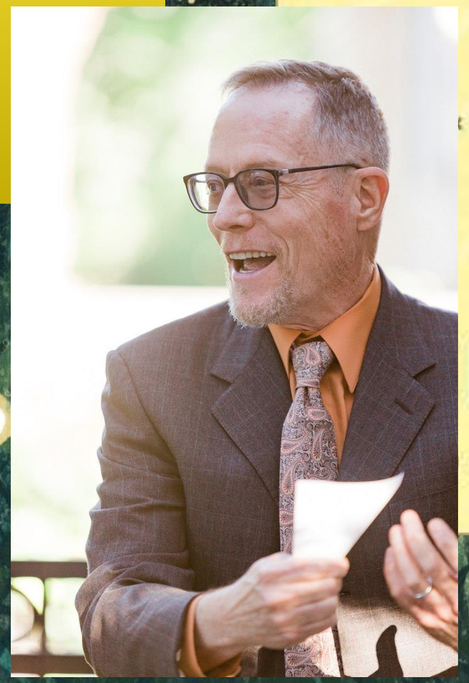
Cliente usa Servidor. Para novas funcionalidades, um design rígido requer modificação. Um design aberto-fechado permite que novas classes sejam usadas sem modificar as atuais.

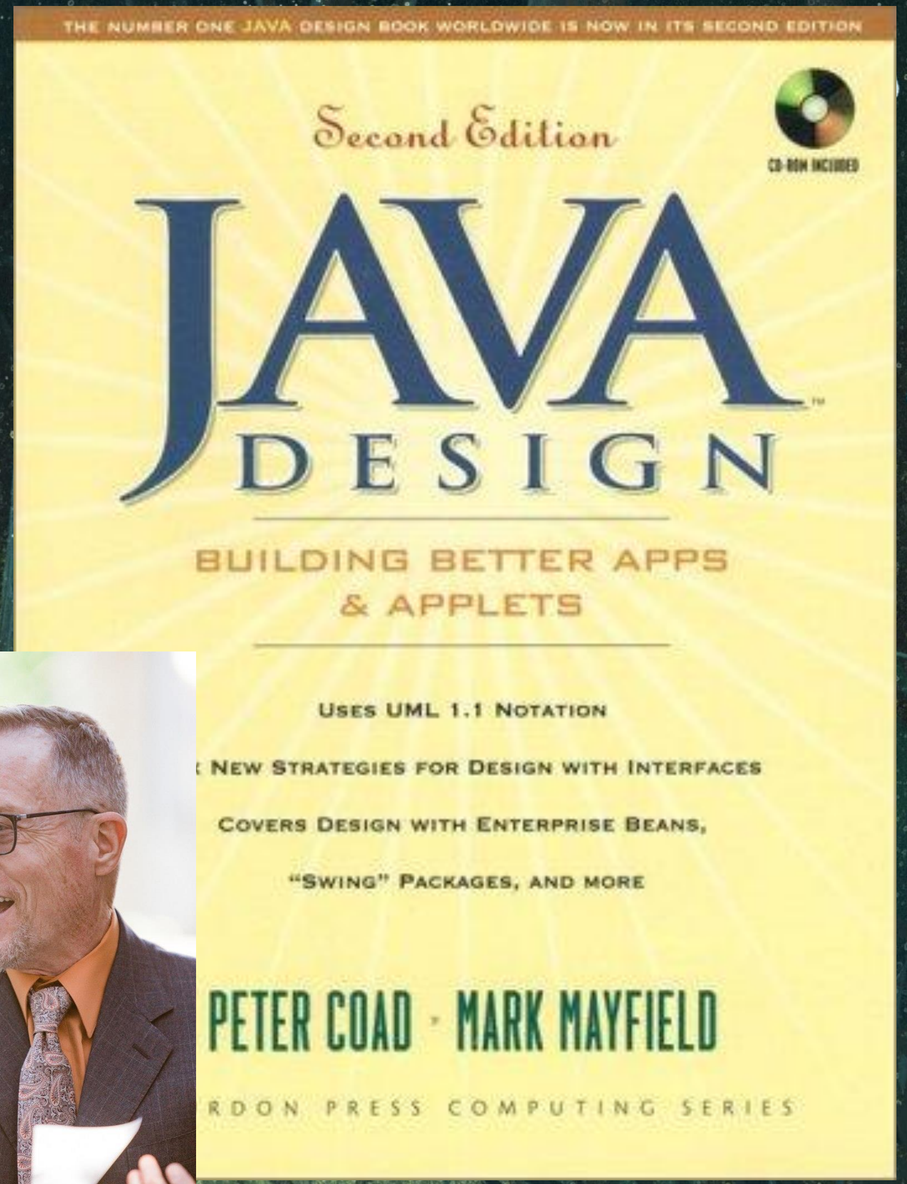
Design aberto-fechado (desejável)



Estratégias de design

- *Design com composição em vez de herança*
- *Design com interfaces*
- *Design com threads*
- *Design com notificação*

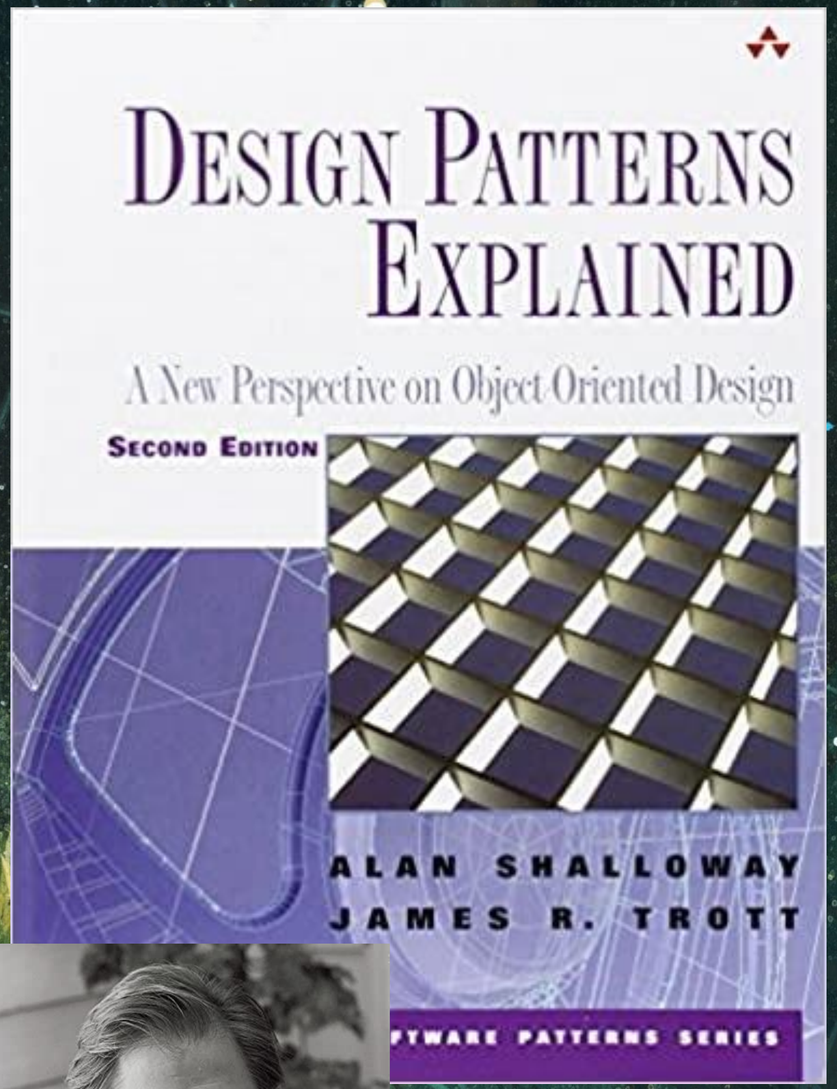




Coad, Peter, and Mark Mayfield. Java design: building better apps and applets. Yourdon Press, 1997.

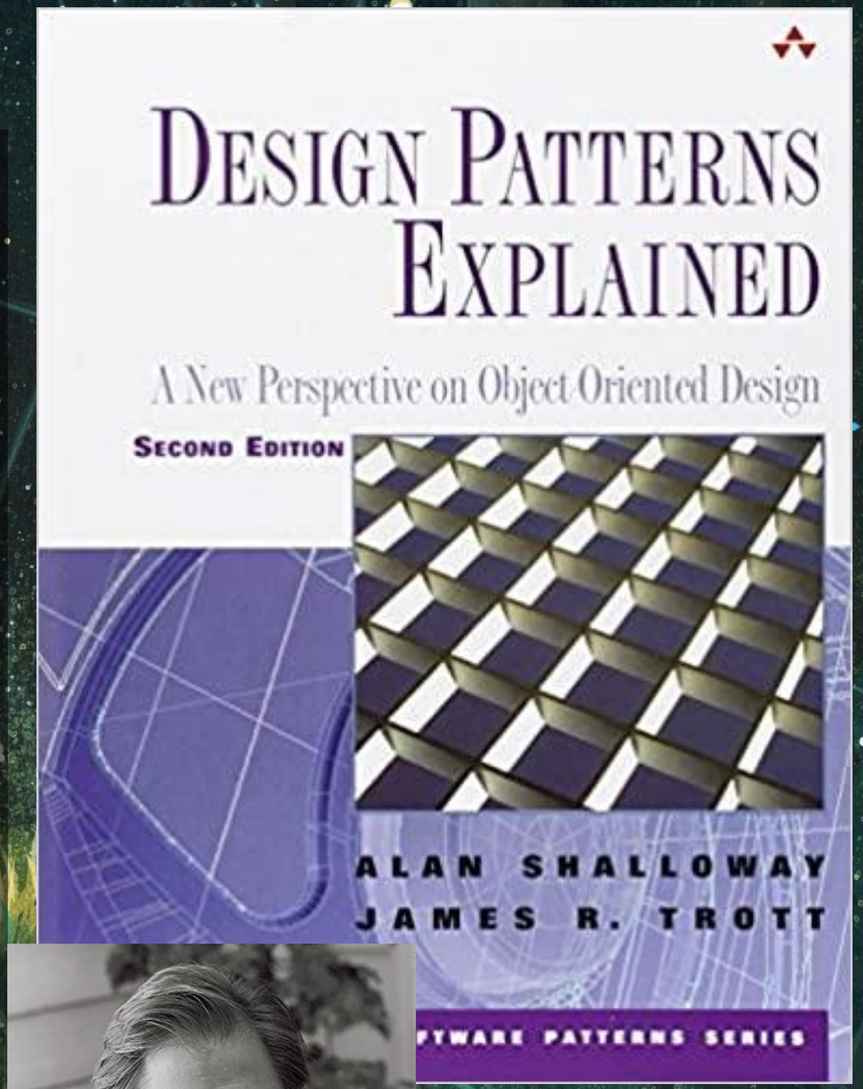
“Os *designers* têm de pensar em orientação a objetos de uma forma nova, renovada e não tradicional:

especialmente, pensar e três conceitos: objetos, encapsulamento e herança”



Uma nova visão sobre os objetos, que encaram como:

- Entidades com responsabilidades
- Interessa o que fazem, não como
- Algo que se vê de fora para dentro
- Elementos de design vistos assim, sem conhecer os detalhes internos



E por fim, o design considera uma multiplicidade de vertentes:

Compatibilidade
Extensibilidade
Tolerância a falhas
Capacidade de manutenção
Modularidade
Fiabilidade
Reutilizabilidade
Robustez
Segurança
Usabilidade
Desempenho
…