Engenharia de Software

Arquitectura de Software

Luís Morgado

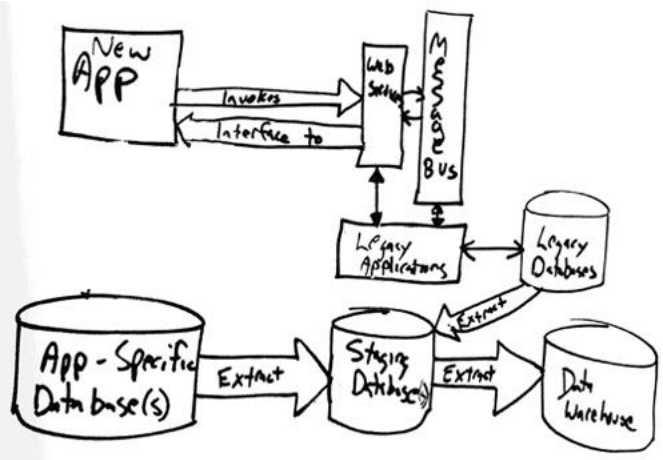
Instituto Superior de Engenharia de Lisboa
Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Arquitectura de Software

Arquitectura de software é a definição da *organização fundamental de um sistema, concretizada nos seus componentes, nas suas relações entre si e com o ambiente, e nos princípios que regem a sua concepção e evolução*

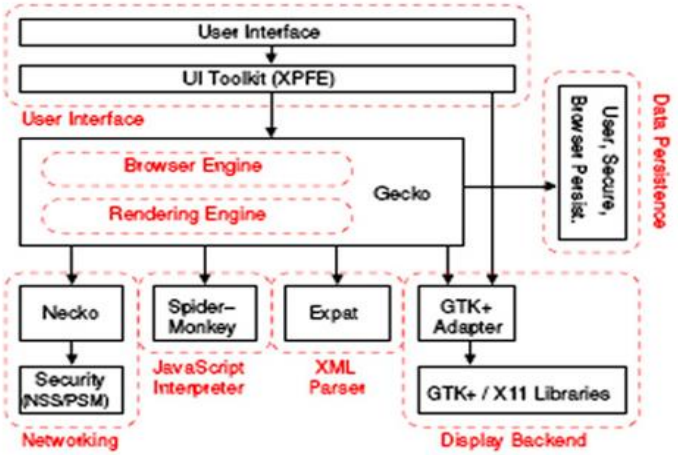
[ANSI/IEEE Std 1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems*]

Enterprise Architecture



McGovern, J., Ambler, S., Stevens, M., Linn, J., Sharan, V., and Jo, E. *A Practical Guide to Enterprise Architecture*. Upper Saddle River, New Jersey: Prentice Hall, 2004

Mozilla Web Browser



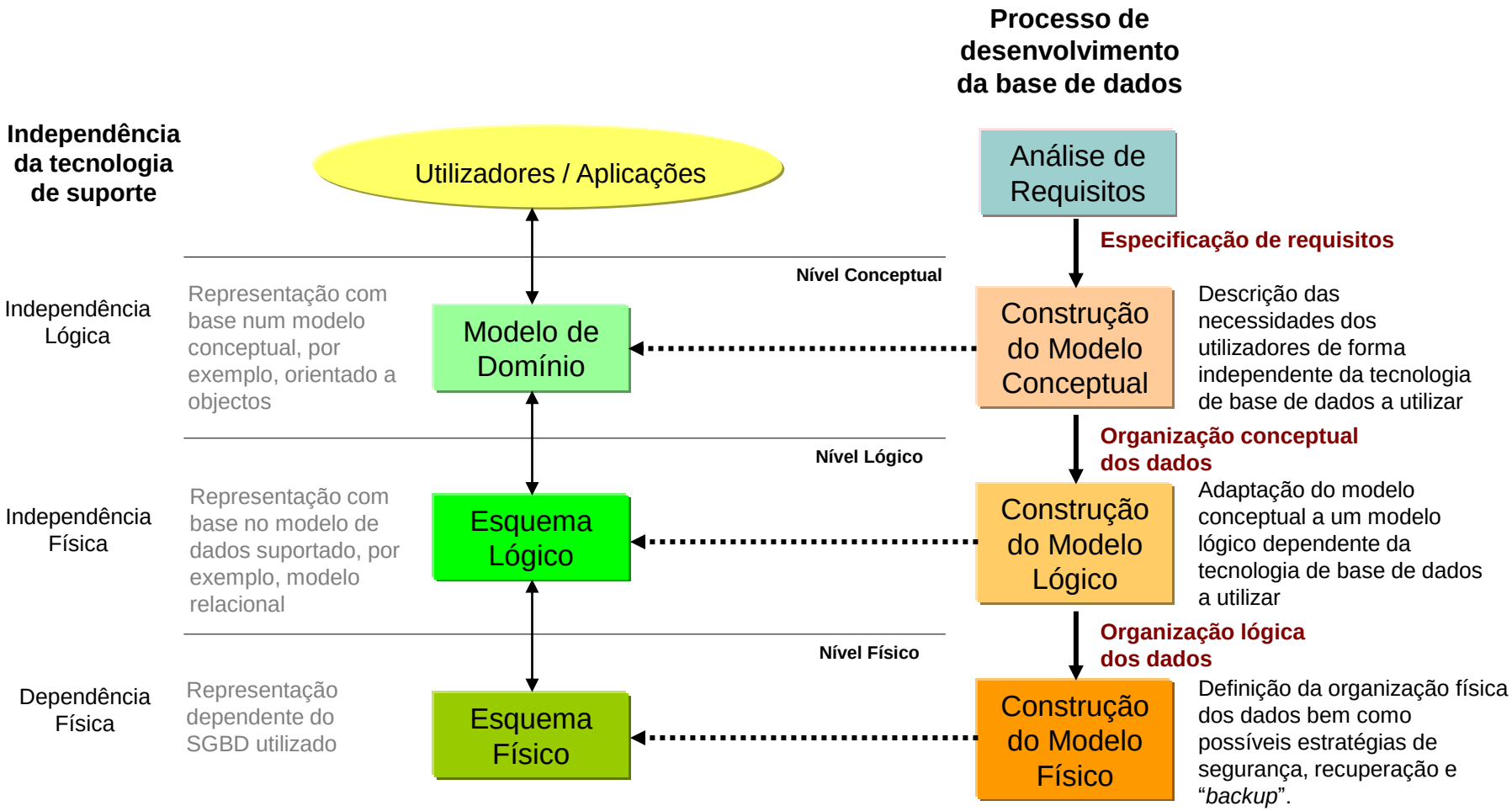
Grosskurth, A. and Godfrey, M. "A Reference Architecture for Web Browsers." *IEEE Conference on Software Maintenance*, 2005

Tipos de Arquitectura de Software

- **Arquitectura conceptual**
  - Representação de alto nível de conceitos e modos de organização de um sistema
  - Proporciona uma visão geral dos conceitos principais de organização de um sistema, das suas relações e respectivas responsabilidades
  - É independente de um suporte lógico ou físico específico
- **Arquitectura lógica**
  - Representação mais concreta da organização de um sistema, mas independente da tecnologia
  - Inclui os elementos necessários à descrição do sistema sem estar dependente de tecnologias específicas
  - É independente de um suporte físico específico
- **Arquitectura física**
  - Representação concreta de um sistema, incluindo pormenores de implementação dependentes da tecnologia, por exemplo, plataforma de suporte de execução
  - Especifica a implantação física do sistema e a forma como os seus componentes são distribuídos por diferentes máquinas ou ambientes de operação
  - É dependente de um suporte físico específico

Tipos de Arquitectura de Software

Exemplo: Arquitectura de dados



Arquitectura de Software

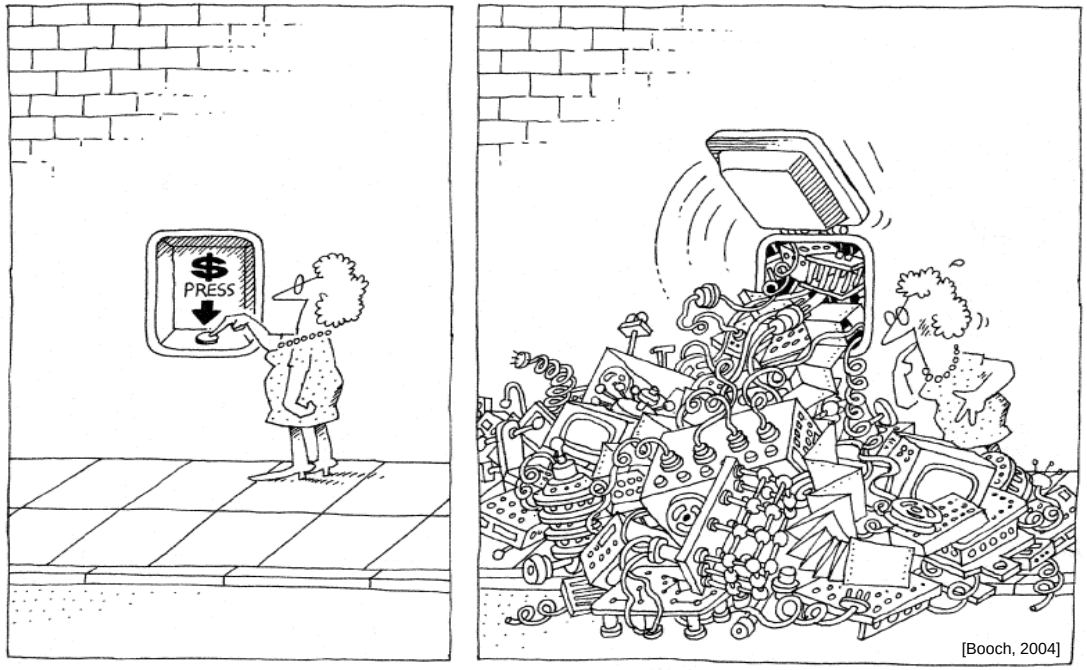
Orientada para abordar o problema da complexidade com base em três vertentes principais: *métricas*, *princípios* e *padrões*

- **MÉTRICAS**
- **PRINCÍPIOS**
- **PADRÕES**

Objectivo: reduzir a complexidade desorganizada e controlar a complexidade organizada necessária à função do sistema

COMPLEXIDADE

- Redução
- Controlo



Quando a complexidade não é adequadamente resolvida, mas apenas acumulada, expressar-se-á em algum momento…

Métricas de Arquitectura

Definem *medidas de quantificação* de uma arquitectura de software *indicadoras da qualidade* dessa arquitectura

- **Coesão**
  - Nível coerência na forma como os elementos de um sistema ou parte de um sistema estão agrupados
- **Acoplamento**
  - Grau de interdependência entre partes de um sistema
- **Simplicidade**
  - Nível de facilidade de compreensão/comunicação da arquitectura
- **Adaptabilidade**
  - Nível de facilidade de alteração da arquitectura para incorporação de novos requisitos ou de alterações nos requisitos previamente definidos

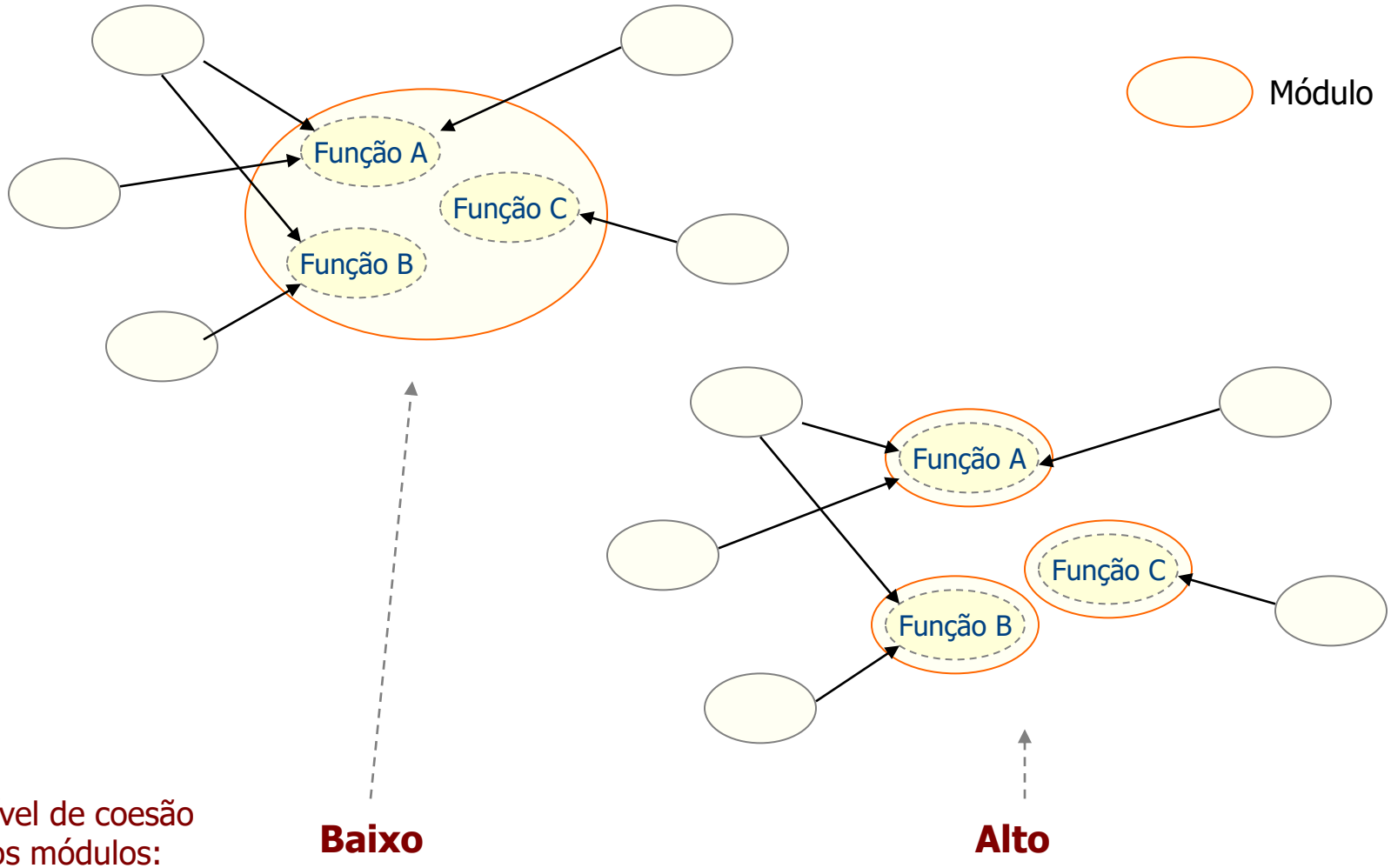
Coesão

- O conceito de *coesão* refere-se à forma como os elementos de um sistema estão agrupados de forma coerente entre si, será tanto maior quando mais relacionados entre si forem os elementos agrupados em cada módulo
- A coesão pode ser medida utilizando diferentes critérios, por exemplo, por tipo de função das partes agrupadas, sendo designada neste caso por *coesão funcional*
- O objetivo é criar software que seja claro e fácil de entender, manter e evoluir, facilitando a reutilização e aumentando assim a eficiência do desenvolvimento
- A *modularidade* e a *factorização* são princípios de arquitectura de software que contribuem para o aumento da coesão

- **COESÃO**
  - Nível de coerência das partes agrupadas num módulo ou subsistema
  - Característica **intra-modular**
    - Expressa relações interiores aos módulos (agrupamentos)
  - Deve ser maximizada
    - Redução de complexidade, facilidade de evolução, manutenção e reutilização

Coesão

Exemplo



Nível de coesão dos módulos:

Baixo

Alto

Coesão

Nível de coesão:

- **Um módulo com um nível de coesão baixo é mais complexo**, logo mais difícil de conceber e de testar
- Um nível de coesão baixo leva a que, em caso de necessidade de **alteração** de um subsistema, o **número de módulos afectados seja elevado**
- Se o nível de coesão for elevado, o número de módulos afectados será minimizado

Tipos de Coesão

- **Coesão funcional**
  - Partes que contribuem para uma função específica bem definida são agrupadas num módulo
- **Coesão lógica**
  - Partes do mesmo tipo são agrupadas num módulo
- **Coesão temporal**
  - Partes que são executadas em momentos próximos no tempo são agrupadas num módulo

Acoplamento

- O conceito de *acoplamento* refere-se ao grau de dependência entre diferentes partes de um sistema, sendo tanto maior quanto maior a dependência entre as partes de um sistema
- O acoplamento pode ser medido utilizando diferentes métricas, como o número de associações entre elementos ou o grau de complexidade de uma interface
- O objetivo é criar software com o menor acoplamento possível, de modo a reduzir a complexidade e a facilitar a sua manutenção e evolução
- A *modularidade* e o *encapsulamento* são princípios de arquitectura de software que contribuem para a redução do acoplamento
- **ACOPLAMENTO**
  - Grau de interdependência entre partes de um sistema
  - Característica **inter-modular**
    - Expressa relações entre partes (módulos)
  - Deve ser minimizado
    - Redução de complexidade, facilidade de manutenção e evolução

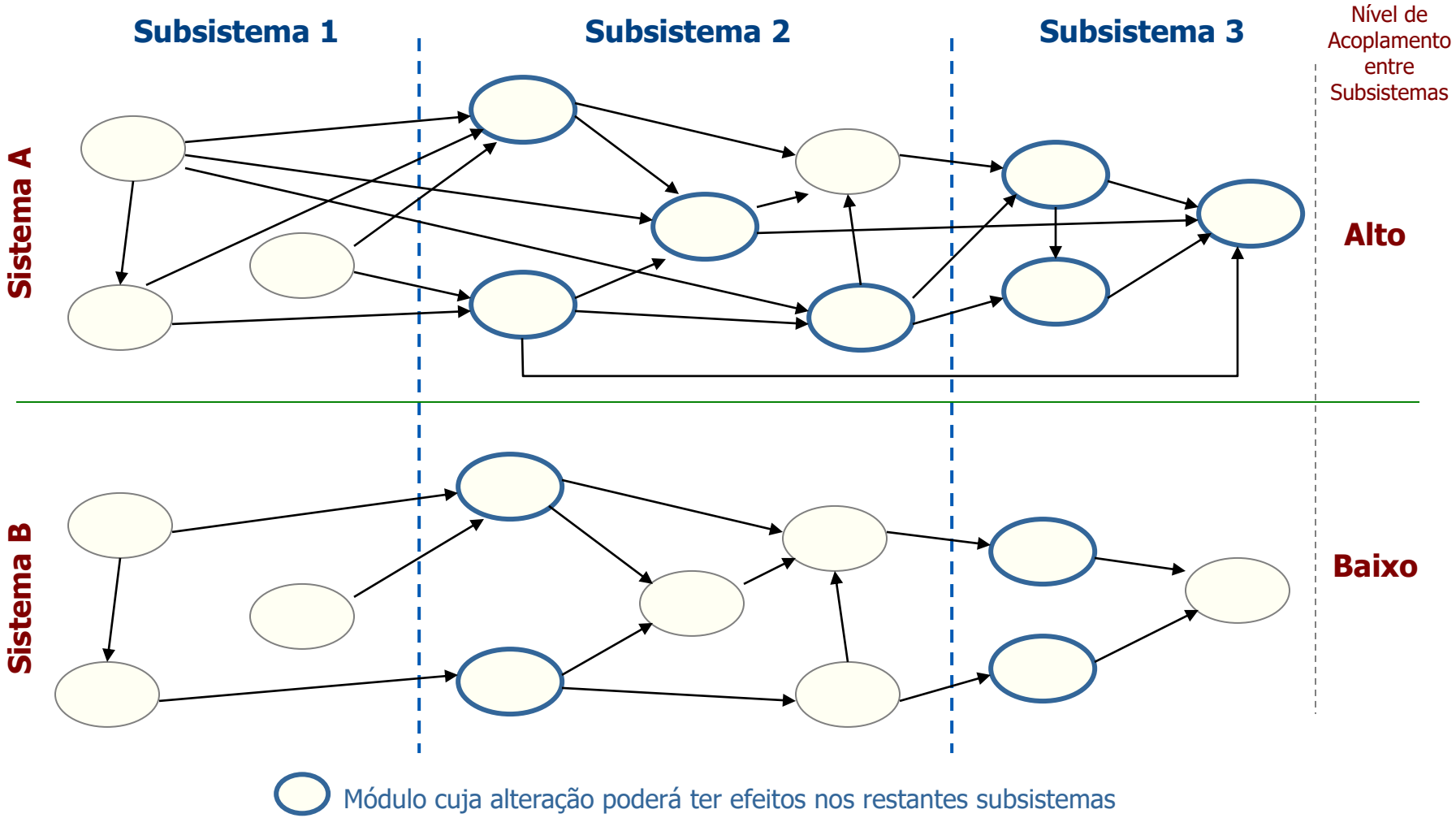
Acoplamento

A redução do nível de acoplamento permite:

- **Maior facilidade de desenvolvimento, instalação, manutenção e expansão** do software
- **Melhor escalabilidade**, devido à possibilidade de distribuição e replicação de módulos que prestem serviços, sem que isso tenha um impacto significativo nos clientes desses subsistemas/módulos
- **Maior tolerância a falhas**, logo maior robustez, uma vez que a falha de um subsistema/módulo tem um impacto restrito

Acoplamento

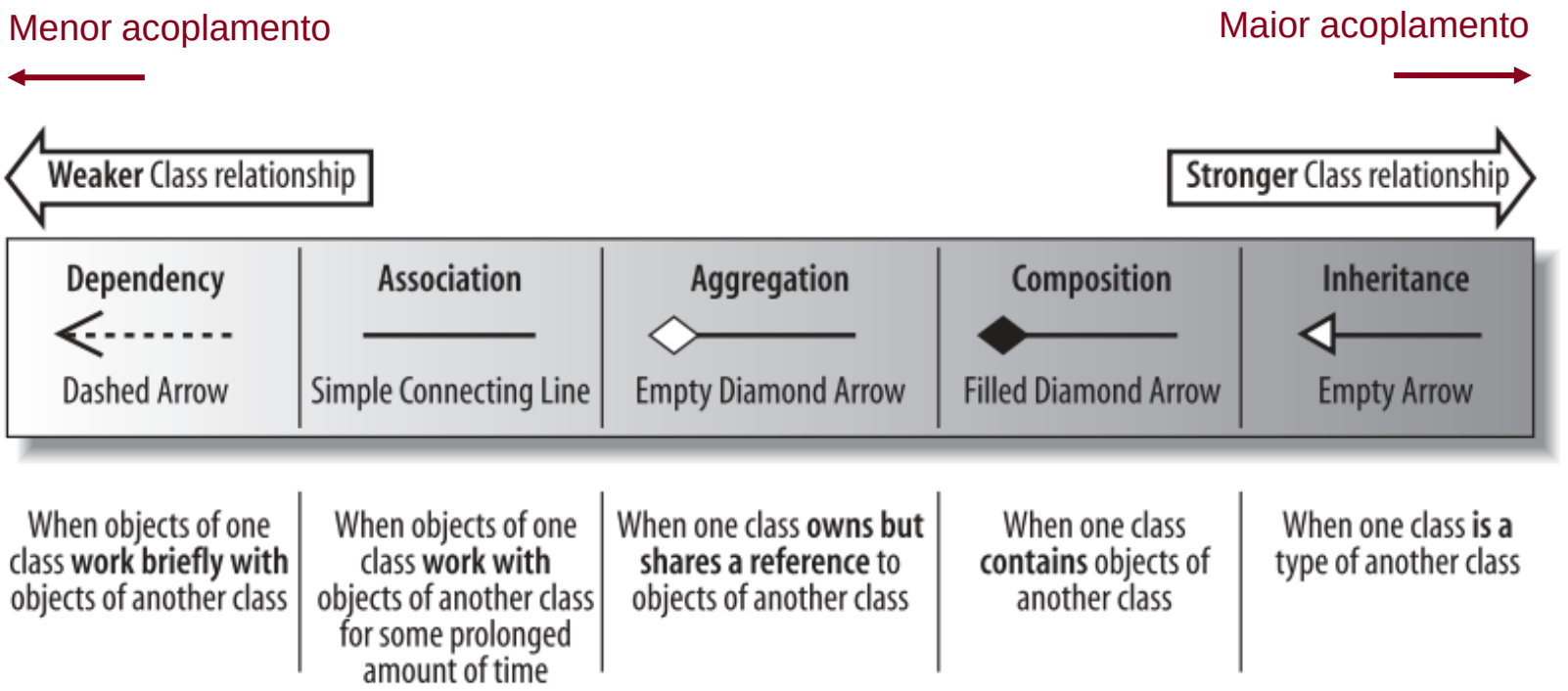
Exemplo



Acoplamento em Modelos de Estrutura

Linguagem UML

Relações entre classes e nível de acoplamento



[Miles & Hamilton, 2006]

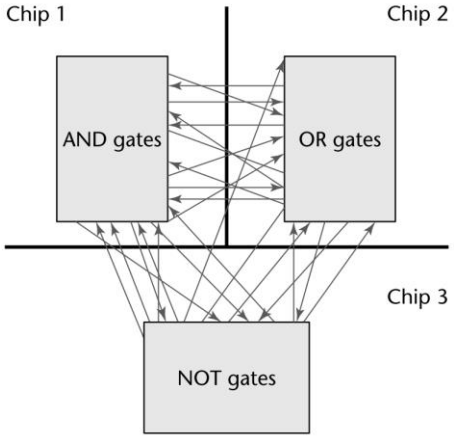
Tipos de Acoplamento

- **Acoplamento estrutural**
  - Uma parte depende estruturalmente de outra parte, podendo directamente utilizar, aceder ou alterar outra parte
- **Acoplamento estrutural comum**
  - Múltiplas partes podem aceder ou alterar uma parte comum
- **Acoplamento funcional**
  - Uma parte depende de um contracto funcional (interface), independentemente da parte que o implementa
- **Acoplamento denotacional (semântico)**
  - Uma parte depende do significado associado a uma característica ou funcionalidade, independentemente da parte que o implementa e do contracto funcional com que é definido

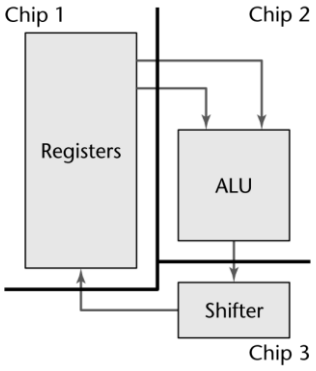
Coesão e Acoplamento

- **Coesão**
  - Nível coerência funcional de um subsistema/módulo (até que ponto esse módulo realiza uma única função)
  - Característica **intra-modular**
- **Acoplamento**
  - Grau de interdependência entre partes
  - Característica **inter-modular**

Exemplo: Organização de um sistema digital



Organização por tipo de porta lógica
Baixa coesão
Alto acoplamento

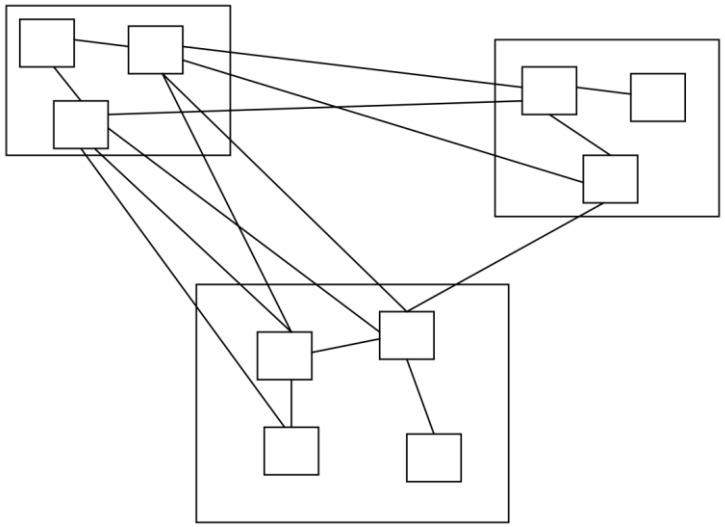


[Schach, 2010]

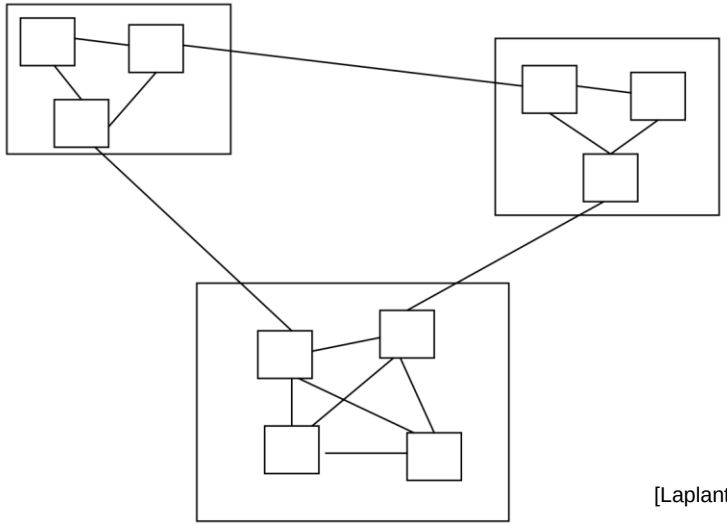
Organização por tipo de funcionalidade
Alta coesão
Baixo acoplamento

Coesão e Acoplamento

Exemplo



Baixa coesão
Alto acoplamento



[Laplante, 2007]

Alta coesão
Baixo acoplamento

Princípios de Arquitectura

Definem *meios orientadores* da concepção de arquitectura de software, no sentido de garantir a qualidade da arquitectura produzida, nomeadamente, no que se refere à *minimização do acoplamento*, à *maximização da coesão* e à *gestão da complexidade*

- **Abstracção**
- **Modularidade**
- **Encapsulamento**
- **Factorização**

Subjacente

- Maximização da coesão
- Minimização do acoplamento
- Redução e gestão da complexidade

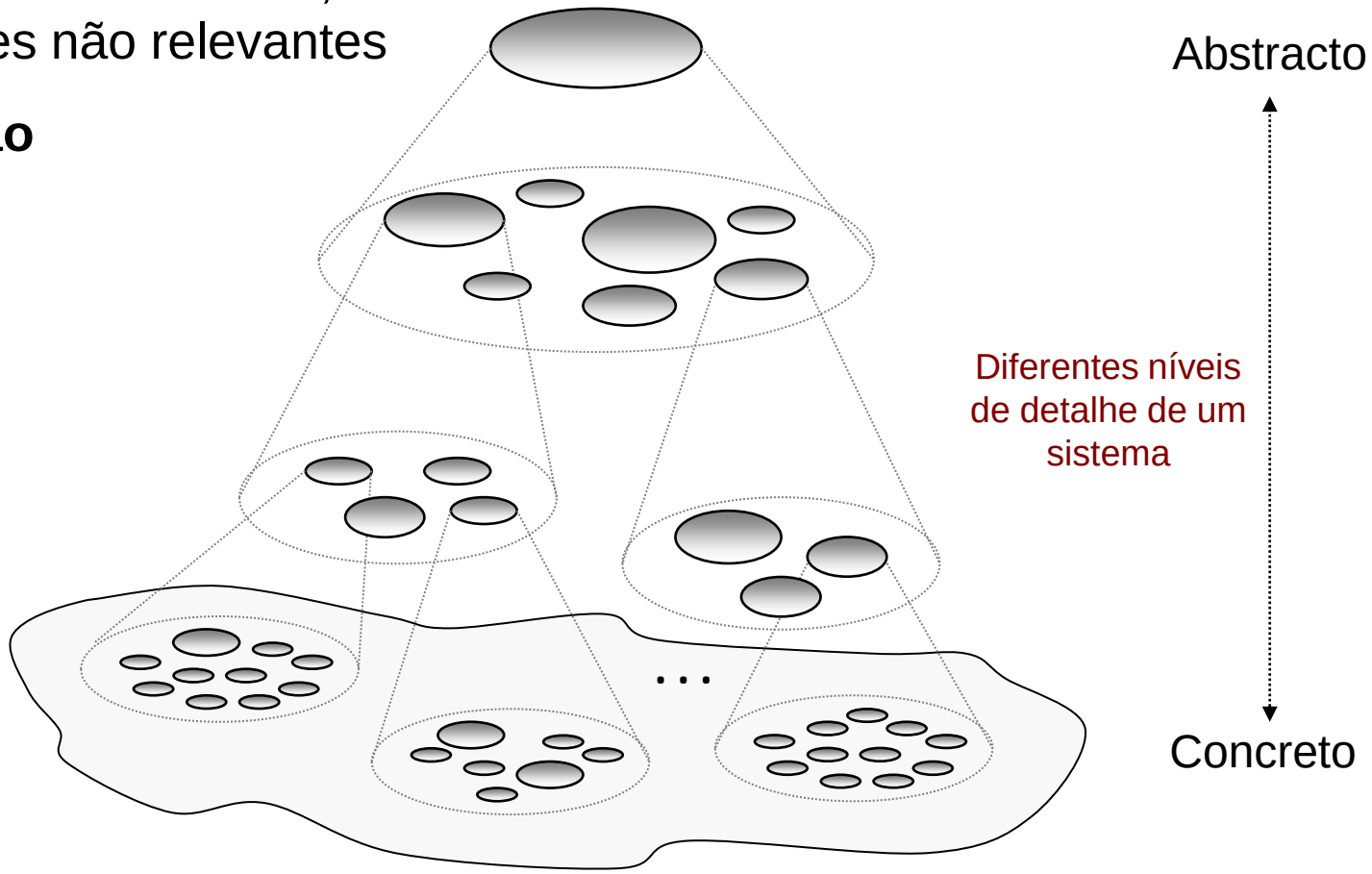
Abstracção

- Aspecto principal da modelação de sistemas
- Processo de descrição de conhecimento a *diferentes níveis de detalhe* (quantidade de informação) e *tipos de representação* (estrutura da informação) [Korf, 1980]
- **A abstracção é uma ferramenta base para lidar com a complexidade**
  - Identificação de características comuns a diferentes partes
  - Realçar o que é essencial, omitir detalhes não relevantes
  - Modelos, representações abstractas de um sistema
- **Desenvolvimento de um sistema complexo**
  - Criação de ordem de forma progressiva através de diferentes níveis de abstracção
  - Processo iterativo guiado por conhecimento

Abstracção

Ferramenta base para lidar com a complexidade

- **Realçar** o que é **essencial**, omitir detalhes não relevantes
- **Simplificação**
- **Focagem**



Modularidade

- O conceito de *modularidade* refere-se à capacidade de organização de um sistema em partes coesas, ou módulos, que podem ser interligados entre si para produzir a função do sistema
- Está relacionado com dois aspectos principais, *decomposição* e *encapsulamento*
  - **Decomposição**
    - Dividir o sistema em partes pequenas, coesas, mais fáceis de compreender e descrever, que podem ser modeladas separadamente
      - Para sistematizar interacções
      - Para lidar com a explosão combinatória
  - **Encapsulamento**
    - Isolamento dos detalhes internos das partes de um sistema ou módulo em relação ao exterior
- A modularidade permite trabalhar em partes específicas do sistema sem afetar outras partes, o que pode aumentar a eficiência e a qualidade do desenvolvimento de software
- A modularidade é alcançada através de boas práticas de arquitectura de software, como coesão e encapsulamento

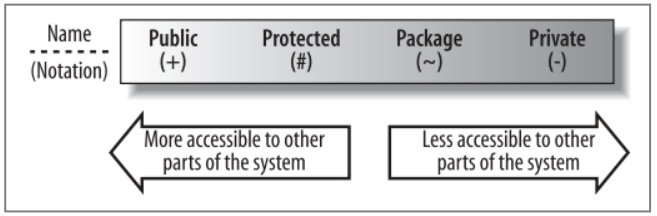
Encapsulamento

Em arquitectura de software, *encapsulamento* refere-se à capacidade de ocultar a complexidade interna das partes de um sistema, expondo apenas aspectos específicos para o exterior

- Isolamento dos detalhes internos das partes de um sistema em relação ao exterior
  - Para reduzir dependências (interacções)
  - Acesso controlado a informação interna das partes através de mecanismos específicos
    - **Visibilidade**
    - **Interfaces**
- Reduz o acoplamento com outras partes
- Possibilita a alteração interna das partes sem afetar o restante sistema
- Contribui para gerir a complexidade e reduzir o esforço de desenvolvimento e manutenção

Meios de Encapsulamento

- **Ocultação de informação** ("*Information Hiding*")
  - Encapsulamento baseado na ocultação da representação interna das partes em relação ao exterior
    - Por exemplo, os detalhes de implementação de uma classe não são expostos para o exterior, apenas a interface pública é exposta
- **Controlo de acesso**
  - Possibilita um controlo selectivo no acesso aos atributos e métodos de uma classe
    - Por exemplo, certos dados e métodos podem ser tornados privados, e apenas acessíveis dentro da classe, enquanto outros podem ser tornados públicos, e acessíveis a partir do exterior da classe
  - **Mecanismos de controlo de acesso**:
    - **Visibilidade**
      - Define a acessibilidade a atributos ou métodos de uma classe
    - **Interfaces**
      - Contractos funcionais que definem a funcionalidade exposta para o exterior



[Miles & Hamilton, 2006]

[Eriksson et al., 2004]

Factorização

O conceito de *factorização* refere-se à decomposição das partes de um sistema de modo a *eliminar redundância* (partes repetidas), relaciona-se com o conceito matemático correspondente de decomposição de uma expressão em *factores* (partes de um produto), por exemplo, $ax + bx = (a + b)x$

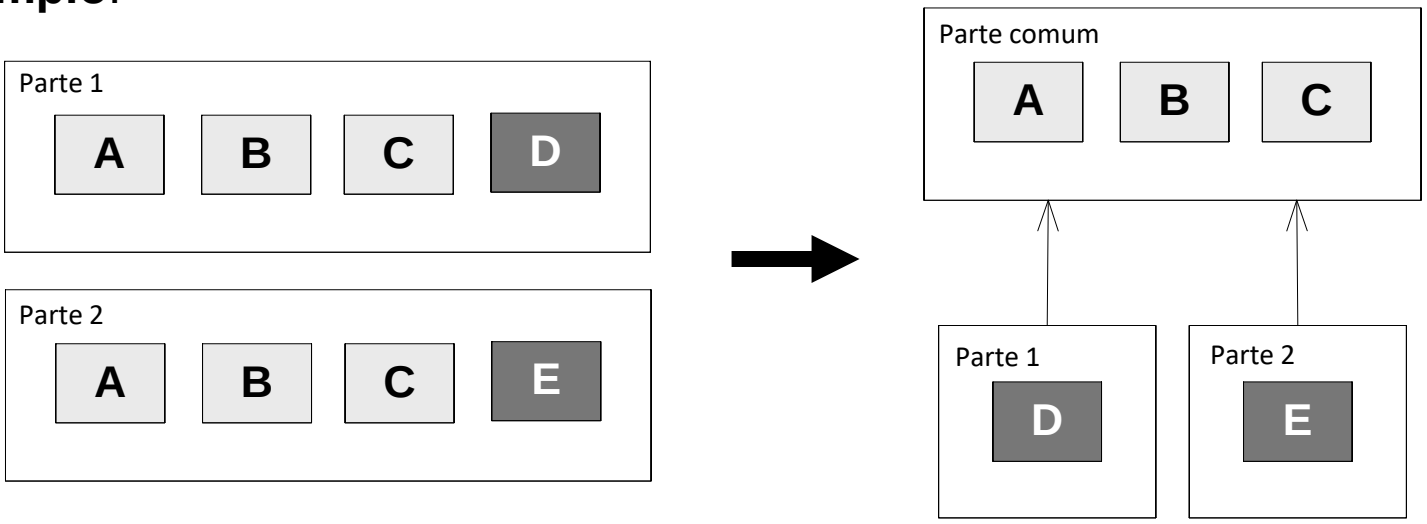
REDUNDÂNCIA

Existência de partes repetidas num sistema, é uma das principais causas de anomalias e de complexidade desorganizada no desenvolvimento de software

Redução de redundância por factorização

As partes repetidas são eliminadas, as partes mantidas são partilhadas

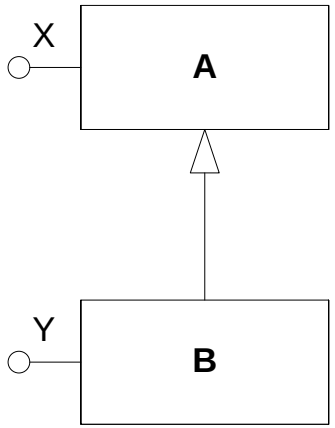
Exemplo:



Mecanismos de Factorização

HERANÇA

- Factorização estrutural
- B **é** A
- Nível de **acoplamento alto**
- Ênfase na estrutura

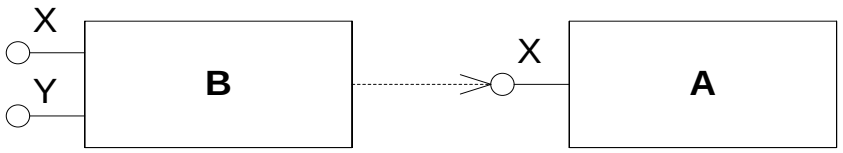


B **é** A

B disponibiliza interfaces X e Y herdando X de A

DELEGAÇÃO

- Factorização funcional
- B **utiliza** A
- Nível de **acoplamento baixo**
- Acoplamento pode variar **dinamicamente**
- Ênfase na funcionalidade



B **utiliza** A

B disponibiliza interfaces X e Y utilizando A para delegar a funcionalidade de X

Padrões de Arquitectura

- Os *padrões de arquitectura de software* são soluções gerais, reutilizáveis, para problemas comuns no desenvolvimento de software
- São um exemplo de *heurísticas* de desenvolvimento
  - Meios de tornar o desenvolvimento mais expedito, neste caso com base em soluções conhecidas (padrões) e boas práticas que orientam o processo de desenvolvimento de um sistema
- Têm a forma de descrições ou modelos de como resolver um determinado tipo de problema, que podem ser utilizados em diferentes situações
- Consolidam boas práticas que foram organizadas e formalizadas para resolver problemas comuns ao desenvolver um sistema
- Permitem acelerar o processo de desenvolvimento ao proporcionar soluções já anteriormente testadas
- Permitem identificar por antecipação questões não imediatamente aparentes do problema a resolver
- No entanto, também podem criar problemas se não forem adequadamente utilizados

Qualidade da Arquitectura de Software

A arquitetura de software deve contribuir para a qualidade do processo de desenvolvimento e do produto final, nomeadamente, garantindo boas métricas de arquitectura e apresentando, entre outras, as seguintes características:

- **Modularidade**
  - Refere-se ao grau em que um sistema é composto por componentes separados que podem ser ligados e combinados de diferentes formas
  - Um sistema modular é mais fácil de desenvolver, testar e manter
- **Testabilidade**
  - Refere-se à facilidade com que um sistema pode ser testado para garantir os seus requisitos
  - Um sistema que é fácil de testar tem maior probabilidade de garantir os requisitos
- **Manutenção**
  - Refere-se à facilidade com que um sistema pode ser modificado ou atualizado
  - Um sistema passível de manutenção é mais fácil de corrigir, atualizar e melhorar
- **Reutilização**
  - Refere-se ao grau em que os componentes de um sistema podem ser reutilizados
  - Componentes reutilizáveis contribuem para a redução do esforço de desenvolvimento e para a melhoria da qualidade de um sistema
- **Flexibilidade**
  - Refere-se à capacidade de um sistema se adaptar a requisitos ou ambientes em mudança
  - Um sistema flexível pode ser modificado ou alargado sem exigir grandes alterações à sua arquitetura

Importância da Arquitectura de Software

A arquitetura de software é um aspeto essencial do desenvolvimento de software, entre outras vantagens:

- Facilita a gestão da complexidade do sistema a desenvolver
- Facilita a compreensão e comunicação de conhecimento acerca do sistema a desenvolver
- Contribui para a redução do esforço de desenvolvimento
- Contribui para garantir a qualidade do sistema a produzir, nomeadamente, que o sistema é concebido para satisfazer os requisitos especificados
- Facilita a gestão da mudança ao longo do processo de desenvolvimento
- Possibilita a identificação de riscos de desenvolvimento e a abordá-los numa fase inicial do processo de desenvolvimento
- Contribui para reduzir custos de desenvolvimento e de manutenção do software

Bibliografia

[Pressman, 2003]

R. Pressman, *Software Engineering: a Practitioner's Approach*, McGraw-Hill, 2003.

[Gamma et al., 1995]

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Shaw & Garlan, 1996]

M. Shaw, D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[Vernon, 2013]

V. Vernon, *Implementing Domain Driven Design*, Addison-Wesley, 2013.

[Parnas, 1972]

D. Parnas, *On the Criteria to Be Used in Decomposing Systems into Modules*, Communications of the ACM 15-12, 1968.

[Kruchten, 1995]

F. Kruchten, *Architectural Blueprints - The "4+1" View Model of Software Architecture*, IEEE Software, 12-6, 1995.

[Schach, 2010]

S. Schach, *Object-Oriented and Classical Software Engineering*, 8th Edition, McGraw-Hill, 2010.

[Booch, 2004]

G. Booch, *Software Architecture*, IBM, 2004.

[Korf, 1980]

R. Korf, Toward a model of representation changes, Artificial Intelligence, Volume 14, Issue 1, 1980.

[Laplante, 2007]

P. Laplante, What Every Engineer Should Know About Software Engineering, CRC Press, 2007.