

14.04.15

verificação vs validação

(ex) um cientista quer um meio de transporte não poluente → constrói-se uma bicicleta

verificação: a implementação está bem feita, ou seja, os roteadores estão bem postos, a bicicleta trava quando é preciso, ...

validação: não fizemos levantamento de requisitos suficiente e a final o cientista quer um meio de transporte que seja sustentável

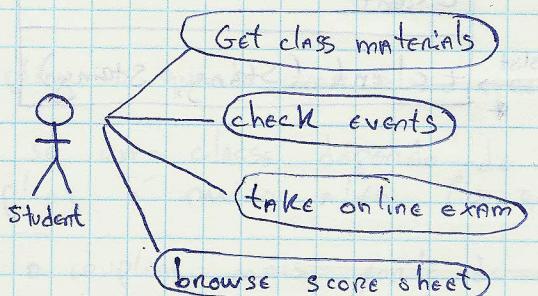
21.04.15

Requisitos

funcionais	{	}	não funcionais (qualidades, atributos)
------------	---	---	---

RESUMO

diagrama de casos de utilização (DCaU)

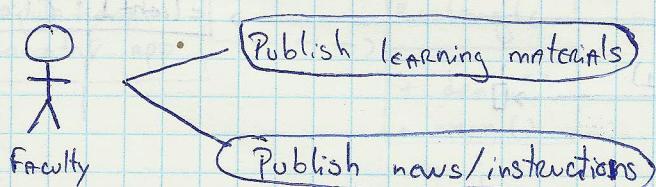


Caso de utilização: sequência de ações que um sistema executa e que produzem resultado c/ valor p/ algum ator em particular

implica na análise de requisitos:

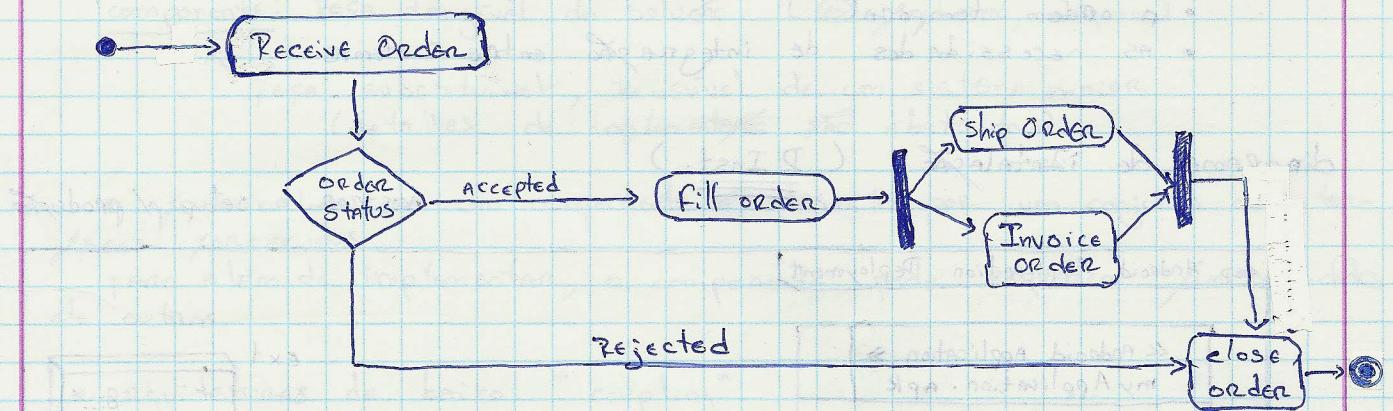
- foco no user e nos episódios de uso
- foco na compreensão do que os atores consideram de valor

p/ cada ator, identificar os objetivos/motivações p/ usar o sistema



modelo de CaUs: serve como vista de um sistema que destaca o comportamento observável, tal como é percecionado pelos utilizadores — o modelo capta os requisitos do sistema ("o que"), não a implementação da solução ("o como")

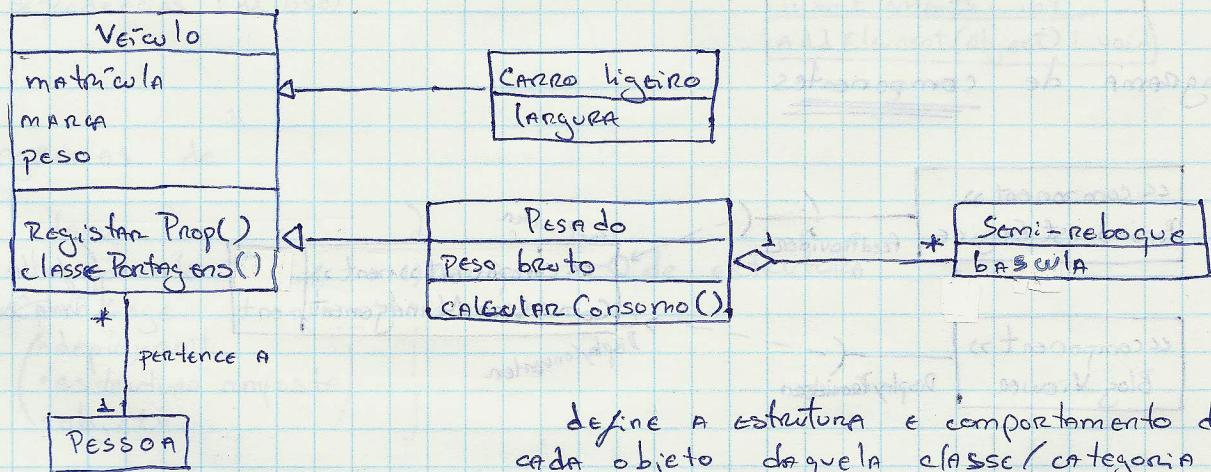
Diagrama de atividades (DATIV)



Quando aplicar?

- modelar fluxos de trabalho / processos de negócio
- desenvolver um algoritmo complexo
- descrever a sequência de interações entre atores e o sistema sob especificação, num caso de utilização
- descrever processos organizacionais existentes/novos
 - neutro em relação à programação
 - bom a captar papéis
 - pode captar fluxo de dados

Diagramas de classes (DC)



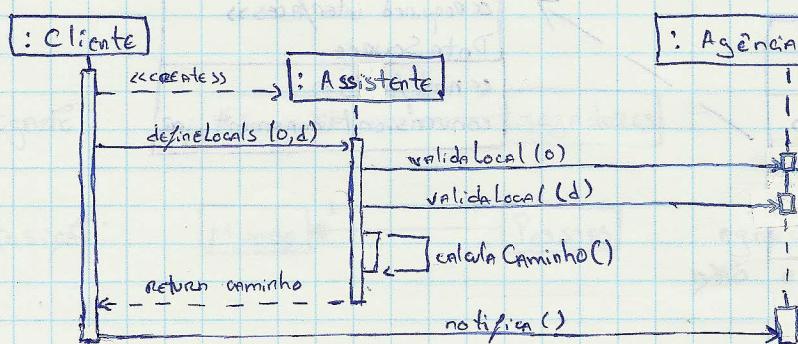
define a estrutura e comportamento de cada objeto daquela classe / categoria

funciona como um molde p/ criar objetos

um objeto tem h/ um estado (interno) — atributos

— comportamento (funcionalidades) — funções/métodos

Diagramas de sequência (DSeq)



Ao desenhar um diagrama de interface, estamos a tabular responsabilidades

diagramas de interface:

- diagramas de sequência (formato alinhado)
- diagrama de comunicação (formato gráfico)

- mostram, p/ um cenário de um CnU:
- os eventos que os atores externos geram
 - a ordem temporal
 - as necessidades de integração entre sistemas

diagrama de instalação (D Inst.)

mostrar o setup p/ produção

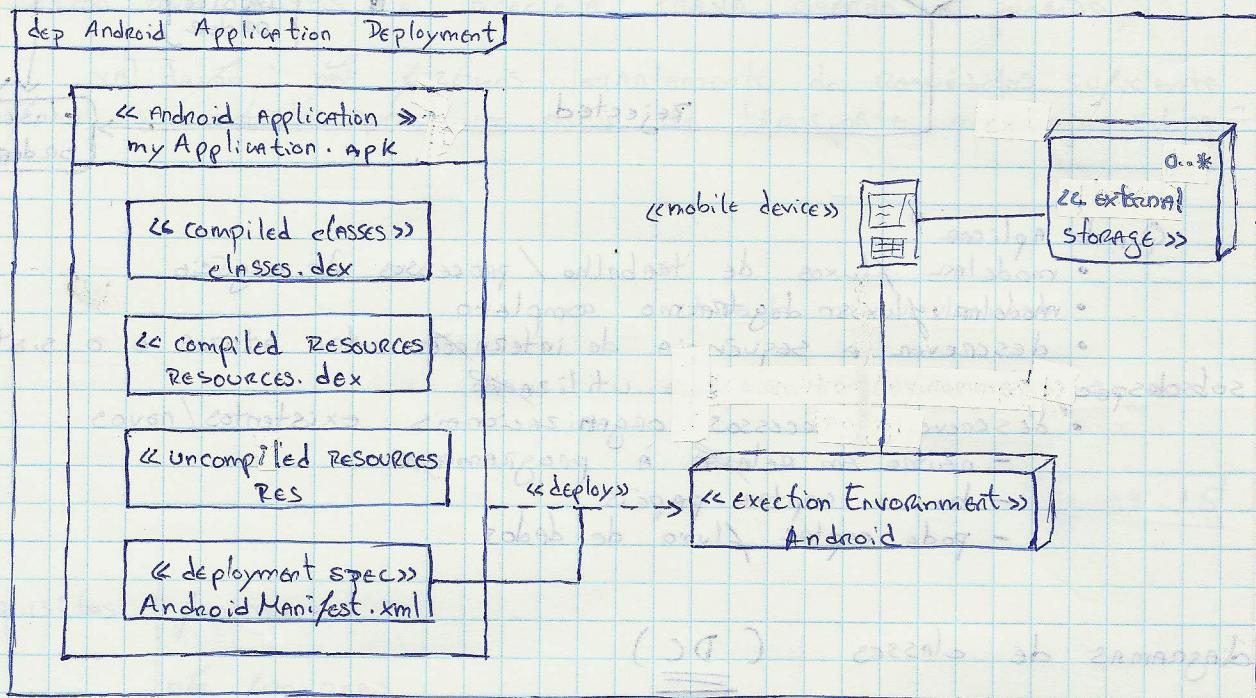
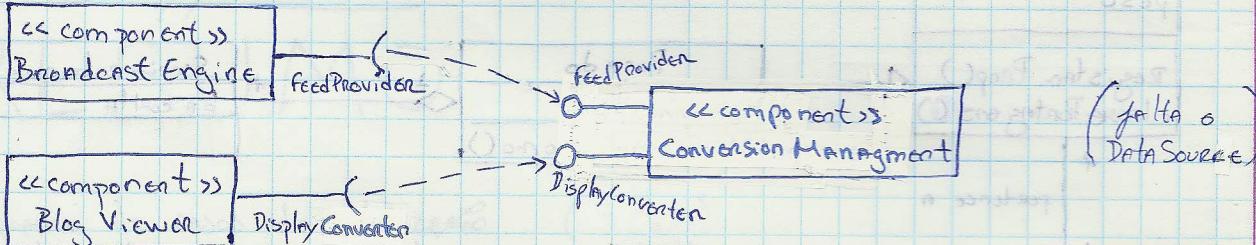
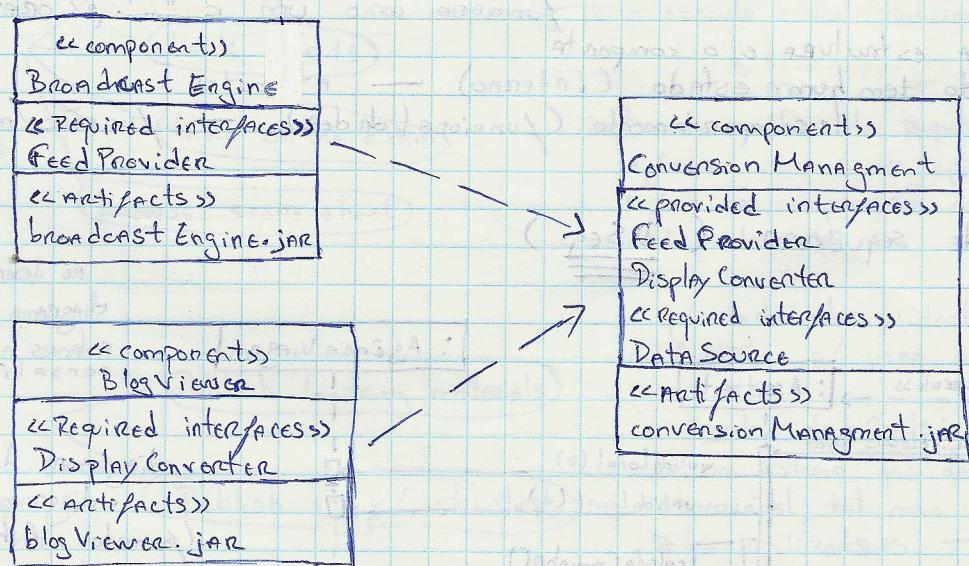


diagrama de componentes



notações alternativas:



divide sistemas complexos em subsistemas mais geríveis

componente: pega tangível da solução (⇒ ficheiro, arquivo)

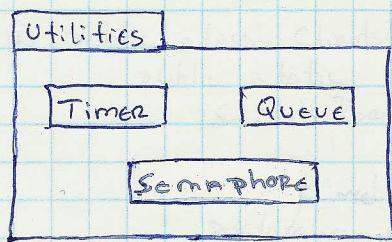
pega substituível, reusável de um sistema maior
(detalhes de implementação são abstraidos)

A funcionalidade de um componente é descrita por um conjunto de interfaces fornecidas

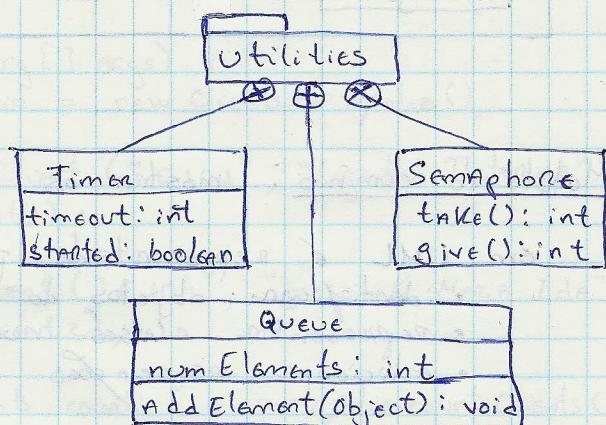
para além de implementar, o componente pode requerer funcionalidades de outros

Arquiteturas de baixo "coupling"

diagrama de pacotes



OU



Exemplifica blocos e dependências (de uso).

Diagramas de

- classe
- componente
- package
- deployment
- estrutura composta
- objeto

} diagramas de estrutura

- Atividade
- caso de uso
- interação
 - sequência
 - comunicação
 - temporal
 - visão geral
 - (- máquina de estados)

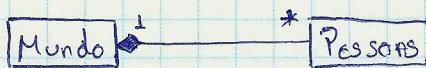
} diagramas de comportamento

Agregação:



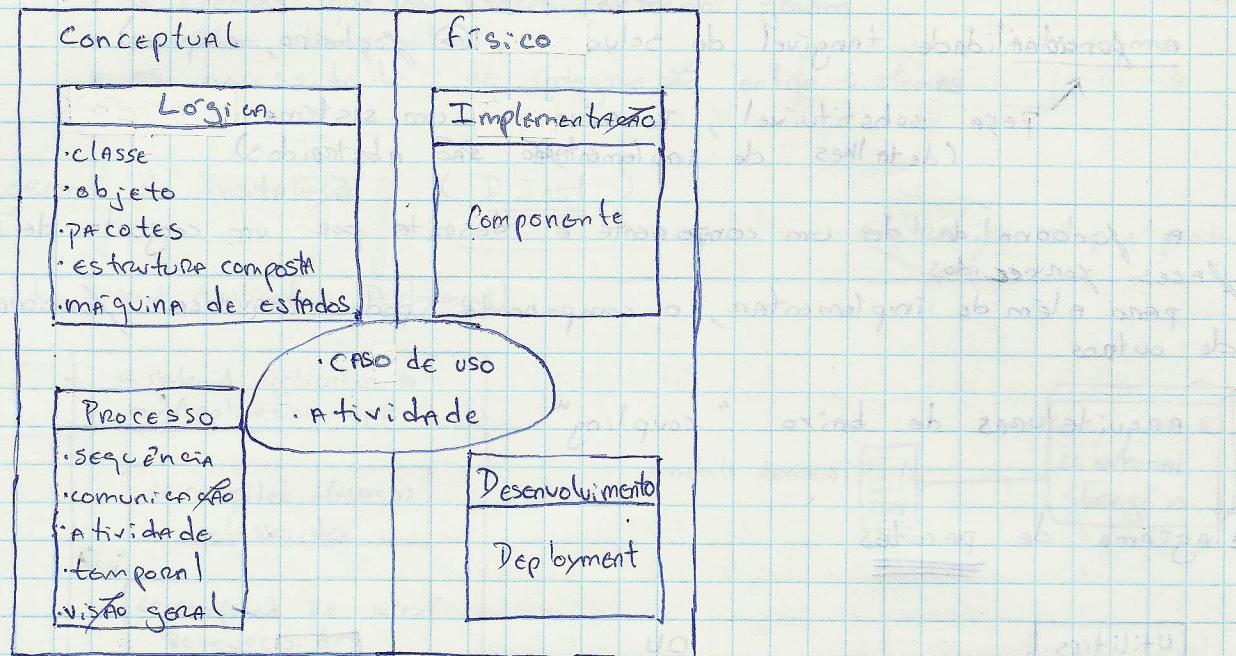
agregação "fraca", pois as partes existem sem o todo

composição:



agregação "forte", pois as partes não existem sem o todo

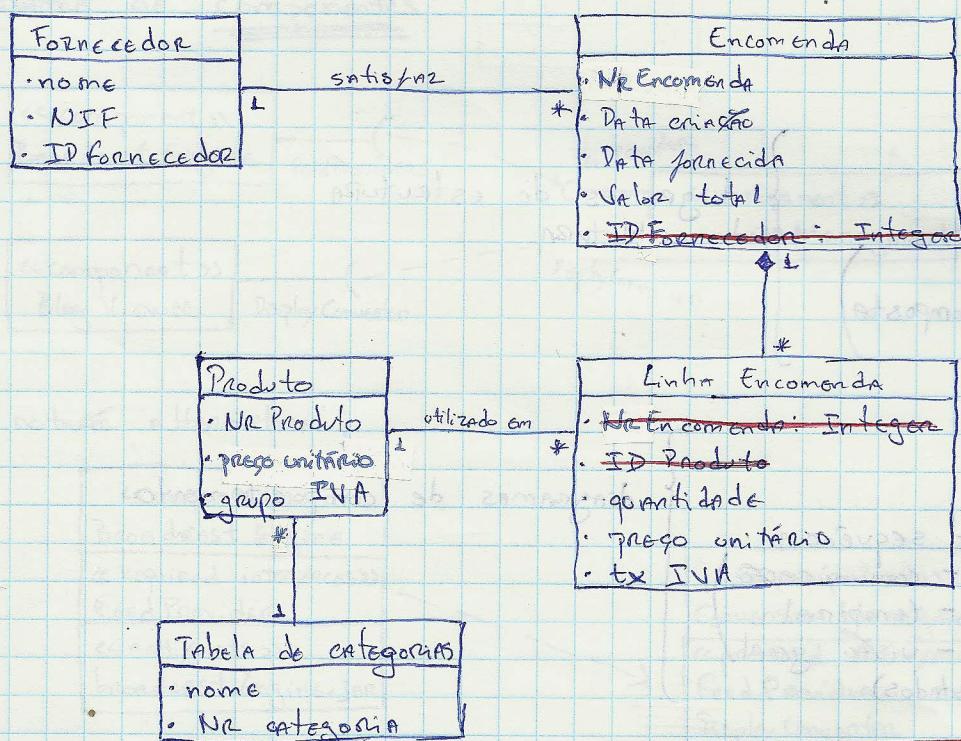
Modelo "4+1" (vistas)



Modelo do domínio: mostra os conceitos de um problema

Em UML é representado ~~e/~~ por diagramas de classes

- identificar objetos do problema
- representar classes num DC
- adicionar associações e atributos



modelo contagiado
por raciocínio do
modelado de BD

Requisitos funcionais

- captam o comportamento pretendido do sistema (serviços, funções ou trajetas que o sistema deve realizar)
- podem ser captados nos CAs
- são descritos c/ diagramas de comportamento: deactividades, sequência

Requisitos não funcionais

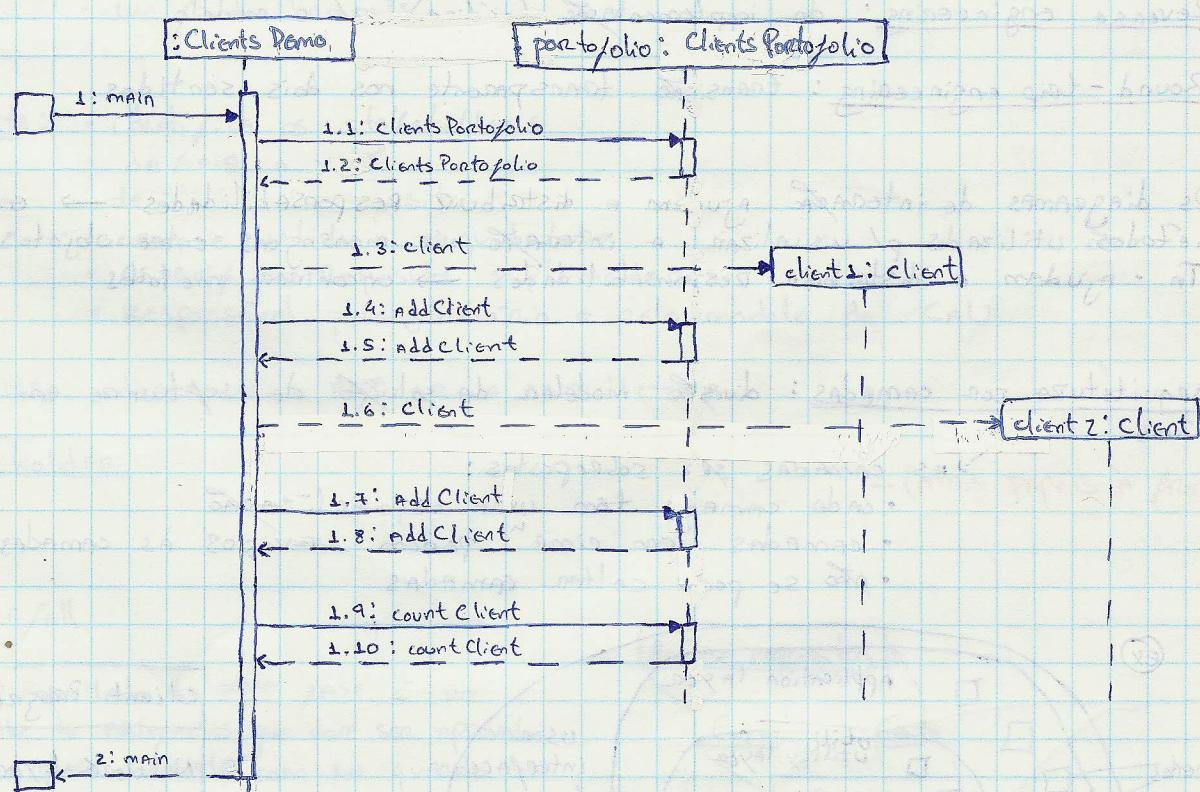
- restrições globais num sistema de software (ex: robustez, portabilidade...)
- por regra, não afetam apenas um módulo/CAU
- também designados como atributos de qualidade

do código p/ o diagrama

```

① public class ClientsDemo {
    public static void main (String [] args) {
        Clients Portfolio portfolio = new ClientsPortfolio();
        Client client1 = new Client ("C101", "Logistica Tartaruga");
        portfolio.addClient (client1);
        Client client2 = new Client ("C104", "Jose & Maria Lda");
        portfolio.addClient (client2);
        System.out.println ("Clients count: " + portfolio.countClients ());
    }
}

```



```

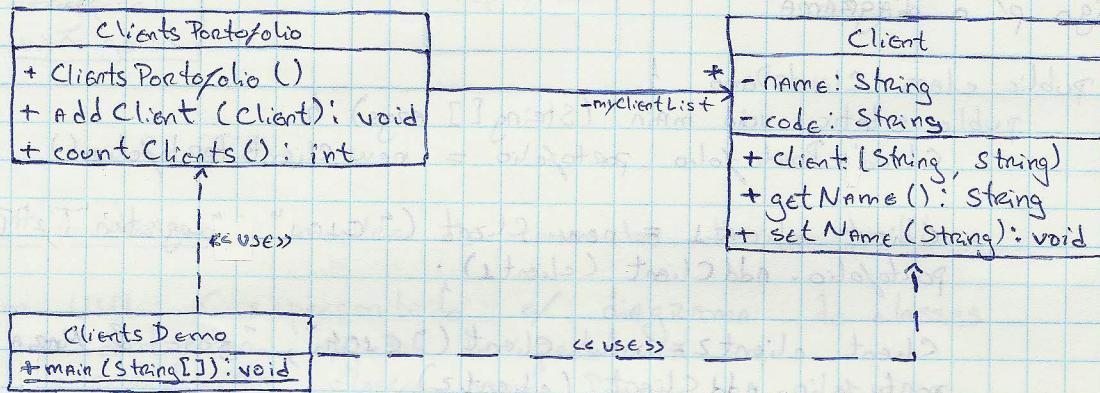
② public class ClientsPortfolio {
    private ArrayList<Client> myClientsList;
}

public ClientsPortfolio() {
    myClientsList = new ArrayList<>();
}

public void addClient (Client newClient) {
    this.myClientsList.add (newClient);
}

public int countClients () {
    return this.myClientsList.size ();
}

```



Forward engineering: do modelo p/ a implementação (código)

Reverse engineering: da implementação (código) p/ o modelo

Round-trip engineering: transição transparente nos dois sentidos

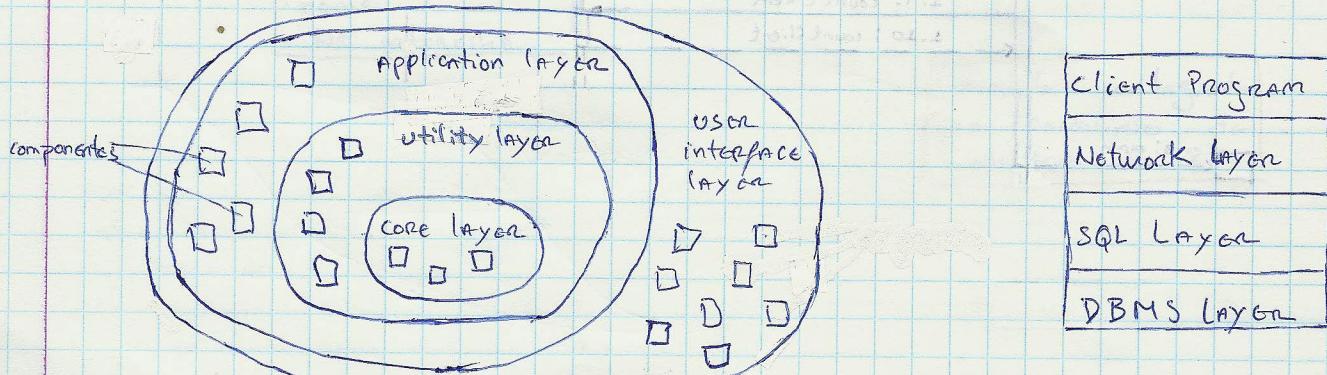
Os diagramas de interação

- são utilizados p/ visualizar a interação via mensagens entre objetos
- ajudam a distribuir responsabilidades → encontrar métodos

Arquitetura por camadas: divisão modelar da solução de software em camadas/níveis

→ As camadas são sobrepostas:

- cada camada tem uma especialização
- camadas "em cima" pedem serviços às camadas "de baixo"
- não se pode coltar camadas



Anarquitectura lógica: organização geral da solução em blocos (packages)

- ↳ os packages podem representar agrupamentos muito diferentes
- (Ex) • packages num programa em JAVA
 • " " " " no modelo UML
 • subsistemas / divisões do sistema sob especificação
 (diagramas de pacotes)

Anarquitectura física: explicita a configuração concreta do sistema no ambiente de produção pretendido

Responde às questões:

- que servidores são necessários?
- onde se instalar cada módulo?

Processo de eng. de software: conjunto de atividades e resultados associados que produzem uma peça de software

modelo de processo de ES: descrição abstrata de um processo da ES segundo uma interpretação

propto:

- atividades a realizar
- produtos " "
- papéis na equipa

- (Ex)
- modelo em cascata (Waterfall)
 - " " espiral (Spiral)
 - Extreme Programming
 - Unified Process

Analista:

- identifica os stakeholders
- definir a visão
- detalhar os requisitos
- cria e valida a arquitetura
- especifica Requisitos suplementares
- responsável pelo glossário e pelo modelo de CAU

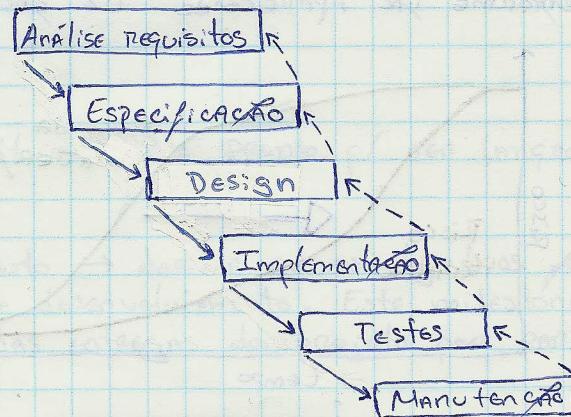
Arquiteto: descrever e refinar a arquitetura.

stakeholder

- (mais papéis à frente)

Waterfall

- o resultado de cada fase é um conjunto de artefactos que deve ser aprovado
- a fase seguinte começa quando a anterior termina
- em caso de erros, é necessário repetir passos anteriores
- encaixa na lógica de outros processos de eng. A. (Ex HW)



- vantagens:
- abordagem simples e disciplinada
 - milestones são fáceis de marcar (no fim de cada fase)
 - progresso linear (de um passo p/ o outro)
 - sabe-se o que se segue
 - pode ajudar a transferência de conhecimentos eficiente (documentação, objetivos claros)

- problemas:
- confirmação tardia de que os riscos estão controlados
 - integração e testes tardios
 - "black out" ostensivo
 - pressupõe-se que os requisitos são estáveis
 - custo de mudança aumenta quase exponencialmente c/ o tempo

Desenvolvimento Ágil

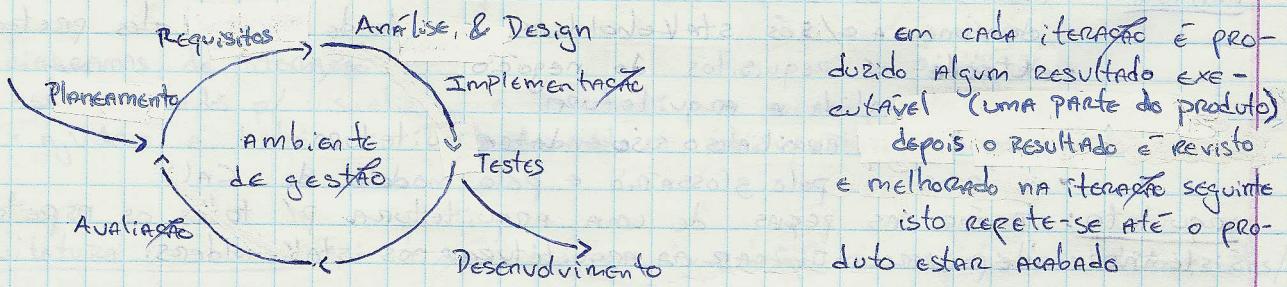
objetivo: desenvolver software rapidamente p/ responder às alterações rápidas dos requisitos

é necessário:

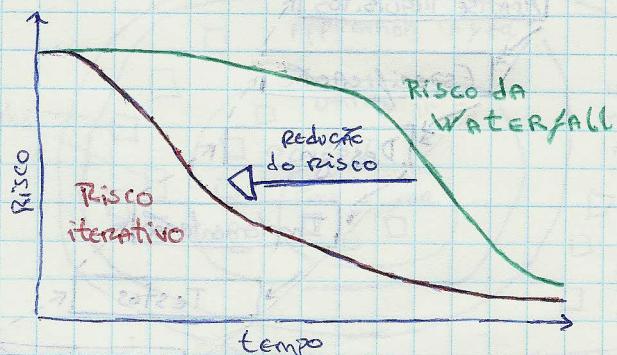
- práticas que equilibrem a disciplina e o feedback
 - ciclos curtos e entrega de valor frequente
 - integração em contínuo
 - desenvolvimento orientado por testes
- aplicar princípios de design que favorecem a construção de software flexível e evolutivo

- abordagens:
- incremental
 - iterativa
 - incremental e iterativa

Abordagem iterativa → foca a entrega de valor orientada por ciclos curtos



entregas frequentes e integração de forma contínua reduzem significativamente o risco do projeto, pois os erros são encontrados e tratados conforme vão aparecendo (e não tudo no fim)



Fases do Open UP

• conceção (inception)

[sem requisitos detalhados]

objetivo: conseguir concordância entre todos os stakeholders sobre os objetivos do projeto e se é p/ avançar ou não.

milestone de objetivos → neste ponto examina-se custos vs. benefícios do projeto e decide-se se é p/ avançar ou cancelar o projeto

• elaboração

[sem design detalhado]

objetivo: atenuar riscos técnicos e não-técnicos

↳ riscos técnicos são habitualmente tratados por estabelecer a base de uma arquitetura executável do sistema e proporcionar uma base estável p/ a maioria do esforço de desenvolvimento da fase seguinte

milestone da arquitetura → é o combinado a base de requisitos. Examinando o detalhe dos objetivos e o âmbito do projeto, a escolha da arquitetura e a resolução dos maiores riscos

↳ este milestone só estará concluído quando a arquitetura for validada

concorda-se c/ a arquitetura executável p/ ser usada no desenvolvimento da aplicação e considera-se que o valor conseguido até este momento e os riscos são aceitáveis?

• construção

objetivo: desenvolver de forma rentável uma versão operacional do sistema que possa ser implantada na comunidade de utilizadores

milestone da capacidade operacional inicial → a este ponto, o produto está pronto p/ ser entregue à equipa de transição. Todas as funcionalidades foram desenvolvidas e os testes iniciais foram completados. Para além do software, foi desenvolvido um manual do utilizador e existe uma descrição da versão atual. O produto está pronto p/ testes como versão beta.

considera-se que a aplicação já está suficientemente perto de ser lançada e que se devia mudar o foco principal p/ a equipa que vai equipar, "polir" e assegurar-se do desenvolvimento e sucesso?

• transição

objetivo: certificar que o software está pronto p/ ser lançado p/ os utilizadores

milestone do lançamento do produto → decidir se os objetivos foram atingidos e se é necessário outro ciclo de desenvolvimento. Este milestone é resultado das reviews e aceitação das entregas do projeto por parte do cliente.

A aplicação está pronta p/ ser lançada?

- gerente do projeto: • planejar o projeto
• gerir e planejar a iteração
• responsável pela lista de itens de trabalho e pela lista de riscos
• avaliar resultados

- (mais papéis aqui)

- "developer": • desenhar e implementar solução
• desenvolver um "plano B"
• integrar e criar a construção

- "tester": • criar casos de teste
• implementar e executar testes

OpenUP é um exemplo de um método ágil, pq

é um processo iterativo c/ as iterações distribuídas por 4 fases: conceção, elaboração, construção e transição.

cada fase pode ter quantas iterações forem necessárias

as iterações podem ter duração variável, dependendo das características do projeto

desenvolvimento orientado por casos de utilização

os casos de utilização são usados p/:

- identificar o que é p/ fazer
- planear o desenvolvimento (os casos de utilização são priorizados e os prioritários serão os primeiros a serem construídos)
- dar um cenário/contexto p/ guiar o programador
- dar cenários p/ os planos de teste

os CAs são um resultado central, que liga as várias atividades do desenvolvimento

desenvolvimento incremental → o produto é projetado, implementado e testado de forma incremental (um pouco mais é adicionado de cada vez) até que o produto esteja concluído. Isto envolve desenvolvimento e manutenção. Considera-se que o produto está terminado quando satisfaz todos os requisitos.

entrega de valor frequente → uma parte essencial dos métodos iterativos é o contacto frequente c/ o cliente e a demonstração dos incrementos que têm relevância p/ o cliente. Por isso mesmo, o ritmo de entrega/demonstração de incrementos (que geram valor p/ o negócio) é mais intenso que nos métodos sequenciais.