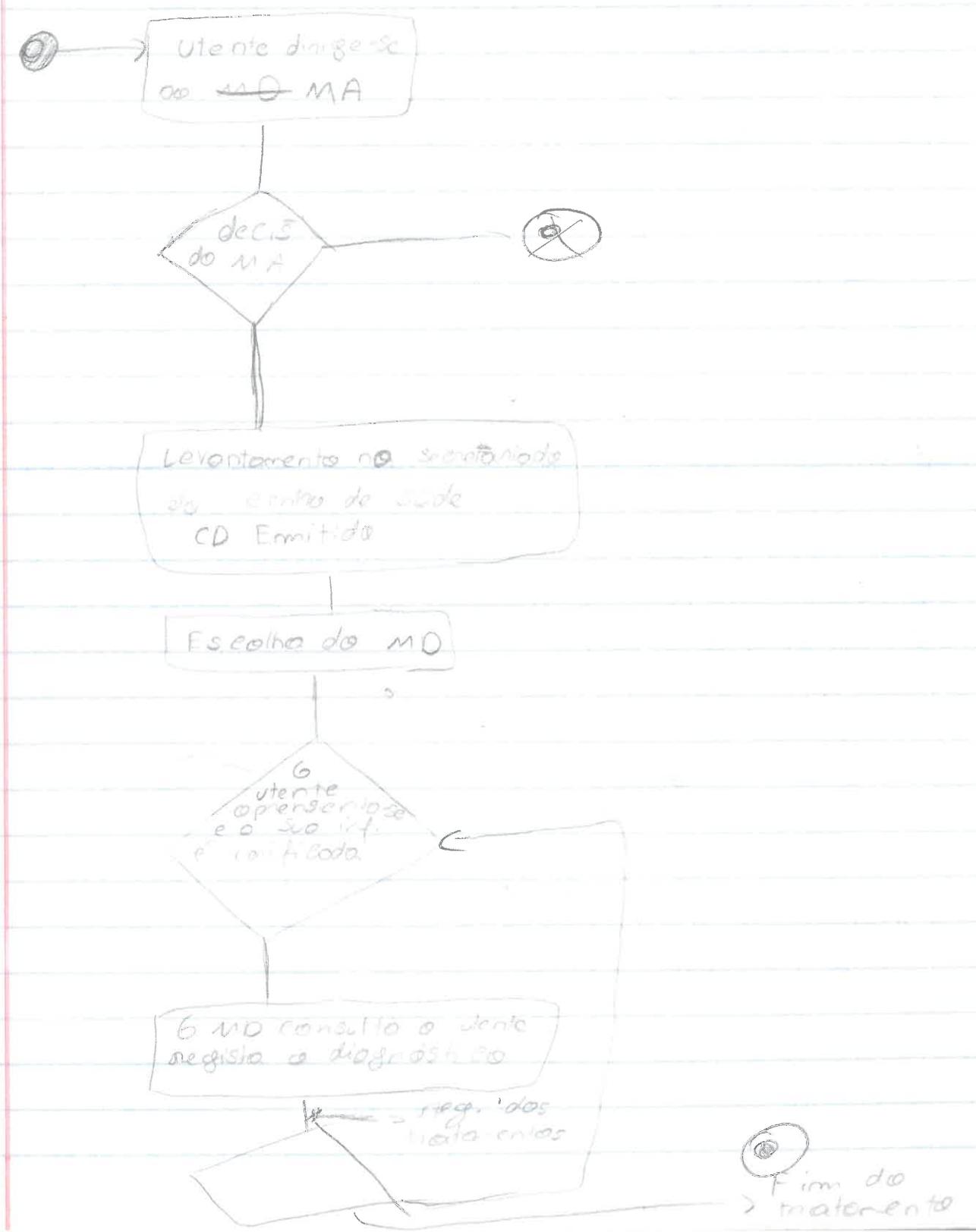
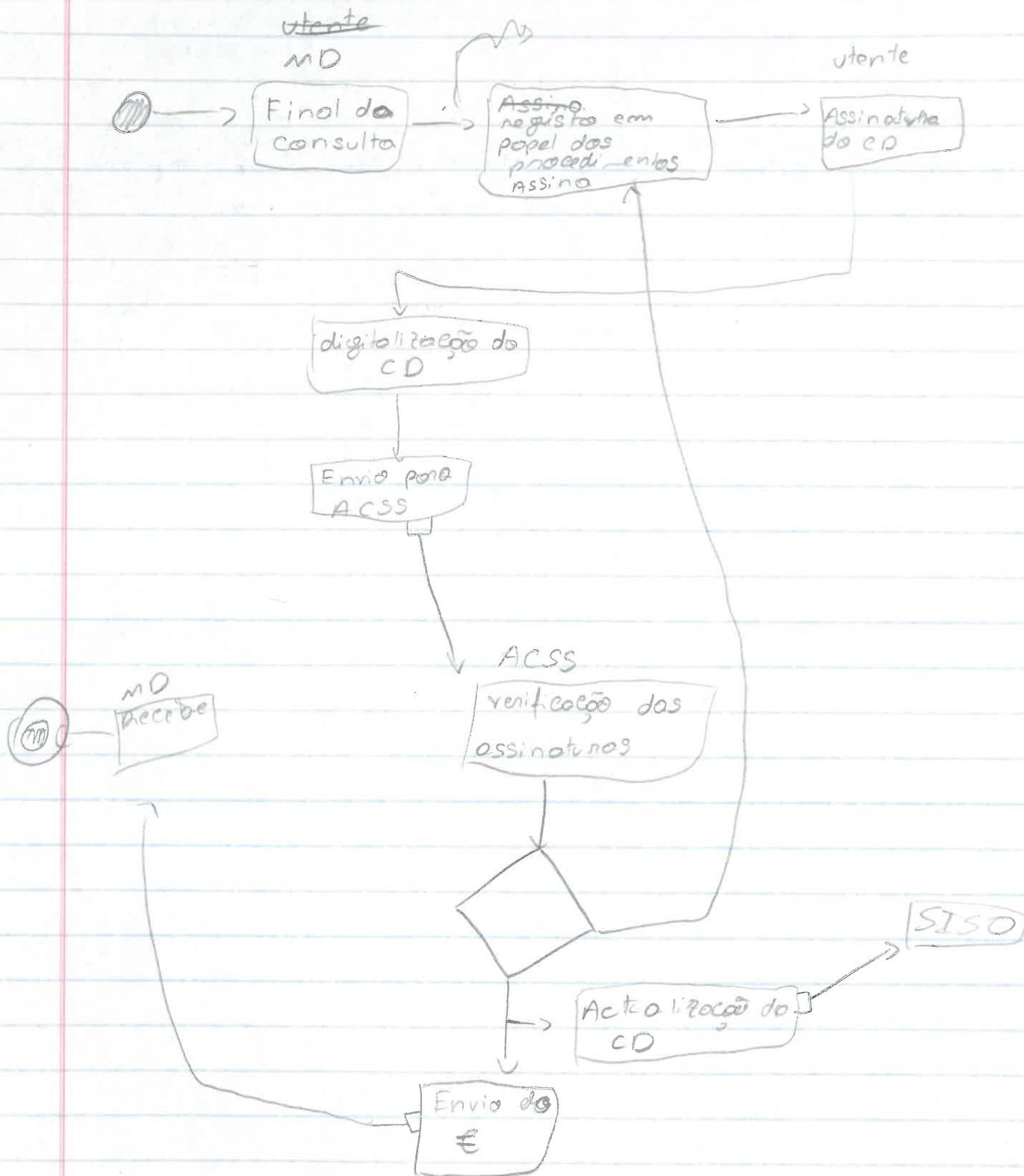


Modelação e Análise de Sistemas

Um diagrama de atividade mostra a sequência de processos que realizam certos trabalhos, é útil para simplificar e expor de forma objetiva problemas de elevada complexidade.





No universo do discurso do mundo académico, os objetos de discussão são:

- mail

- nº MEC

- nome

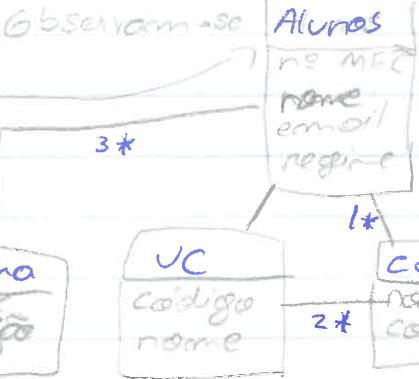
- Região

- Turma

- Cadeira

Criando um mapa de Informações:

Observam-se



Correção de interacção para o mundo académico

Algumas categorias são tangíveis (reais), outras são conceitos
As reais categorias têm relações entre elas

1* frequenta

2* disponível em

3* contém

No universo de discurso possuímos objetos que se interagem através de funções / métodos, os objetos não sóbem estando interna de outros objetos.

Classes são dotadas de um nome, atributos e métodos

Um objeto tipo carro pode ter o

Atributos:

- cor
- marca

Também temos a necessidade de manter relações entre objetos.

Operações:

- Acelerar

- freiar

Multiplicidade

um objeto Aluno, apenas está associado a um curso, a multiplicidade é de N:1, pois um curso tem vários alunos.

slide 38



No classe Gorden, o customer ID, FirstName, dizem respeito ao cliente e deviam pertencer a um classe cliente.

Generalização

- uma classe mais genérica pode ter relações de generalização

Ex:



Conceito ou Atributo?

Usando o exemplo anterior, a classe Aluno

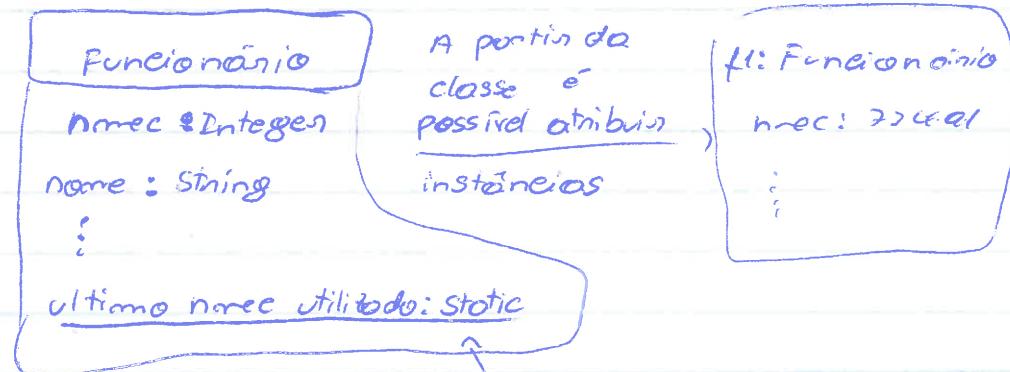


não é algo simples, um campo

que se passa preencher com uma linha,
assim devia ser uma classe

classe é uma categoria → nome singular

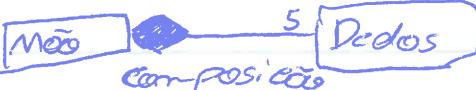
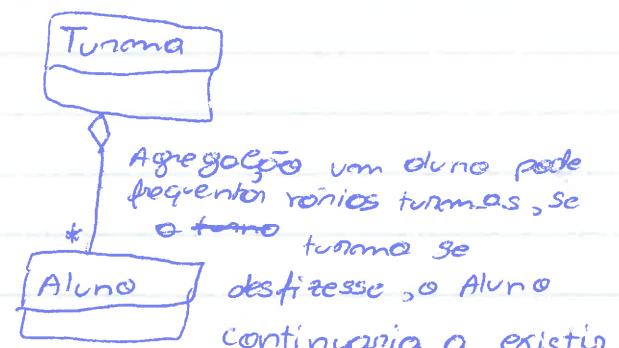
É possível especificar os atributos de uma classe



este atributo não é atribuído

• a uma instância fica apenas definido na classe.

Agregação vs Composição



Não existem um sem
6 dedos não existem de
existir sem mão

Classe
Nome: Livros

Atributos: Título (str)
Língua: Eng
Editor: Addison-Wesley
Ano: 2004
Autor: Fowler, Martin, Rosenberg, Doug

Instâncias
Objetos

UML Detailed

vis. Obj

b)

Descrição do Livro

UML Detailed

Eng

Addision - Wesley

cop. 2004

Fowler, martin

Livros

código de barras

Localização:

c)

classe
não c. Utilizador

Requisito

Atributos: nome

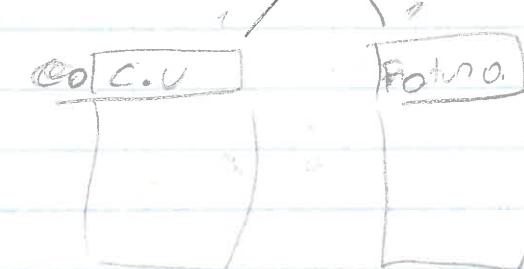
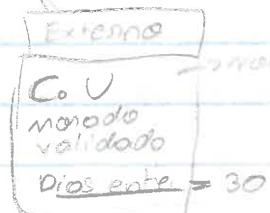
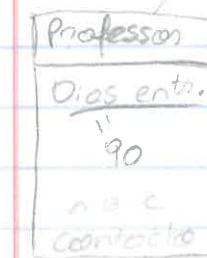
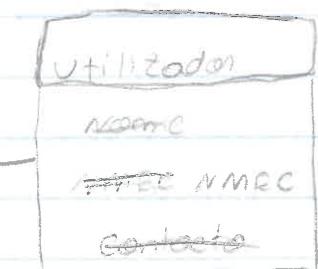
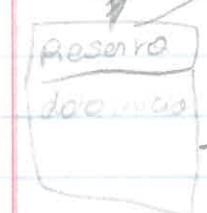
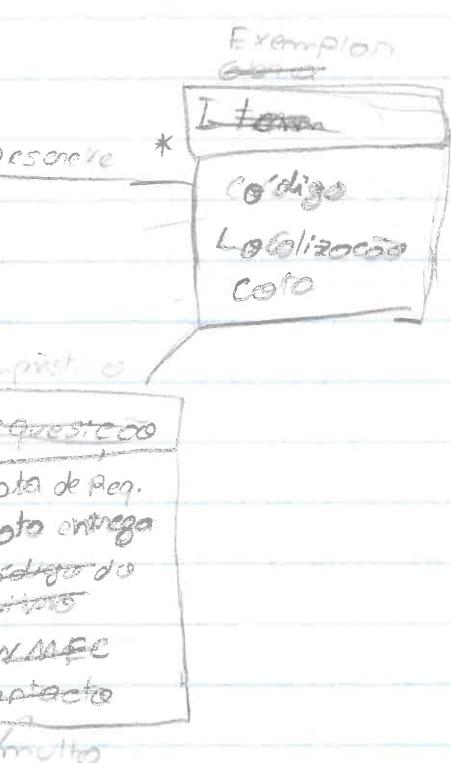
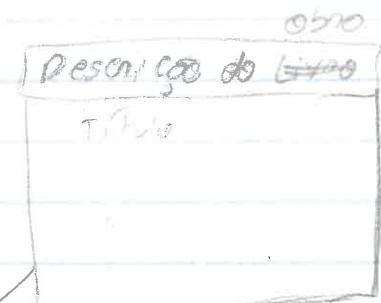
user id

email

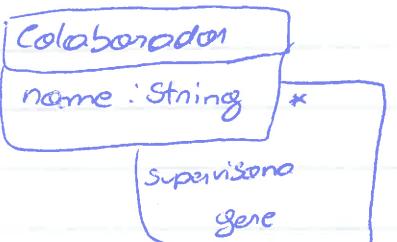
Entregar Livro()

Requisitar()

PesquisarLivro()



Um objeto da de uma classe pode estar relacionado com um objeto da mesma classe



Um analista também pode fazer notas se achar importante

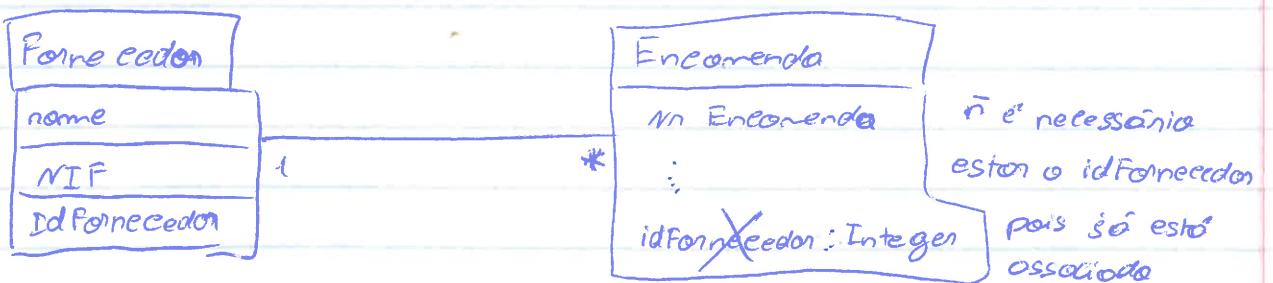
Classe de Associação

A função dessas classes é descrever uma Associação

Classe abstrata

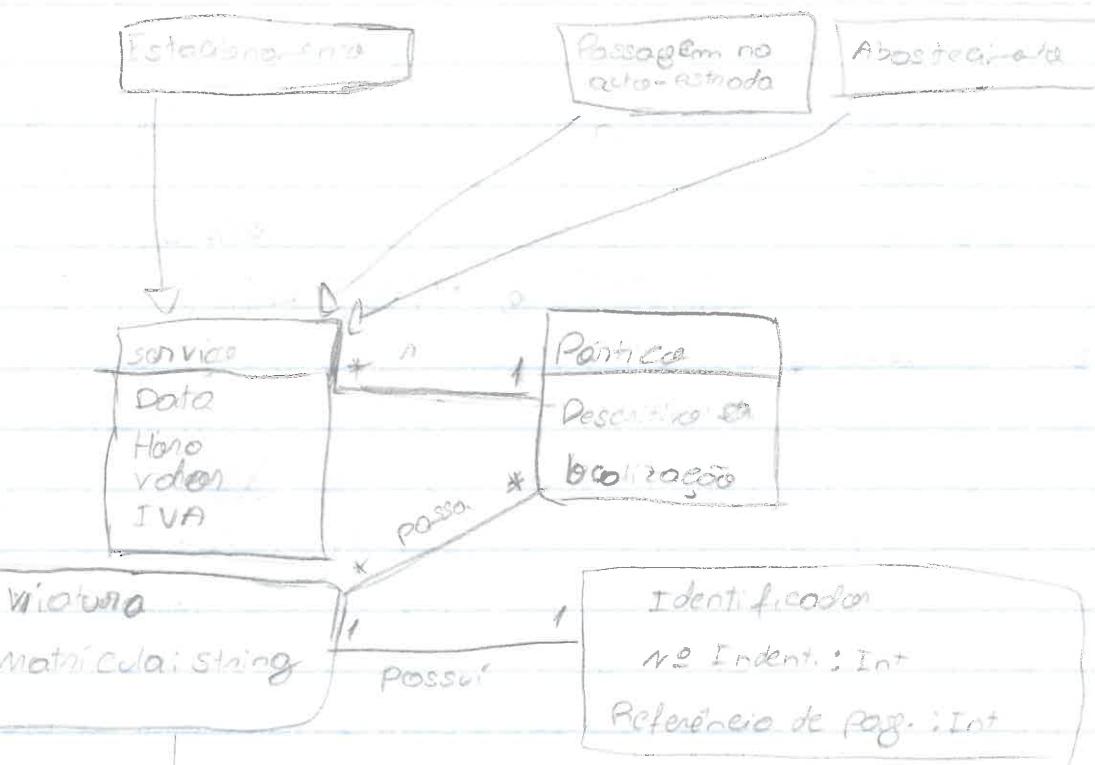
Uma classe que não pode ser instanciada diretamente, os seu métodos são abstratos e só implementados naquela classe. Usam-se quando há a necessidade de criar uma classe genérica onde ainda não se tem dados suficientes para instanciar.

Este tipo de Modelação não deve ser confundido como de Base de Dados



Aula 3 parte 2

c)



Especificação de requisitos

É importante perceber os requisitos do sistema e a linguagem usada no contexto.

Um analista deve fazer uma lista de requisitos após falar com os stakeholders

no caso de um auricular:

Como é que eu faço ... ?

→ Atender chamada

→ Emparelhar com o telemóvel → Todos estes requisitos

→ Colocar na orelha)

→ Aumentar / diminuir o volume

... .

s vistos do ponto de

utilizador. Este estilo

de linguagem está

orientado para o user

Tarefa acessória mas não corresponde à interacção com o aparelho

Estilo:

- perspectiva do utilizador.
- uma ação para usar o sistema
- Verbo
- caixa fechada → descrição de sistema em que não interessa os componentes de um sistema nos sim as suas funcionalidades.

Os atores interagem com o sistema

Exemplo:

Numa loja há dois papéis que são nos 5 relevantes:

- o vendedor e o cliente

Podemos notar que há um sistema de ação-resposta entre os dois atores e o sistema de vendas

Chamamos ao ~~o~~ ator de interagir com o sistema ator

Caso de utilização

Caso de utilização → tem fluxo tipo ~~existe compre~~ Ex: no ~~compra~~ compra paga-se com dinheiro

mos também podem ocorrer situações alternativas → também se pode pagar em cash

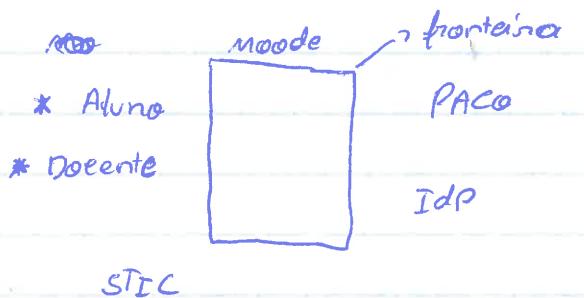
Também pode ocorrer situações de erro

Ex: G cash não é detectado.

Todos estes situações têm de fazer parte na análise de requisitos.

- . A cada caso de utilização está associada uma nomenclatura que descreve o caso, essa nomenclatura segue uma estrutura

→ ver slide 14 dos requisitos



Qual a motivação do aluno usar o móbil?

- Slides Buscar o estídio

- ver as notas

- Entregar trabalhos

- Procurar emails

- Ver fórum

... " " " " " " ?

- notificar os alunos

- adicionar slides

- Pedir a submissão de trabalho

- lançar notas

converm utilizar uma

linguagem # fluente
e que se use no dia-a-dia

1 Trabalho 3 Ponte I

a) Definir Atores:

- Aluno - TIC
- Docente
- PAPE
- IdP

Identificar ecos de utilização

- Aluno
 - Consultar slides
 - Consultar notas
 - Fazer testes
 - Entregar trabalhos
 - Ir ao fórum

- Docente
 - Entregar notas
 - Submeter slides
 - Desconegor os trabalhos dos Alunos
 - Ir ao fórum
 - Notificar Alunos
- TIC
 - manutenção do moodle
 - Alterações do site

Aluno / Docente

Desenvolver ~~só os~~ o ~~do~~ COU

consultas ^{de} notícias 6 Aluno acede à página de estudo
e escolhe o material de estudo
e faz o seu download.

Aluno / docente

consulta da avaliação 6 Aluno não desconhece ou abre o documento que contém a sua avaliação ^{da} respetiva UC

Realizar Prova
de Avaliação

6 Aluno acede à página de teste Prova e faz a sua realização

Submissão de
trabalhos

6 Aluno acede à respetiva UC, ~~com~~
escolhe o ficheiro a enviar e procede com a sua submiss.

Aluno Verifica notificações

6 Aluno acede à página do fórum e informa-se de alguma notificação

Serviço de
manutenção

Falar de quaisquer problemas de funcionalidade relacionada com o moodle, mantendo o site em condições operacionais

Atualizações do sistema. Informar de qualquer necessidade de alterar alterar o sistema para que o moodle possa requerer

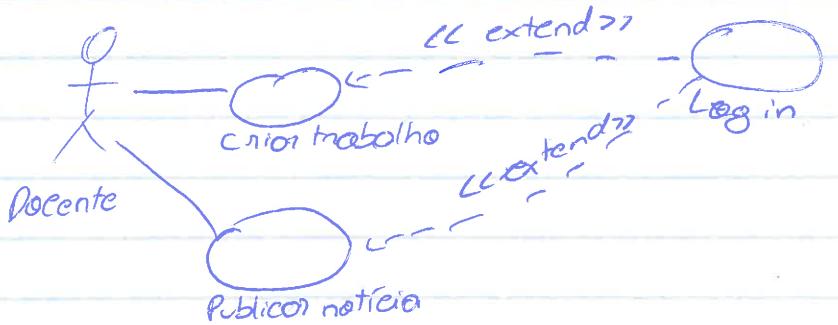
* É necessário fazer a adaptação dos requisitos para um contexto de engenharia, com conceito genérico na linguagem da utilizador tem de ser adaptado para a sua linguagem precisa e consisa.

respeitam enquadramento
regulamentar

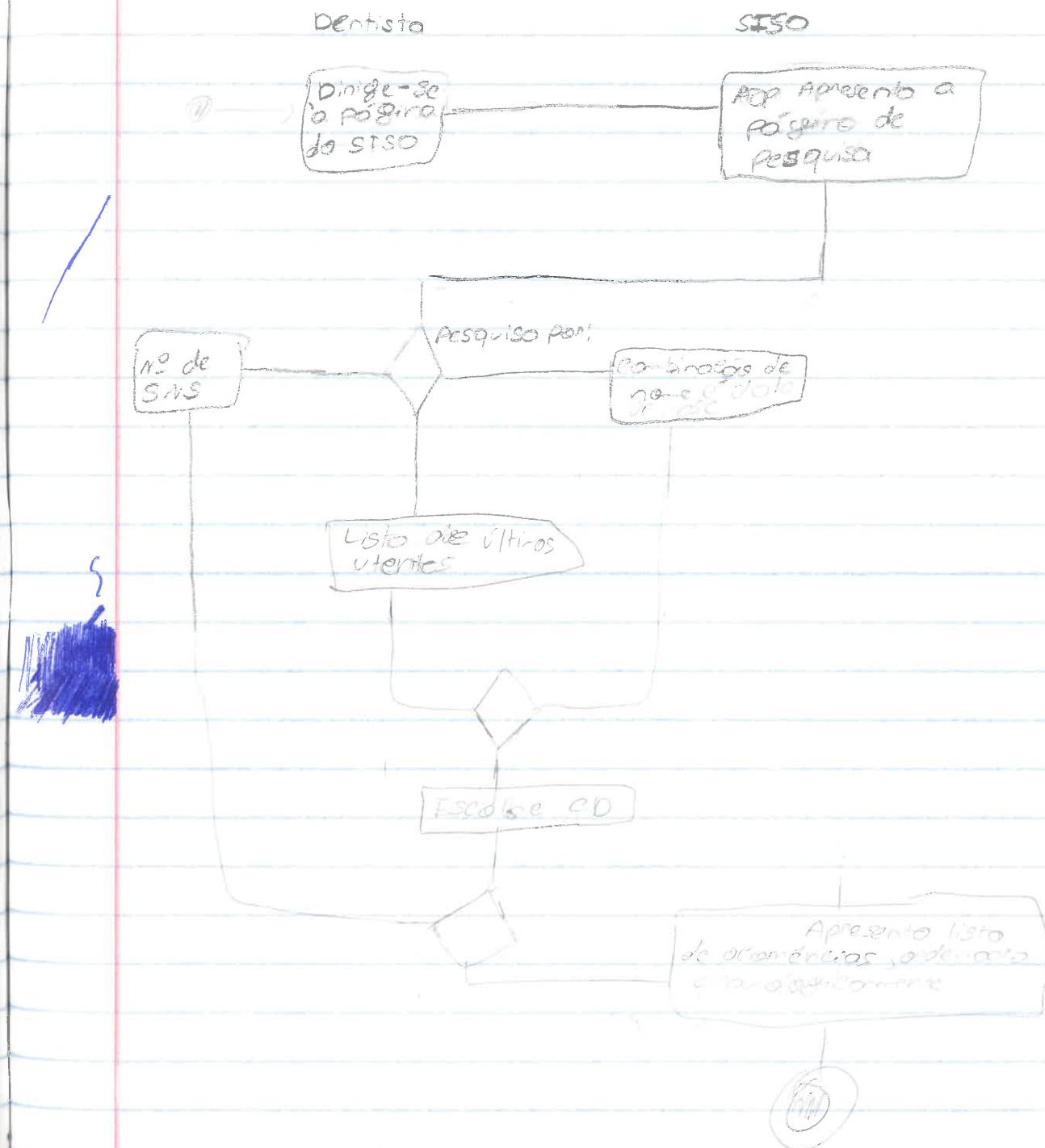
Requisitos Funcional:
Ex: um programa
tem que fazer a
inscrição de um aluno

Requisitos não funcionais:
o programa permite
fazer log in com o
facebook.

Um analista deve fazer uma lista de requisitos que servirão como contrato entre o cliente e quem irá implementar o sistema. Há projetos com grandes orçamentos por isso essa lista deve ser cuidadosamente elaborada.



C - CREATE
R - RETRIEVE
U - UPDATE
D - Delete



MA = Comunicação entre o SIS

- MD = Peça que o clube
- Regista diagnósticos
- " " tratamentos
- Verifica inf. do cliente

Médico atende o cliente - Recebe login e password
- Verifica se é credível
- de pagamento existente no SIS

ACSS = validação de CD

ER = validação dos
- validador CD

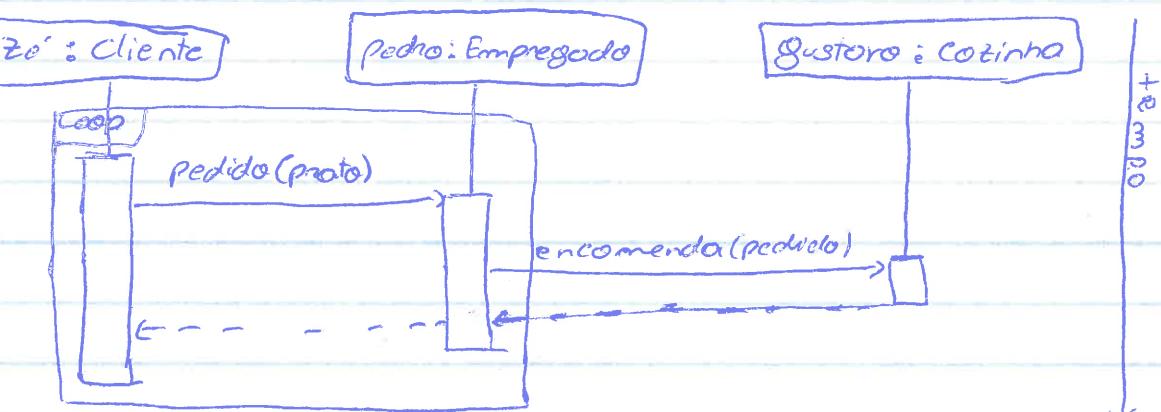
Diagrama de sequência

Anatomia → estrutura, ponto de vista das partes
VS

Fisiologia → comportamento, ponto de vista da ação

Em UML temos grupos de diagramas usados para todo uma componente, ex: Diagramas de classes não é possível verificar evolução ao longo do tempo.

○ Diagramas de Sequência podem ser usados para representar Colaboração entre objetos em gara



pode haver atividades assíncronas ou síncronas

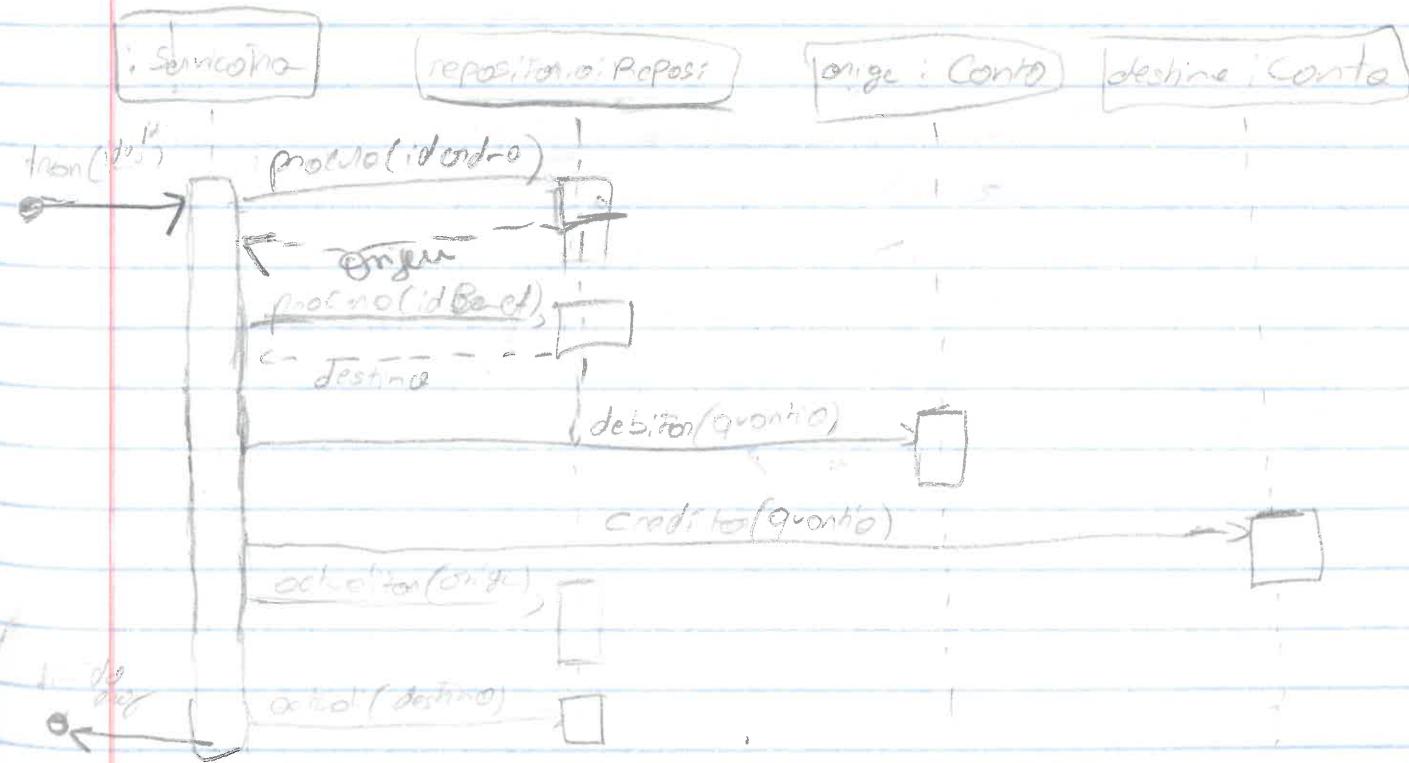
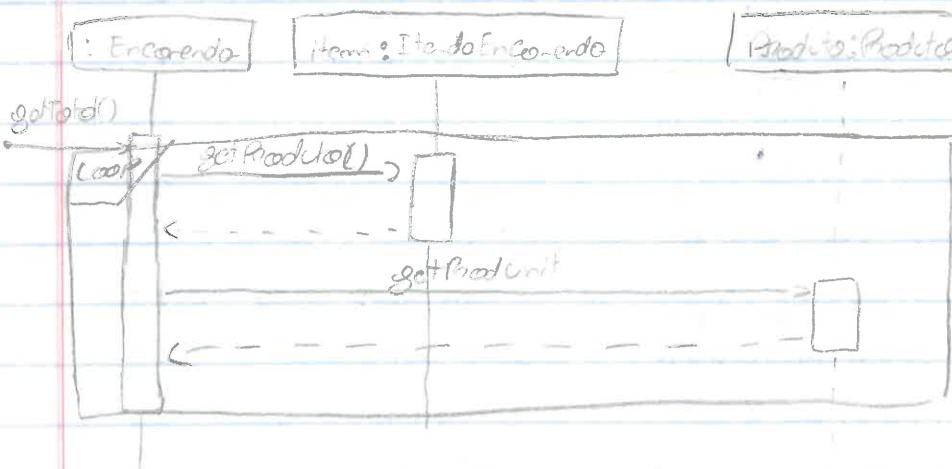
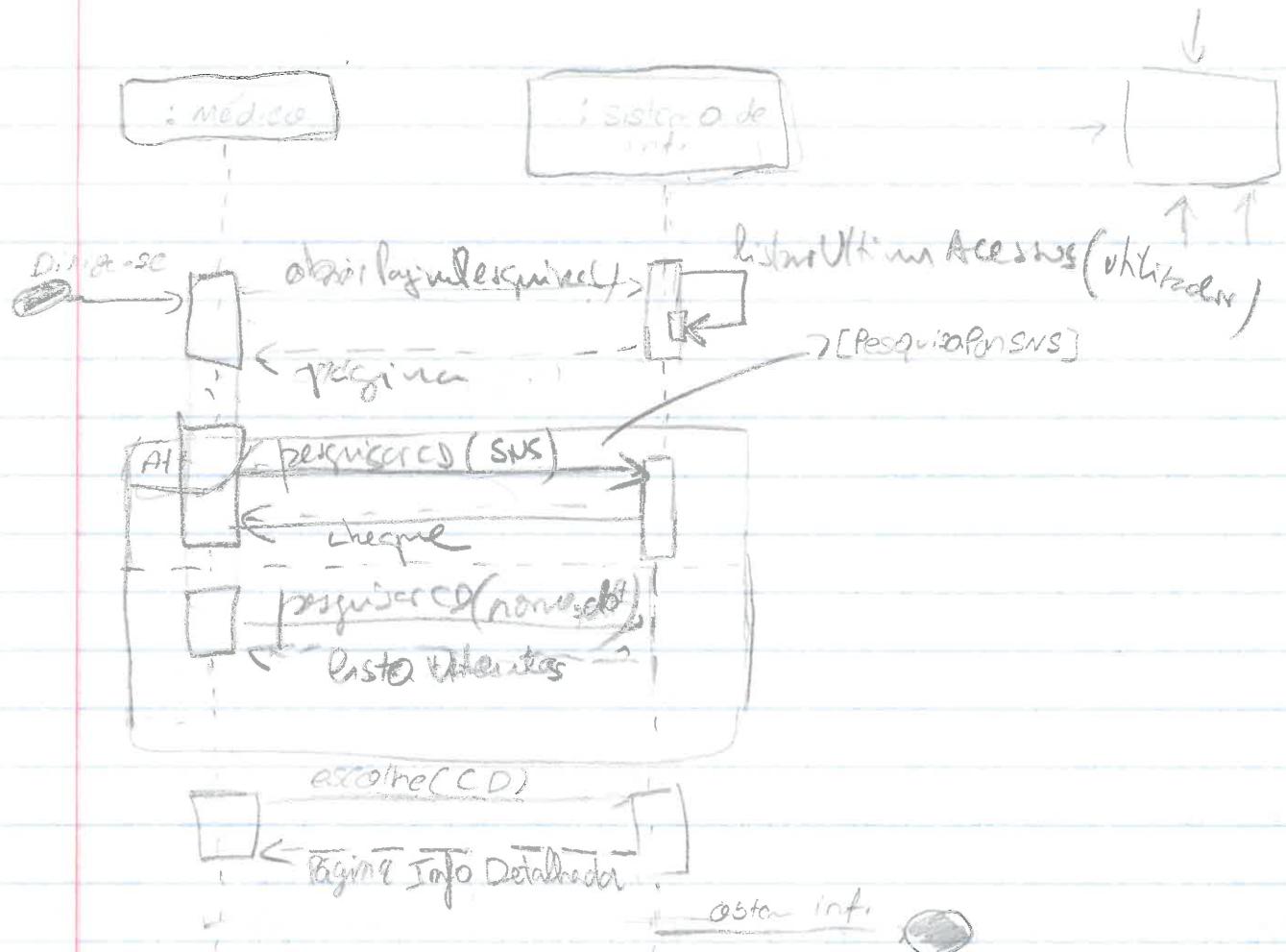
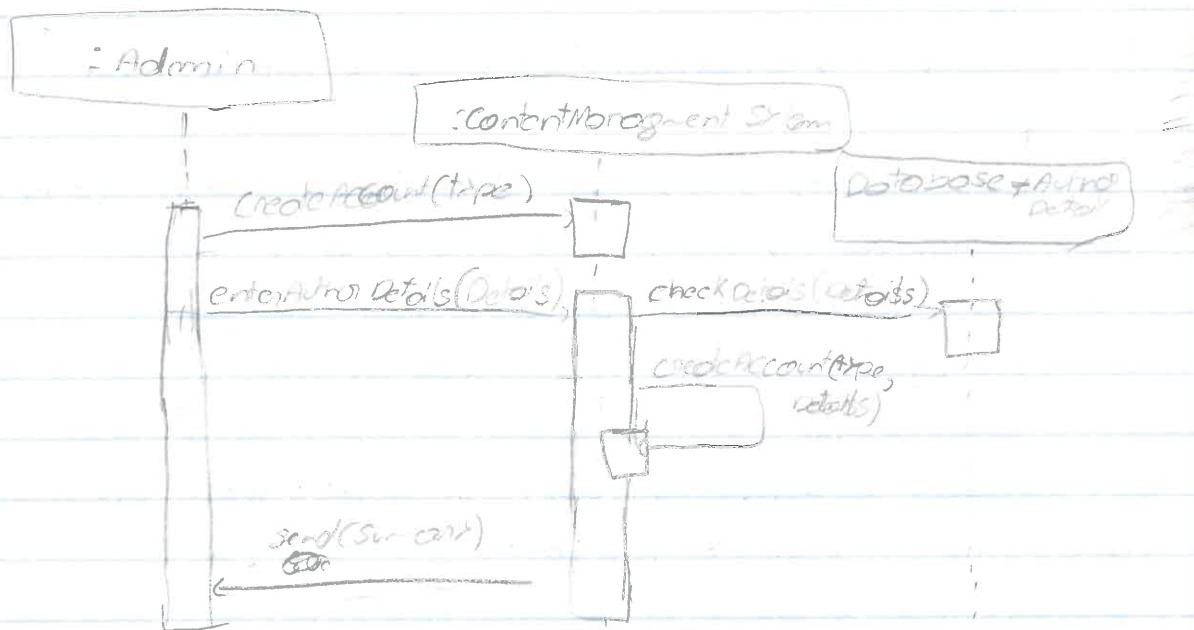
! n espere compõe
rotor de retorno

esperar pelo retorno
de retorno

Também é importante modelar a criação ou destruição de um objeto

É possível fragmentar o diagrama de maneira a o grupo ação que só ocorrem em determinados situações (if) else ou loops.

DSS - Diagrama de seqüência de Sistemas



UML na visualização de código → programador

Os diagramas de interação ajudam-nos a encontrar os métodos e a perceber a sua interação, só depois se deve fazer o diagrama de class

O UML é uma linguagem de modelação, não influencia os conhecimentos do programador

Ao desenhar um diagrama atribuímos responsabilidades aos objetos

Coupling → dependência de classes que necessitam de outras para realizar suas tarefas

Exemplo: numa classe figura é possível ter método que interagiam com círculos e retângulos, apesar de não haver coupling não é motivo coeso.

O coupling é necessário mas deve ser controlado

Quando duas classes estão dependentes, é difícil fazer alterações ou escalabilidade pois se uma é alterada, afeta os outros.

Information Expert → distribuição de responsabilidades por objetos



Quem deve instanciar uma class (coupling)? criando um objeto
Se uma classe agrupa ou usa outra classe

Forward engineering

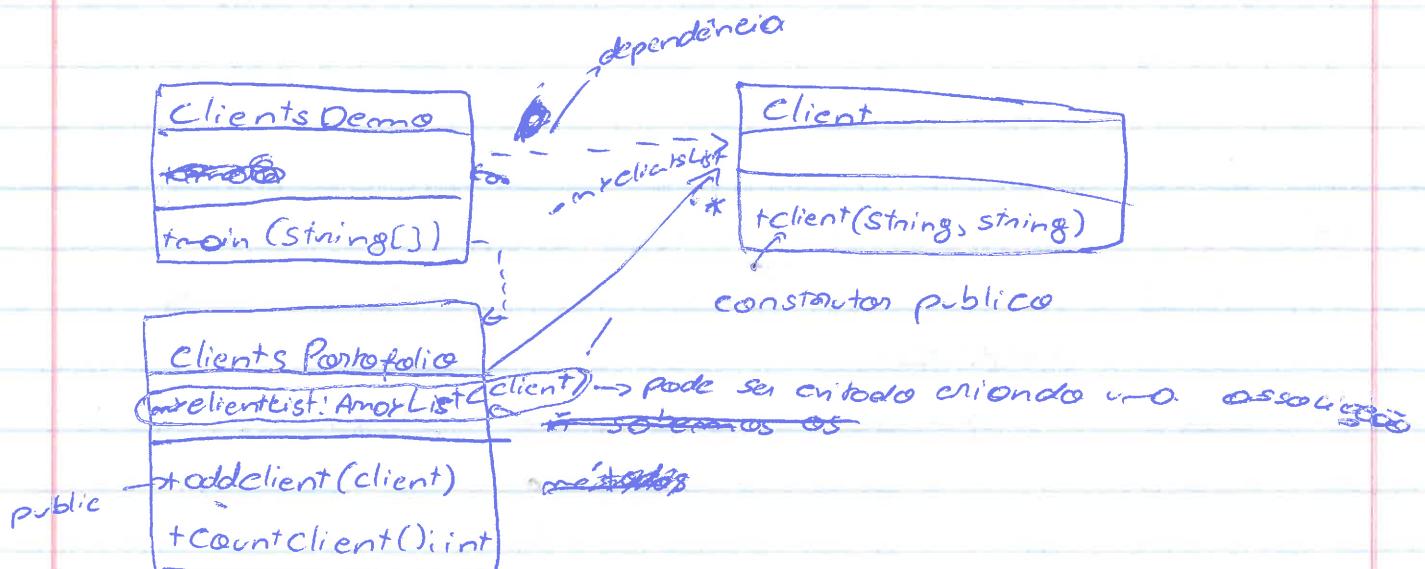
Do modelo para a implementação

Reverse engineering

Da implementação para o modelo

Round-trip engineering

Transição transparente nos dois sentidos

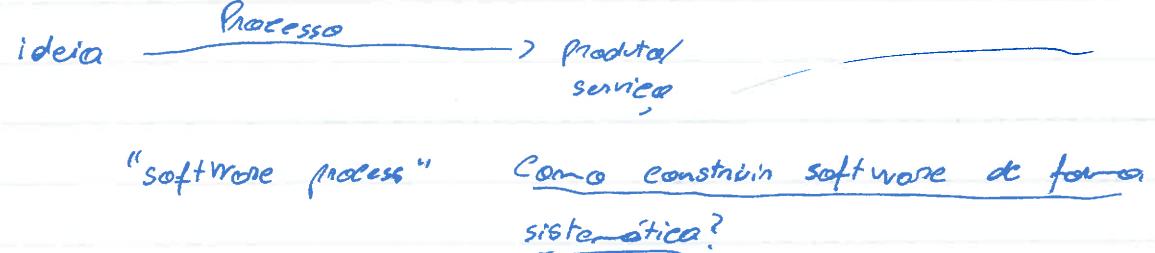


criando a seta afirmamos que na classe **ClientsPortofolio** há referências para objetos do tipo ~~Client~~ **Client**.
A nova geração interessante é é importante.

Há coupling entre o **ClientsDemo** e os outros

Diferença de associação pois na **ClientsPortofolio** há um agregado de **Client**

- todos static são sublinhados
- priorite é representadas com um menor



As empresas têm colaboradores a entrar e a sair, assim a construção de software não pode estar dependente dos colaboradores

processo fundamental de um processo de Eng. Software

- Análise de requisitos
- Especificação de requisitos
-

~~requisitos~~ garantia de qualidade

verificação → verifica se o produto está a funcionar bem ~~de~~

validação → se efectivamente faz o que os requisitos pretendiam

manutenção corretiva → Correção de bugs ~~adicionais do cliente~~

- " adaptativa -> mudanças de ambiente de execução
- " Evolutiva -> fazer actualizações, melhoramentos

Em engenharia de Software estudos indicam que 67% dos custos é para a manutenção do software.

Desenvolvimento de software

Aproximação clássica: waterfall

Há uma sequência de acontecimentos no desenvolvimento, cada fase tem um resultado, antes de avançar cada resultado tem de ser aprovado. Passos anteriores só são repetidos em caso de erros.



Este processo de engenharia tornou-se obsoleto porque pelo pressuposto que os requisitos ficam bem definidos logo no início, mas na construção do software o que acontece muitas vezes é a mudança de requisitos ou ~~de~~ crescimento.

Também pressupõem que o processo é feito de maneira muito lenta, em que tempo de resposta não é piorado caso fatores que influenciam o processo.

Quanto mais tarde aparecer uma alteração mais dispendiosa vai ser.

Chamam os utilizadores quase no fim da construção para testar o projeto sistema é perigoso, pode ser o que o cliente quer. Há ~~uma~~ não troca de informação entre os clientes e quem desenvolve software, à medida que o software é desenvolvido o cliente compreende melhor o que quer.

Abordagem incremental

Realizar a solução às partes

- " iterativa

Correção por fazer algo pouco detalhado e ir expandindo a sua complexidade

por vezes os requisitos não funcionais podem ser mais fundamentais do que os próprios funcionais, por isso não faz sentido afirmar que são despesáreis.

para fazer o levantamento dos requisitos ^{funcionais} de sistemas usamos os casos de utilização

~~casos de utilização~~

um requisito de usabilidade deve ser pode ser explicado de forma monetária: 6 no de formulários preenchidos com erro deve ser abaixo de 10%.

O utilizador deve conseguir fazer um pedido em menos de 5 minutos.

FURPS

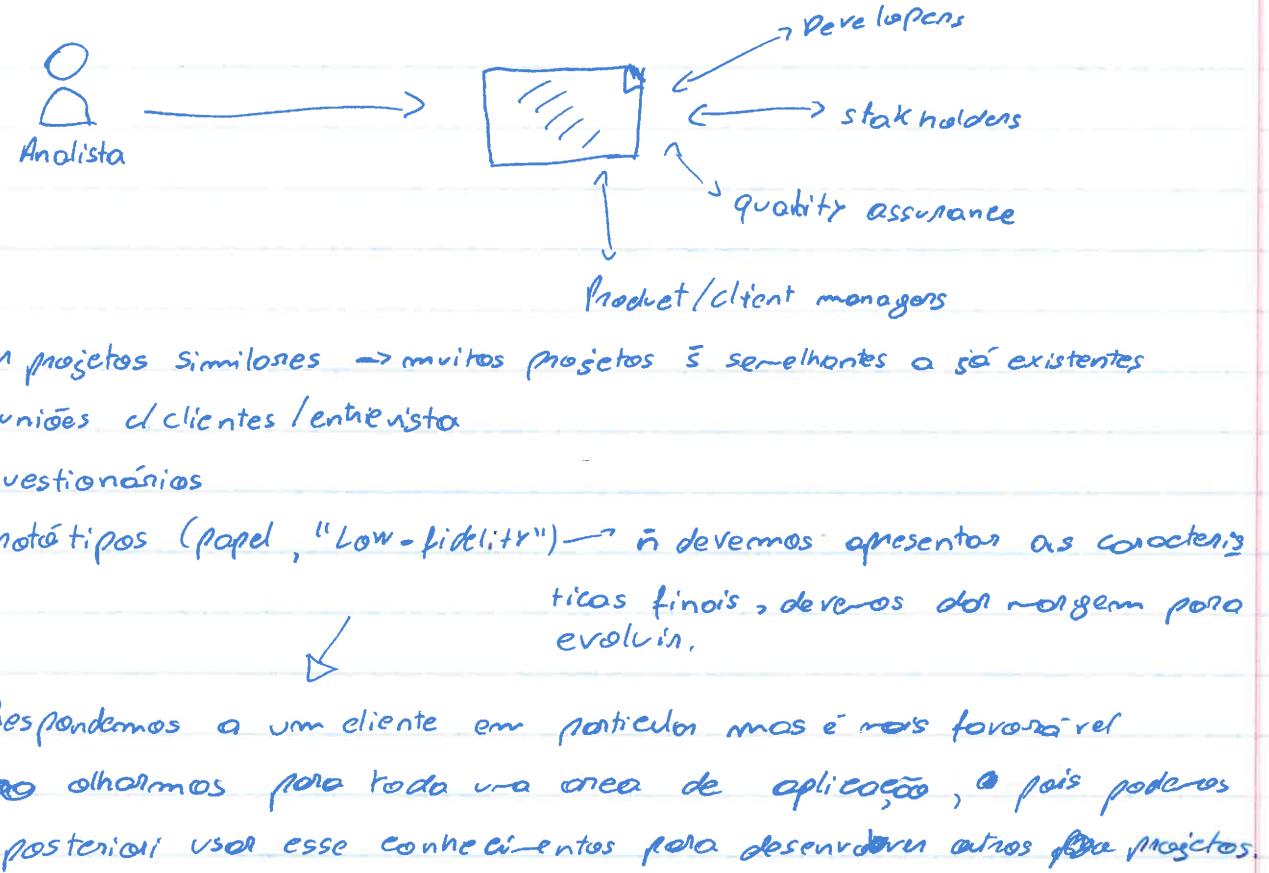
SMART
Specific
Measurable
Attainable
Relevant
Traceable

- Specific → termo que se refere a algo que é específico
- Measurable → algo por onde o programador se passa quando é feita a avaliação
- Attainable → termo que se refere a algo que é possível
- Relevant → Algo de interesse para os stakeholders
- Traceable → Algo que se possa monitorizar

Os stakeholders não são a única forma de obter requisitos

Falar com os target users ou até mesmo envolver-los no processo de desenvolvimento, bem como pessoas especializadas na área em que o sistema a desenvolver vai atuar.

Ex: uma aplicação que serve para detectar a doença de Parkinson, um médico especializado na área pode ser uma boa fonte de info., bem como um utente.



Para o projeto ser valorizado os métodos de aquisição de requisitos
ver OpenUp → levantamento de requisitos

Site e aplicação móvel

fornecedores → são contactados diretamente

aqueles que têm opção de comprar produtos
pagam uma taxa anual para que
os produtos estejam na
aplicação

Aplicação → intermédio

- ✓ ~~disponível~~ disponibiliza os módulos de manutenção
• organizada, de fácil acesso e convincente

Transporte → Inicialmente - Recrutar transportadora como parceiro colaborador.

- Mais tarde - Avaliar o se é rentável possuir os próprios meios de transporte

~~Cliente fornecedor~~ → pessoas mais afastadas das zonas rivais
~~Cotavação~~ através do marketing
pessoas com mais dificuldade em deslocar-se

~~Serviços de registo dos utilizadores e armazenamento dos dados~~

Aplicação web e mobile

- ↳ Acesso fácil
- ↳ Ligação à Internet

funcionalidades:

- Escolher produto
- Apresenta imagens representativas
- Permite filtragem:
 - por fornecedor
 - por tipo de produto
 - modo de produção
 - categoria pré-definida
- Disponibilização de conjunto de produtos divididos por temas

• Após escolha é possível adicionar a um cesto

- no acto de compra - → É apresentado o total
 - pode pagar por transferência
 - por pal
 - Entidade referência

• ~~contato~~ contacto com os fornecedores e transportadora

- A entrega pode ser normal (dentro de 12 horas durante o horário diurno) ou especial (entrega rápida num prazo máximo de 8 horas), taxa adicional

Pontos de inovação:

- Agendamento regular dos enregos (semanal, mensal, anual)

- Os fornecedores podem pagar uma taxa para o seu produto ser colocado em destaque

- Stream live stream dos colheitas (exemplo no canhão de neve)

Aplicações do unified Process continuado

Software modular traz várias vantagens

• UML surge no âmbito de modelar a informação dos vários objetos, isso pode ser aplicado à programação orientada por objetos

↳ junção de várias linguagens na tentativa da unificação numa linguagem que comum a todos.

Houve ainda várias concretizações do UML.

A IBM disponibilizou o UML.

~~Classificação~~

Modelo 4+1

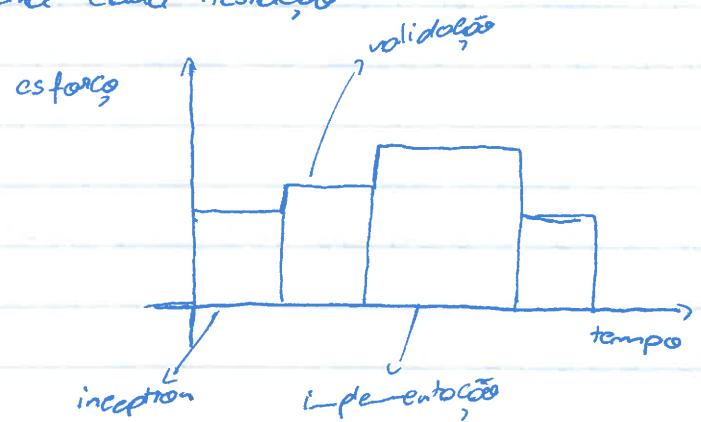
- conceptual - o que se espera do sistema, qual os seus objetivos de usabilidade
- Físico → o que é preciso para o sistema funcionar

Logical view → diagrama descreve estruturas de class.

Process view → dia. de dia. de sequência

A realização de um projeto deve ser iterativa & incremental
as disciplinas podem ser repetidas
Ex! A análise de requisitos pode voltar a acontecer

milestone → é um objetivo / ponto de controlo planeado para cada iteração



No final da inception → milestones - A equipa percebe os requisitos? São proto. & adequados

Na fase inception tenta que seja uma visão geral do projeto

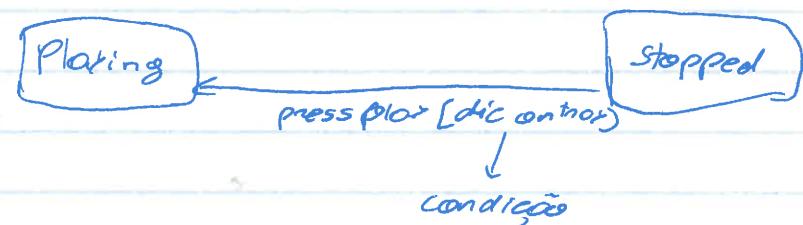
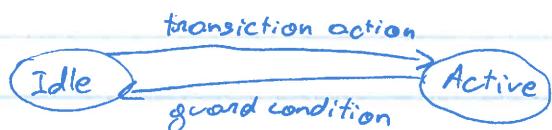
Elaboração → ~~fixar~~ fixar a visão do projeto, perceber as implicações técnicas do projeto e se é executável

- Detalhar os casos de utilização
- Descrição da arquitetura do software

Vai a haver uma sessão de validação de requisitos para receber feedback.

milestones → elaboração

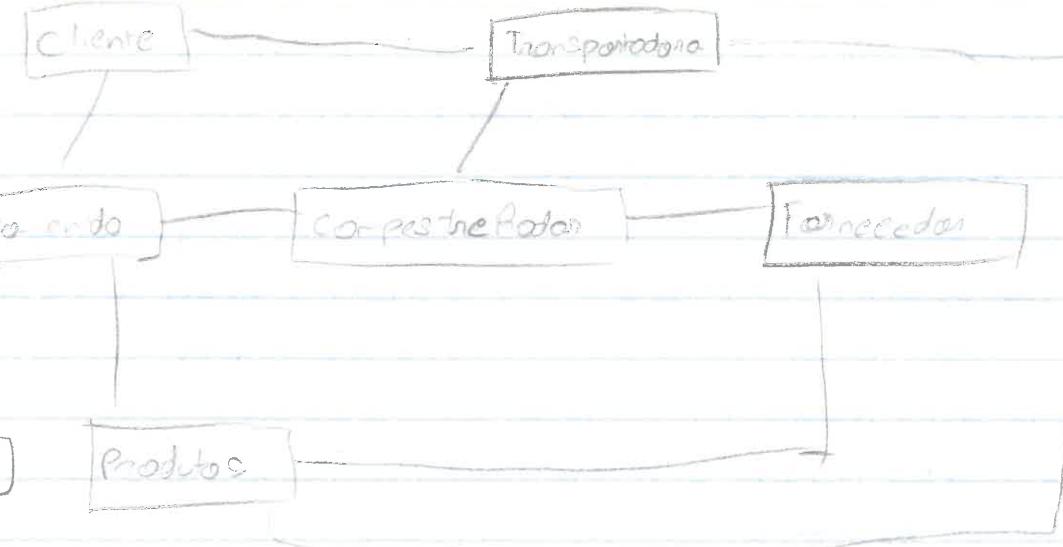
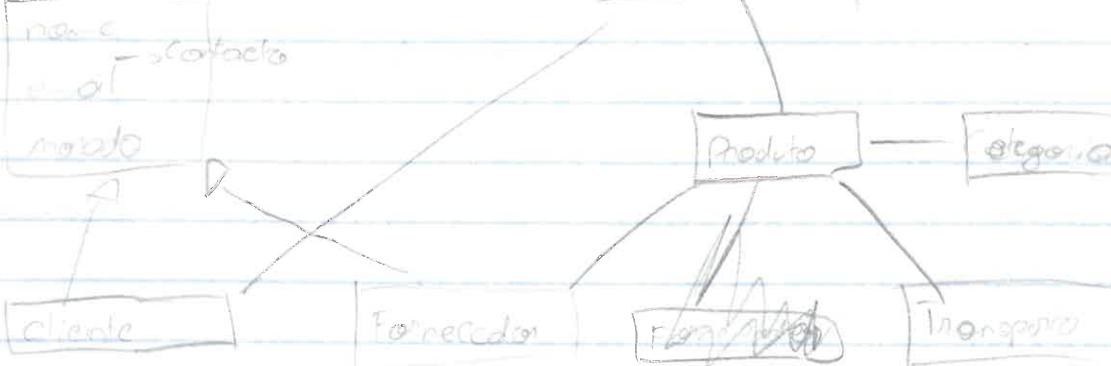
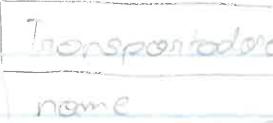
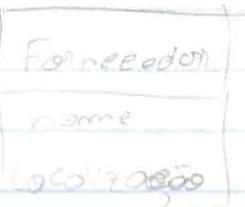
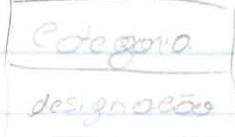
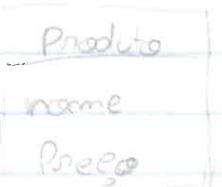
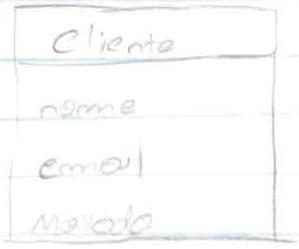
Diagrama de estados → modelam o estado de certo sistema



Quando usar?

Quando estivermos objectos cujo comportamento depende dos estados

No projeto usamos os dia. de estados para algumas entidades onde é interessante observar os seus ciclos de vida.



O que é para fazer nos requisitos não funcionais?

Vai o pena estender o protótipo para o mobile?

Onde os produtos têm o destaque?

Onde a pessoa tem o acesso?

Fornecedor →

Produto

início

Visões de arquitetura (perspectiva)

~~Um arquiteto preocupa-se~~

um arquiteto planeia, constrói um plano-modelo que pode ser usado/reaproveitado para outros projetos.

SaaS → Software as a Service

Ex: Word online

Não são bens comprados mas utilização de serviços.

Este tipo de software tem de estar preparado para ser usado em várias plataformas.

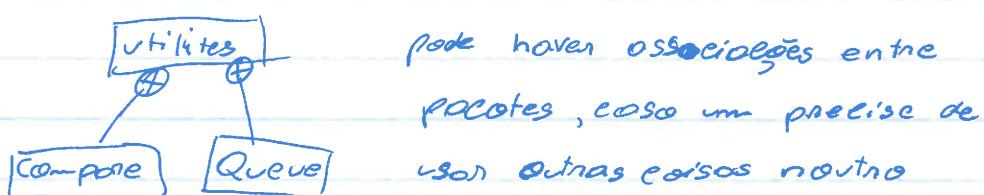
Requisitos não funcionais são determinantes.

Arquitetura lógica do software

Arquitetura lógica

- Organização lógico
- Package e podem

Ex:



Arquitetura de componentes de software

Cada componente obedece a um contrato de funcionalidade

- Encapsulamento não precisamos de saber como estão implementados
- só precisamos de saber trabalham com eles

Em UML é possível ilustrar estes associados, é ainda possível mostrar os componentes usados no interior de cada módulo ou não mostrá-los

Arquitetura "física" de instalação da UML

Algo que necessitamos para realizar a tarefa

Ex: para utilizar certas ferramentas é necessário instalar componentes adicionais

As aplicações podem precisar de um ambiente de execução, ou seja, de determinada sistema operativo para ser executado.

UML

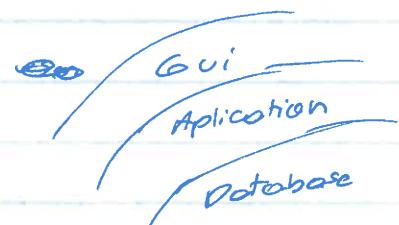
{ <ul style="list-style-type: none"> D. pacotes D. componentes D. instalação 	(deployment)
---	--------------

Estilos de arquitetura

Cada arquiteto deve ter a noção de decisões que vão tomar a respeito em relação a que arquitetura vai usar. Para isso deve estar ciente das implicações da sua decisão.

Arquitetura por camadas

Arquitetura por camadas



Pode ser utilizada para dar serviços → podemos mudar a base de dados nos casos não dependendo a funcionalidade onde não é efetuada

Estes corredos de serviços só comunicam com os corredos de cima.

Problemas:

- e

• nos casos de utilização:

Diagrama: Adicionas ~~o~~ administradas
→ representas os interesses do investidor ~~o~~ no sistema

mostram como funciona o sistema.

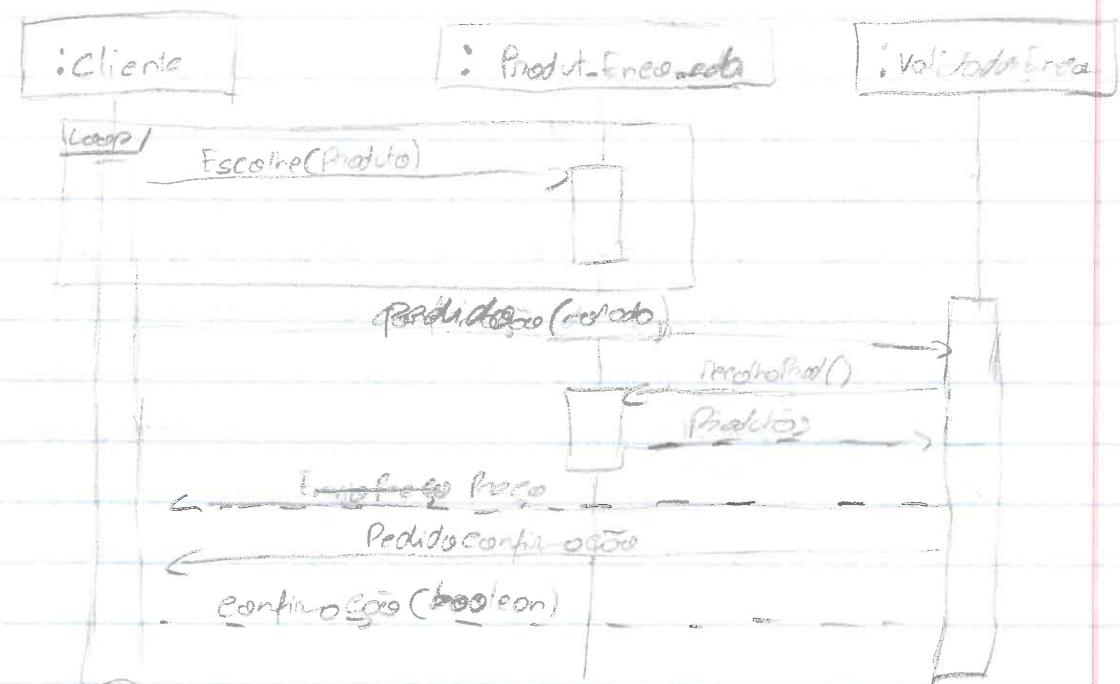
- O Modelo de Domínio - Coloca dotos onde é necessário armazenar informação
 - Especificar os atributos
 - Tentar identificar mais classes entre relações

Processos de Trabalho - Adicionas ~~o~~ na validação de pagamento adicional decision node

Na análise de requisitos só mostras os produtos que estão disponíveis

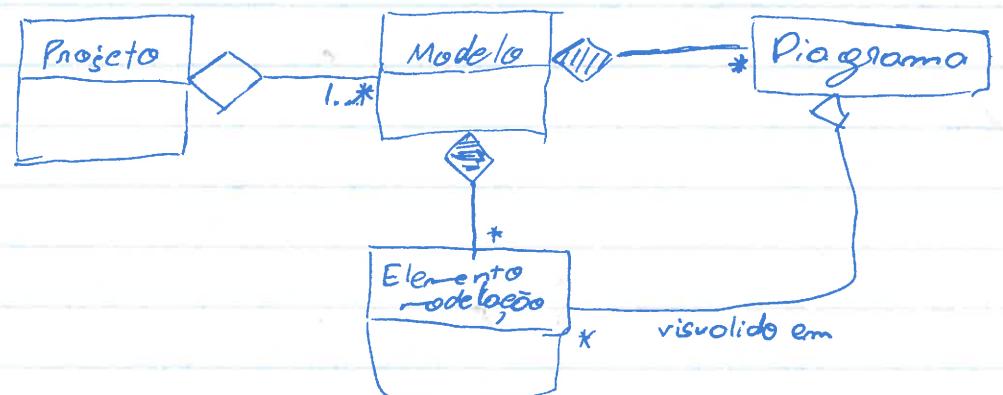
→ Isto não é necessário ter a partição da aplicação.

- verbos no infinitivo
- Union object node para a cneamenda
- Pode-se utilizar mais do que um diagrama
- Cuidado com os os eventos que podem causar mal, adicionar decision node se necessário



Apesar do UML estás ser independente do linguagem de programação mas está um "pouco" orientado a objetos, é mais fácil de usar quando orientado a objetos.

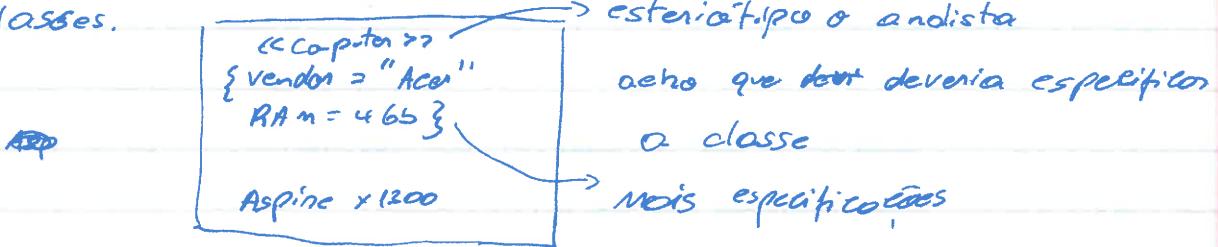
podemos usar UML para programação e para muitos aspectos da vida real



No UML

Temos diagramas
de estrutura e outros de comportamento

Também tem estereótipos Ex: Num diagrama de classes podem ter classes normais mas uma nota a ofício que é classes.



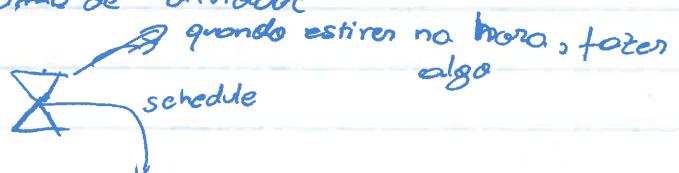
Restrições

Podemos adicionar regras de interpretação

ver slide 14

Os diagramas de classes podem servir para explicar um sistema por camadas, notando uma hierarquia, ou podem servir para explicar uma biblioteca sia implementada. Ex: SOA

Diagrama de otividade



O open up é uma linguagem que nos explica que o projeto projeto só deve ser desenvolvido usando iterações

Dados vs Informação

Dados conhecimento ainda não processado ou analisado, difícil de perceber.

Informação Conhecimento organizado, co~~de~~lo~~de~~ e filtrado de maneira que seja fácil de perceber.

Sistema - conjunto de componentes que interagem para a atingir uma finalidade.

Ciclo num ciclo de vida de um projeto, quanto mais tarde se fizer alterações, maiores custos das variações

Anteriormente considerava-se que o catálogo de um sistema era um processo sequencial e que os requisitos são estacionários mas isso não é verdade e o que ocorre por acidente é que existem problemas que são criados muito tarde e são muito dispendiosos. O problema é que os progressos no projeto só são apreciáveis nos momentos em que o produto final está funcionando como pretendido, pois o que produzido não é só o pretendido.

Os custos aumentam e têm a tendência de mudar durante o desenvolvimento do projeto, logo se considera é portanto

Recomendações:

- Desenvolver software de ~~tal forma~~ modulares
- usar modelo visual
- Gestão da mudança

Cada iteração deve produzir algo que se possa testar. O projeto por cada iteração deve apresentar.

los resultados aos stakeholders

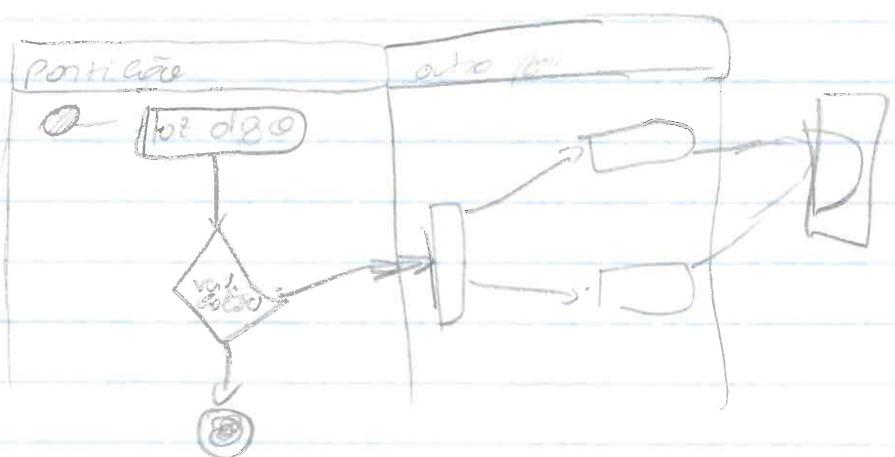
Pode-se assim fazer a distinção entre um método tradicional e um ágil
unified process

Modelação:

- A comunicação torna-se mais clara e suave
- mostra ou esconde vários níveis de detalhe conforme é apropriado
- Pode suportar modelos de programação atômica

Diagrama de atividades

modelos fluídos de trabalho



Objetos e classes

Esquema mental comum à análise e à programação

modela-se entidades reais, conceitos, etc.

um objeto é um entidade que tem atributos e funções

classe → categoria de objetos que partilham os mesmos semelhanças, é ~~uma~~ uma abstração

• o um objeto é uma instância de uma classe

As classes não são conjuntos de objetos

atributo → propriedade da classe

método → a operação, serviço pedido a uma classe

multiplicidade: 1 - exatamente

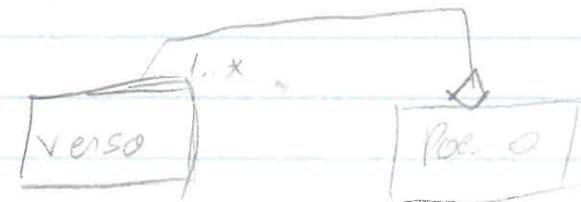
0..* → zero ou mais

* → 1

1..* → 1 ou mais

0..1 → opcional zero ou 1

Agregação → uma classe é parte de...



Também há hierarquia

• Modelo de Domínio

solve como um mapa, dicionário visual

É representado com um diagrama de classes sem operações mas podem ter atributos.

classes em análise

- Símbolos à análise e de criação
- Resultados da era análise de requisitos

⑩ Class em programação

- Resultado do design de implementação
- Escrito numa linguagem concreta

Criar um modelo de domínio

- Identificar os objetos
- Representar classes num DC
- Adicionar associações

O nome das classes deve ser singular

No diagrama de classes podemos especificar o tipo dos atributos como string, int, double, etc. Atributos sublinhados são estáticos

Agregação vs composição

A é uma parte
de B.

B é portátil

B é parte constituinte
de A.

B instâncias B não
portáteis

Pode haver seções reflexivas

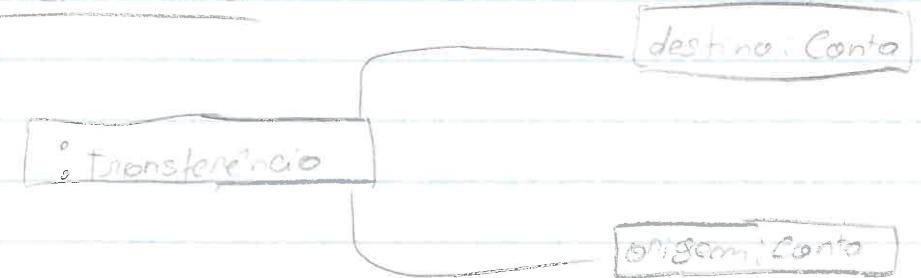
[Entidade] [verbo] [Entidade]

Anotações também S elementos de modelação

Lê-se da esquerda para direita, de cima para baixo

As associações devem ter um nome

Diagrama de nomes



O objecto do instância pode ser criado

Case de utilização

O sequência de ações que um sistema executa e que produzem um resultado

Impõe foco na competência nos episódios de uso
e ~~o caso~~ naquilo que é considerado um resultado de uso

Um actor ~~to~~ interage com o sistema de maneira produtiva
algo de valor

Faz-se uso de uma notação para mostrar como se usa
o sistema

Principais entidades:

- Ator
- Caso de utilização
- Cenário

6 COU tem um fluxo típico e ~~outras~~ outras alternativas,
5 situações para elas mas que levam ao mesmo resultado.

Para construir um COU temos de perguntar o que é que acontece, até ~~o~~ obtermos o nosso ~~o~~ cenário típico e de seguida deve interrogar que mais é possível acontecer.

A normativa descreve como é que o caso ocorre e acaba, deve apenas fazer referência a interações dentro do sistema em análise e deve elencar o tipo de interação entre acto e sistema.

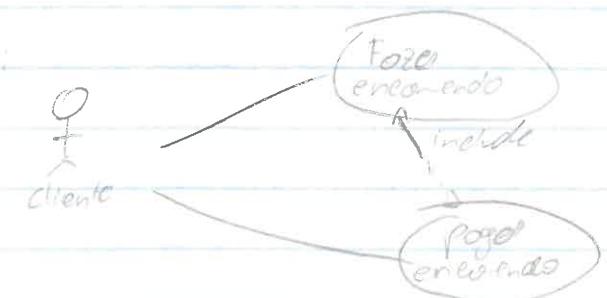
Como encontrar os COU?

- Identificar o fronteira do sistema
- " " atores que interagem com ele
- " " ~~o~~ ativação para interagir com o sistema
- Definir COU que satisfazem os objetivos do sistema

Os casos de uso podem ser agrupados em pacotes

Diagrama de casos de utilização

Alguns COU podem incluir outros quando o escrito inclui



extend

A estende B se A depende de uma condição ~~o~~ de acto

Resumindo, para que serve o COU?

uma vista de um sistema que destaca o comportamento observável.

Captar as funcionalidades do sistema é a implementação

Requisitos devem ser selecionados

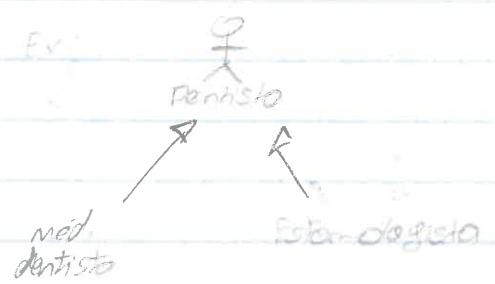
Requisitos funcionais

- Captam o comportamento do sistema
- Pode ser captado no COU
- Pode ser descrito em forma de diagramas de comportamento (UML, sequências)

Requisitos não-funcionais

- Designadas por atributos de qualidade
- Robustez, portabilidade, etc.

No num diagrama de COU pode haver especificações de papéis nos atores



Ainda pode haver a diferenciação entre atores

primários, os que usam o sistema para realizar uma tarefa, os COU são iniciados por eles.

E os secundários fornecem serviços ou informações para o cenários dos COU

6 COU tem um fluxo típico e ~~outras~~ outras alternativas,
5 situações para elas mas mais que levam ao mesmo
resultado.

Para construir um COU temos de perguntar ~~o que é~~ o que é
que acontece, até ~~o~~ obtermos o nosso ~~o~~ cenário típico
e de seguida deve interrogar o que mais é possível acontecer.

A normativa descreve como é que o caso começo e
acaba, deve apenas fazer referência a interações dentro do
sistema em análise e deve elencar o fluxo de informação
entre ator e sistema.

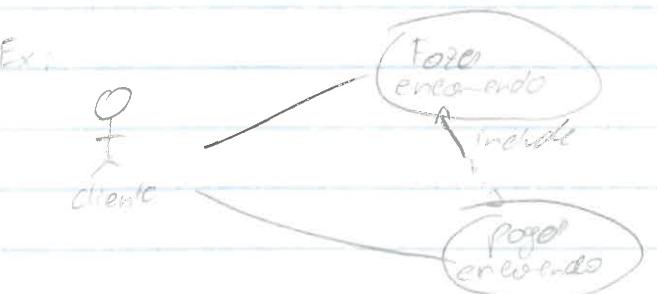
Como encontrar os COU?

- Identificar a fronteira do sistema
- " " " " " atores que interagem com o
- " " " " " motivoção para interagir com o sistema
- Definir COU que satisfazem os objetivos do sistema

Os casos de est. podem ser agrupados em pacotes

Diagrama de casos de utilização

Alguns COU podem incluir outros quando o cliente inclui



extensão

A extensão B se A depende de uma condição ~~de~~ de extensão

Resumindo, para que serve o COU?

Uma vista de um sistema que destaca o comportamento
observável.

Captar as funcionalidades do sistema na implementação

Requisitos devem ser selecionados

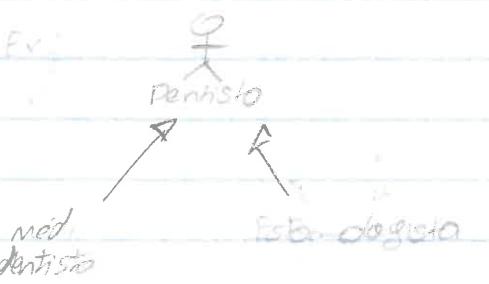
Requisitos funcionais

- Captam o comportamento do sistema
- Pode ser captado no COU
- Pode ser descrito em termos de ações de utilizador
- Qualidade, sequência

Requisitos não-funcionais

- Designados por atributos de qualidade
- Robustez, probabilidade, etc.

No num diagrama de COU pode haver especificações
de papéis nos atores



Ainda pode haver a
diferenciação entre atores
secundários, os que

usam o sistema para
realizar uma tarefa, os
COU são iniciados por eles.

E os secundários fornecem serviços
ou informações para o cenário
dos COU

É possível organizar os COU em termos e em pacotes

B) Sistemas que têm mecanismos de autenticação, o login e logout é de com fazer parte dos COU

C - Create

Para fazer a distinção entre

R - Retrieve

estas várias situações deve usar

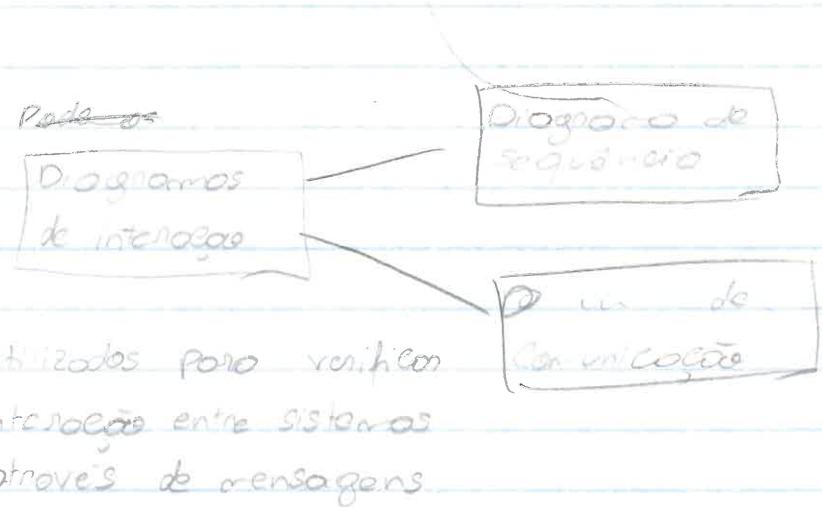
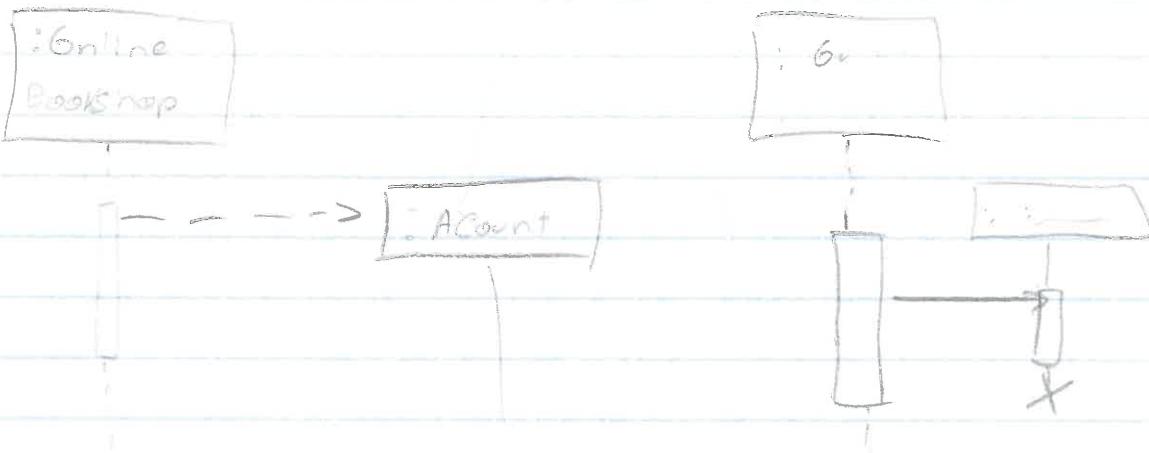
U - Update

mais COU

D - Delete

Este fuso deve ser explicado num DA

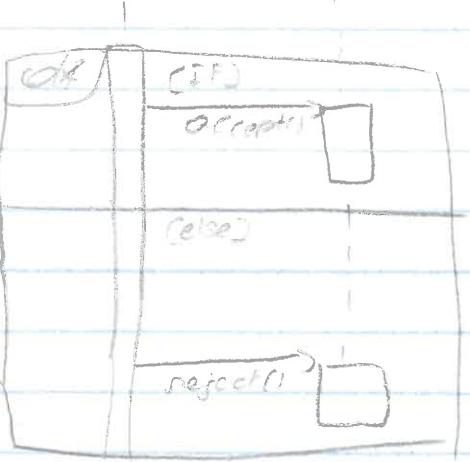
Também é possível modelar a criação e destruição de um objeto



utilizados para verificar
interacção entre sistemas
através de mensagens

Também podem ser usados para verificar a interacção entre
os objetos em gera

A mensagens podem ser sincronas ou assíncronas, consequentemente
espera retorno ou não respetivamente



Diagramas de sequências

mostram para um cenário de um COU:

- eventos que dão externas geram
- necessidade de troca de informação com os sistemas externos

Estilo de Construção:

- O sistema é tratado como uma entidade, não mostrando componentes internos
- As mensagens que chegam são operações invocadas por entidades externas

Vem novo diagrama de se ser criado por código C++

Q. uma operação de sistema corresponde a um ponto de entrada no sistema, na construção de um modelo dinâmico deve-se mostrar o rede de objetos que vai ser criada, esses objetos são instâncias de classe que se devem identificar

¶

Para vam ~~o~~clar objetos:

- #1 - modelo mental para código
- #2 - UML e depois #1

Ao desenhar um diagrama de interação estaremos a atribuir responsabilidades, não método exato para o fazer

Os princípios do OOP (Object Oriented Design) São:
diminuir o "coupling" e aumentar a coesão

2)

↳ foco único

dependência que
uma classe tem de outra
classe

Dare principio criar o DSS e só seguir o
de classes, dessa maneira podemos logo verificar
os dependências com outras classes bem as portadas,
métodos e tipos de retorno.

Tech. de Software → conjunto de atividades e resultados associados
que produzem software

Atividades fundamentais de um processo de Es:

- Especificação de requisitos
 - Ambito
 - visão do problema
 - Levantamento de requisitos
- Esp. de software
 - Modelo do solução
 - Interações com sistemas externos
 - Estrutura de dados
- Desenvolvimento → Gerar o cod.
 - Construir a base de dados
 - Refatoração
- validação e garantia de qualidade
 - verificar correção (testar modulos/unidades)
 - validar adequação
- Instalação e Operação → Instalação
 - Entregar Entrar em produção
- Evolução / Manutenção
 - Manutenção corretiva (detetar erros)
 - " " adaptativa
 - " " evolutiva

Em IS a fase que irá consumir mais recursos, tanto é que irá ser a fase de manutenção.

• Em open UP um processo de engenharia de software é um processo de incremental e iterativo dentro de um ciclo de vida estruturado.

Popéis:

- Analista - Representa os interesses dos stakeholders, compreende o problema o seu resolvido e define a prioridade para os requisitos

- Arquiteto - Responsável por definir toda das soluções técnicas que irão afetar a implementação e design do sistema

- Programador - Responsável por implementar parte do sistema e desenhá-lo de maneira enquadradna no projeto. Possivelmente também construir interfaces de utilizador e implementar.

- Alguma entidade que irá ser afectada com o desenvolvimento do projeto. As suas necessidades devem ser satisfeitas.

- Project Manager - lidera e planeia o projeto, coordena as interações com os stakeholders

Abordagem clássica / water fall

Presupõem o desenvolvimento do projeto por etapas de fases seqüenciais, ou seja, se perde quando os objectivos de uma fase são corrigidos posso-se à seguinte, e assim de forma repetitiva a anterior.

Esta aproximação tornou-se obsoleta pois pressupõem que os requisitos iniciais mantêm-se constantes ao longo do projeto. Tal situação é rara e com essa abordagem obter é um grande gasto de recurso para implementar alterações em fases tardias de desenvolvimento. Outro grande problema com esta aproximação é o facto de só se realizarem testes à aplicação quando esta estiver concluída, o que pode ser perigoso, levando o risco de aplicação não ser o que se pretende. Deve-se envolver o cliente no processo de desenvolvimento este também irá perceber melhor o que pretende do produto final.

Mesmo como

vantagens:

- Abordagem simples
- Disciplinada

Abordagem ágil

Pretende desenvolver SW de maneira muito dinâmica e flexível a responder às mudanças nos requisitos.

Para permitir tal abordagem é necessário:

- Práticos que equilibram a disciplina e feedback
 - ↳ Ciclos curtos de entrega e desenvolvimento orientado a testes
- Aplicam princípios de design que favoreçam o construção de software flexível

Abordagem itativa

Esta abordagem dá ênfase ao desenvolvimento de trabalho em ciclos itativos de pequeno escopo e duração, cada ciclo possa por uma análise de requisitos, análise e design, planejamento, teste, cada iteração produz um resultado executável que deve apresentar aos stakeholders, a cada nova análise requisitos é feita e ciclo começa novamente. Esta abordagem minimiza o risco de não perceber alterações de requisitos em fases

tardios do desenvolvimento do projeto

unified process

6 OpenUP propõem uma proposta iterativa dividida em 4 fases:

• Inception: - Produzir documentação da visão do projeto

• Prototípico

- Desenvolver os requisitos de alto nível

- Produzir protótipos concepcionais

• Elaboration: - Balançamento dos riscos

- Produzir uma arquitetura executável, identificando

os principais riscos. Infraestrutura de opção e

de até 10%

- Orientar a arquitetura com os stakeholders

• Construction: - Incrementalmente define-se e desenvolve-se, e implementa-se e testa-se vários componentes

- Demonstrações frequentes

• Transition: - bug-fix incremental

- Optimizações

- Análise pós-lançamento

6 openUP segue o esquema do unified process cada fase acaba com um "momeo" e são respondidas questões que são críticas para os stakeholders.

↓ Mais detalhados no openUP

Na Inception é feito análise do sistema e é feita uma avaliação dos riscos e dos benefícios do projeto na tentativa de obter o consenso sobre o que vai com o projeto.

Elaboration → Escolher uma arquitetura e resolução dos maiores riscos, o objetivo é a validação da arquitetura.

Construção → Implementação, todos os funções são implementados e testados, foi criado documento

Transition → Decidir se os objetivos foram cumpridos

Visões de arquitetura

• Assuntos:

- Organização em grandes blocos

- Satisfazer requisitos

A arquitetura envolve vários componentes:

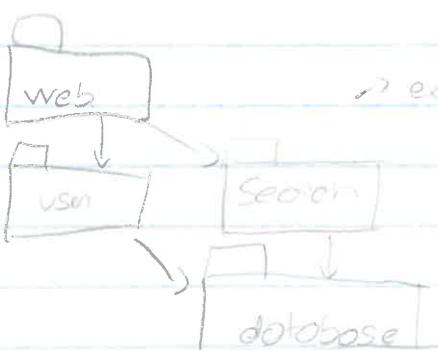
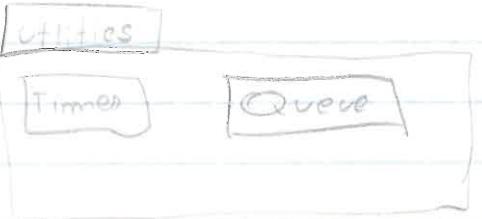
- Arquitetura lógica do software - Organização dos blocos de software, é independente da implementação

- II de Componentes de Software - Peças construída com determinada tecnologia, construção modular; algum software pode ser estabelecido

- Arquitetura de instalações - visão dos equipamentos
- As responsabilidades da arquitetura de software → deve ser só para além de erar um arquiteto que resolva o problema, garantir que está a ser usada e bem usada, a montar o código, garantir que o gerente o entende, etc.

Arq. lógica

Solução bloco (packages)



→ exemplo de arquitetura lógica, identificando dependências de uso

Componente → peça substituível a nível local, parte de divisão de um grande sistema em subcomponentes mais genéricos, basta-se de módulos que podem ser usados em vários projetos e se talvez se possam obter pré-fabricados. As funcionalidades só são expostas através de interfaces.

6

O objetivo deste tipo de arquitetura é baixar o coupling.

Arquitetura "físico" de instalações

Explica a configuração correcta do sistema (que sendas são necessárias?)

Diagrama de instalações

Nó → equipamento de hardware

Ambiente de execução → SO, serv. web

Antefato → ficheiro concreto usado pela solução
↳ 5 executáveis em nós

Estilos de Arquitetura

Data-Flow

Programas principais e subprogramas

Arq por camadas → div. módulos da solução de software com camadas, cada uma tem especialização. A de cima pedem serviços à de baixo.

Aplicações de UML ao longo do desenvolvimento do processo

UML

Elementos Comuns:

Anotações

- Páginas

- Estereótipos

COU - organizar os funcionamentos em episódios de utilização

Diagrama de Atividade:

- mundo uso: - modelar fluxo de trabalho
- modelar um algoritmo
- Descrever uma sequência de interacção entre atores e um sistema

Sob a especificação de um COU:

- bem o captar papéis
- recto e relação à programação
- pode captar fluxo de dados

Diagramas de

- Classe
- Componente
- Páginas
- deployment

Atividade

Caso de uso

interacção



Diagramas de comportamento

Modelo 4+1

conceptual

Lógico

classe

Páginas

de estados

Caso de uso

Atividade

Físico

• páginas

Componente

Processo

Seqüência

Comunicação

Atividade

Desenvolvimento

Deployment

Diagramas de

Diagramas estruturais