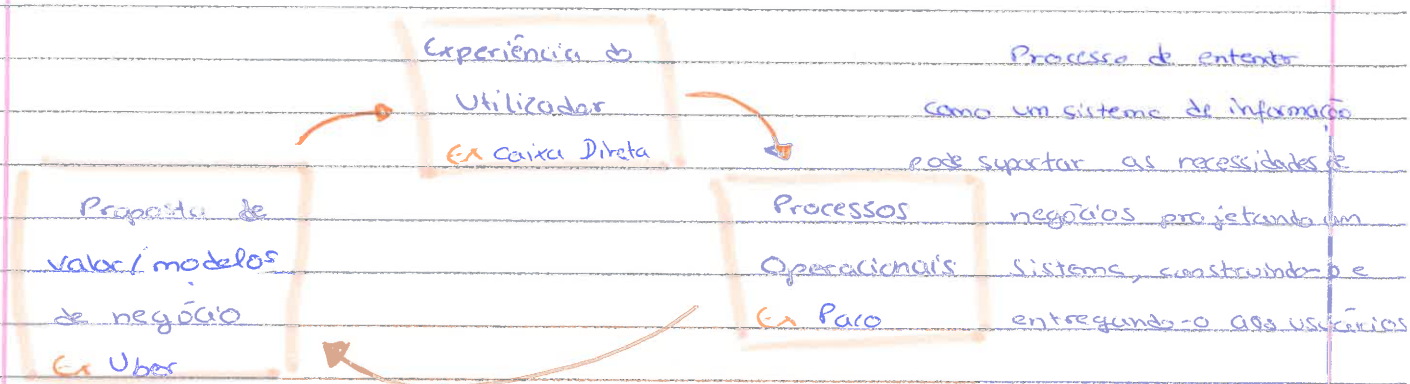


## Modelação e Análise de Sistemas

### Ciclo de vida no desenvolvimento de software:



### Transformar os processos operacionais:

- **Desmaterialização dos processos** → privilegia-se a automação e a troca eletrónica de informação, evitando papel e outros suportes físicos.
- **Mais flexibilidade para o trabalhador** → o teletrabalho, os meios de partilha de informação e de telepresença (conferência web) facilitam o trabalho colaborativo com redução de custos para a organização.
- **Melhor informação para gerir o desempenho da organização** → a informação sobre desempenho (produtividade, vendas...) deve ser transparente e completa. Esta informação pode ser obtida dos sistemas de informação em tempo real, não sendo preciso trabalhar com informação antiga.

### Transformar o modelo de negócio:

- **Estender o negócio para o digital** → aumentar a oferta que já existia no mundo físico para estar disponível no mundo digital. Ex. jornais online.
- **Novo negócio de base tecnológica** → novos produtos e serviços assentes nas tecnologias de informação. Ex. Uber.
- **Globalização (da organização e rede de parcerias)** → a facilidade de troca de informação e partilha permite às organizações formar centros de competência centralizados e ter operações descentralizadas, incluindo as colaborações com fornecedores e parceiros. Ex. design do produto feito numa cidade e a produção noutra.

## Transformar a experiência do cliente

- Estudo do cliente → desenvolvimento de algoritmos para estudar o comportamento dos clientes e dos seus interesses.
- Facilitar a compra → facilitar ao máximo o momento da compra com o uso de um canal eletrónico de venda.
- Canais de contacto com o cliente → o cliente pode usar canais inovadores na interação com a organização.

Transformação digital = recurso às tecnologias de informação para mudar profundamente o desempenho ou a área de atividade de uma organização.

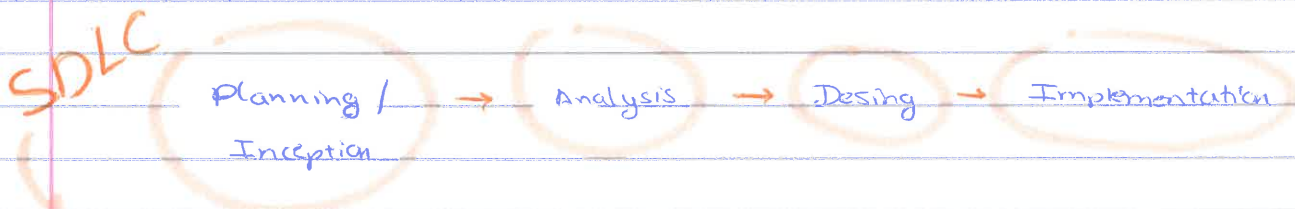
Dados: Factos, em cru; observações/medidas sobre a realidade.

Conhecimento: Compreensão das relações entre os dados; Explicar padrões que ocorrem nos dados.

Informação: Coleção de factos organizados de maneira a que apresentem mais valor, para além dos factos em si. A Informação é obtida do processamento dos dados.

Sistema de informação: Um conjunto de recursos interrelacionados (humanos e tecnológicos) para satisfazer as necessidades de informação de uma organização e dos seus processos de negócio.

Os sistemas são funções de transformação complexas.



System Development Life Cycle

## Planning:

- avaliar a viabilidade do sistema, perceber o porquê da construção do mesmo e determinar o valor que irá gerar
- Os pedidos ("request") do sistema e a análise da viabilidade são apresentados a um comité de aprovação que decide se o projeto deve ser realizado.
- Stakeholder é alguém com um interesse a defender
  - explica o que é necessário para o sistema

## Analysis:

- estudo do sistema
- sistematização dos requisitos
- Perguntas: Quem irá usar o sistema? O que fará? Quando e onde será usado?
- Passos chave: análise de sistemas existentes
  - recolha de requisitos
  - conceito de solução

## Design:

- como o sistema vai operar
- arquiteto de software
  - escolhe as componentes q não ser inseridas na solução
- Passos chave: design de arquitetura do sistema
  - do modelo de dados
  - do programa
  - selecção dos "frameworks" / estruturas

## Implementation:

- onde o sistema é realmente construído
- Passos chave: construção
  - instalação (programação) e transição

## Rational Unified Process boas práticas

- Desenvolver de forma iterativa
- Gestão explícita dos requisitos
- Usar arquiteturas e componentes
- Usar modelos visuais

- Práticas de verificação contínua de qualidade
- Gestão explícita de mudança

Um modelo é uma simplificação da realidade que vai ajudar a gerir a complexidade.

4 razões para usar modelos: ajuda a visualizar o sistema como se pretende que venha a ser

- especificar e estruturar o comportamento do sistema antes de implementação
- serve como referência para a construção
- documentar as decisões (de desenho) q foram feitas

## UM2 2: Unified Modeling Language

Benefícios > comunicação + clara e sucinta


> manter o desenho (planeamento) e a implementação (construção) coerentes

> mostrar/ocultar + níveis de detalhe

> Pode suportar processos de construção automática


## Diagramas de Atividades:

 initial node       activity final

 fork → divide em caminhos paralelos, concorrentes, à medida que um é executado o outro também é



 ação → Linhas de fluxo

 decisão → caminhos alternativos, condições de acesso entre [ ], ou 1 caminho ou outro

 merge → reúne as alternativas

 join → reúne os caminhos

 quando há subdiagramas associados a um elemento

 informação  
pins  ligação de dados em objetos

Eventos  saída       entrada



## Quando usar?

- \* Modelar fluxos de trabalho
- \* Descrever um algoritmo complexo
- \* Descrever a sequência de interações entre atores e o sistema ou especificação num caso de utilização

Pode ser usado para descrever processos organizacionais:

- \* Neutro em relação à programação
- \* Bom a captar papéis desempenhados
- \* Pode captar fluxo de dados

Zonas de responsabilidade / partições → servem para enquadrar as ações.

## Especificação de requisitos através de casos de utilização:

- Requisito
- capacidade pretendida do novo sistema
  - especificação do que deve ser implementado

## Caso de Utilização (CaU)

Sequência de ações que um sistema executa e que produzem um resultado com valor para algum ator

Implica: Foco no utilizador do sistema e nos episódios de uso

Foco na compreensão daquilo que os atores consideram um resultado de valor.

Capta quem (ator) faz o quê (interação) com o sistema, com que fim (objetivo).

Narrativas para contar como alguém usa o sistema.

## Diagrama de Casos de Utilização:

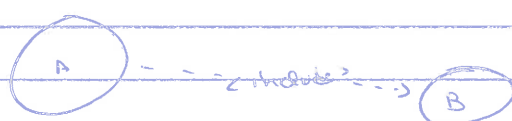
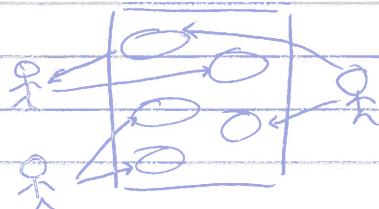
• Atores → intervenientes que interagem com o sistema e contribuem para a realização dos objetivos 

• Casos de Utilização → formas como o sistema vai ser usado.

conj. de cenários relacionados com o mesmo objeto.

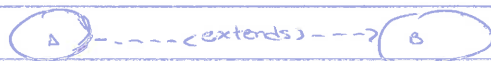
modelo de todas as maneiras úteis de usar o sistema.

→ permite saber rapidamente o âmbito do sistema



A inclui B

O cenário A incorpora o comportamento em B



A estende B

O comportamento de B pode incorporar o comportamento em A, dependendo da verificação de uma "condição de extensão".

→ O diagrama de atividades é adequado para apresentar os processos/ funções de um sistema ou de uma organização.

São usadas para mostrar os vários passos/etapas (Ações) na execução de um processo/operação (Atividade) e as transições que permitem o fluxo de execução. O DA admite caminhos alternativos que decorrem de uma decisão/reunião e caminhos concorrentes (Divisão/Sincronização). A atribuição da responsabilidade das tarefas a entidades é feita c/ portões. Também se pode representar a relação entre ações e objetos (mostrar que certas ações precisam de certos dados e produzem determinado resultado). Em alguns casos é útil mostrar a ocorrência de eventos.

→ O modelo de casos de utilização inclui atores, casos de utilização, narrativos e diagramas.

Um CaU é um episódio da interação c/ o sistema iniciado por um ator c/ um objetivo em mente, e é completado c/ sucesso quando este objetivo está satisfeito.

A especificação dos cenários é normalmente feita por níveis (de uma visão geral para versões detalhadas).

## Requisitos:

Requisitos têm de ser selecionados

Os atributos de qualidade são necessários para definir o produto.

Um sistema tem funcionalidades e atributos de qualidade

### Requisitos funcionais

- \* Capturam o comportamento pretendido do sistema
- \* Expresso como serviços, funções ou tarefas que o sistema deve realizar.
- \* Pode ser descrito c/ diagramas de comportamento: atividades, sequência.

### Requisitos não funcionais

- \* Restrições globais num sistema de software Ex. robustez, portabilidade
- \* Também designados como atributos de qualidade.

**F**uncionalidade

**U**sabilidade

**R**eliability (confiável)

**P**erformance

**S**upertável

**S** Requisitos devem ser Smart

**N** ão devem ser ambíguos

**A** têm de ser measurable / contáveis

**R** devem ser possíveis

**T** poderemos saber em que parte de implementação está e quem criou

## Situações de modelação c/ CAU:

Pode haver especialização entre atores.

### Ator primário

Solicita o sistema para resolver problemas / realizar objetivos

Os CAU são iniciados por um ator primário.

### Ator secundário

Fornece serviços ou informação por algum cenário do CAU

Podem ser sistemas externos

ou papéis de pessoas, que não são utilizadores.

CAU não mostram workflow!!!

## Modelo do Domínio com classes UML

**Desenvolvimento por objetos** → É uma estratégia para simplificar o espaço do problema, modularizando-o.

Esquema mental comum à análise (requisitos) e programação.  
Facilita reutilização de software.

Os objetos (em OO) modelam entidades do mundo real / espaço do problema.

- ★ Observáveis no mundo físico (Ex. Aluno, Avião)
- ★ Conceitos (Venda, Reserva)
- ★ Abstrações próprias do software (Lista Ligada, Vetor)

**Objeto** = propósito + estado + comportamento

representado através

↙ atributos e relacionamentos

podem mudar ao longo do tempo

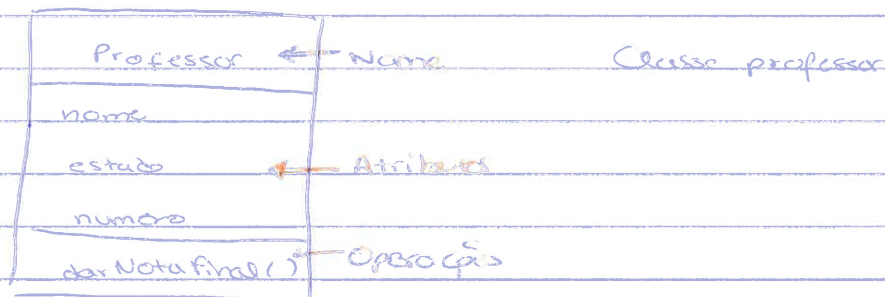
representado por

operações

Define como objeto age

**Classe** É uma categoria de objetos semelhantes, que partilham os mesmos atributos, operações, relacionamentos e semântica.

O obj. é uma instância de uma classe.



**Atributo**: Propriedade de uma classe que descreve a gama de valores que as instâncias podem deter.

Uma classe pode ter vários ou nenhum atributo.

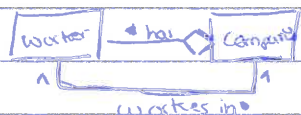
**Operação**: Implementação de um serviço / ação que pode ser pedida a qualquer objeto de uma classe.

O que a classe sabe fazer.



Multiplicidade: N<sup>o</sup> de instâncias de uma classe que relacionam com uma instância da outra.

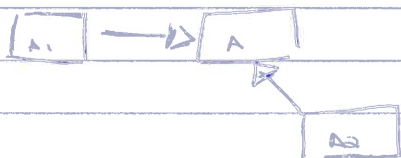
Agregação: "É parte de..."



Navegabilidade: Possibilidade de navegar de uma classe de partida para uma classe de chegada.



Generalização: Define uma hierarquia em que a subclasses herda das características da superclasse. "é um tipo de?"



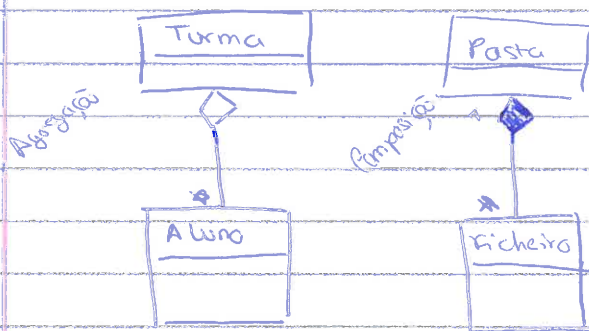
## Modelo de Domínio:

O Modelo de Domínio é um mapa para os objetos.

Mostra os conceitos de um problema.

Representado c/ um diagrama de classes (s, operações)

Boas Práticas: Uma classe representa um tipo de coisas. O nome é singular.  
O nome de classe é substantivo (representa um conceito).



Agregação: A detém partes de B.  
As instâncias B são partilháveis.

Composição: os B são partes constituintes de A.

Instâncias B não partilháveis.

Ligação fraca

Ligação forte

Dependência

Associação

Agregação

Composição

Generalização



→ O modelo de domínio é uma representação dos objetos da área do problema. Este mapa visual é construído utilizando o diagrama de classes da UML. As classes representam entidades naturais do Universo de discurso do problema e não classes de programação.

OpenUp:

Inception → Elaboration → Construction → Transition

O openUp segue Iterações do modelo de Processo Unificado (Unified Process) organizadas num conjunto de fases.

Cada fase termina com um milestone.

Inception

Finalidade: Alcançar acordo entre todos os interessados sobre os objetivos do projeto e fazer avançar / ou não a decisão.

Analisam-se os custos os os benefícios do projeto e decide-se se este avança ou se é cancelado.

Normalmente uma iteração curta; produz-se o documento de visão.

Elaboration

Finalidade: Reduzir os riscos técnicos e não técnicos. Os riscos técnicos são geralmente abordados estabelecendo a linha de base de uma arquitetura executável do sistema e fornecendo uma base estável para a maior parte do esforço de desenvolvimento na próxima fase.

Neste ponto, a "baseline" de requisitos é acordada, examinam-se os objetivos e o âmbito do sistema detalhadamente, escolhe-se a arquitetura e a resolução das principais risos.

O Milestone é concluído quando a arquitetura for finalizada.

### Construction

Finalidade: Desenvolver de forma econômica um produto c/ recursos completos (uma versão operacional do sistema) que possa ser implementado na comunidade.

Neste ponto o produto está pronto para ser entregue à equipe de transição. Todas as funcionalidades foram desenvolvidas e todos os testes concluídos. (teste alfa). O produto está pronto para testes beta.

### Transition

Finalidade: Garantir que o software está pronto para entregar ao cliente.


Neste ponto, decide-se se os objetivos foram concluídos e se se deve iniciar outro ciclo de desenvolvimento. O produto do Milestone é o resultado do cliente revisado e aceite (...)

## Diagramas de Sequência (Modelação do Comportamento)

Ilustra os objetos que participam no sistema e as mensagens que passam entre eles ao longo do tempo. Um diagrama de sequência é um modelo dinâmico que mostra a sequência explícita das mensagens transmitidas entre objetos em uma interação definida.

→ Comportamentos descritos pelas operações. Uma operação é uma ação que um objeto pode executar. Cada objeto pode enviar e receber mensagens. Mensagens são informações enviadas a objetos, para que estes executarem um dos seus comportamentos.

Mensagem = função / chamada de 1 objeto para outro

Actor: Pessoa ou sistema que beneficia com o sistema, é externo ao sistema ; no topo do diagrama

Objeto anObject: aClass participa na sequência, mandando e/ou recebendo mensagens; no topo do diagrama

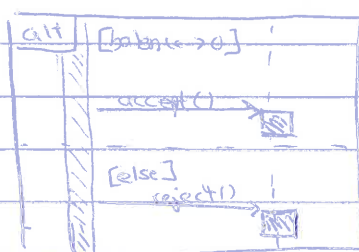
Life Line: denota a vida do objeto durante a sequência

Mensagem  Contém informação de um objeto para outro

Destruição de objeto: X

### Circuitos Alternativos:

```
If (balance > 0)
    otherObj.Accept()
Else
    otherObj.Reject()
```

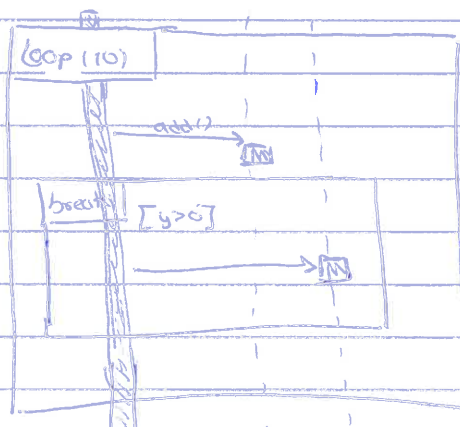


```
If (no errors)
    otherObj.PostComments()
```



### Loops:

```
i = 0;
While (i < 10)
{
    otherObj.Add()
    If (y > 0) break;
    i++;
}
```





## Diagrama de sequência: Benefícios:

- \* Mostra claramente a sequência / ordem temporal das mensagens
- \* Possibilidades de notação alongada

## Limitações:

- \* Cresce para a direita à medida que se acrescentam objetos

Mostram, para um cenário de um CAU:

- os eventos que os atores externos geram
- a sua ordem temporal
- as necessidades de integração entre sistemas

## Vistas de Arquitetura:

Decisões significativas sobre a organização de um sistema de software, seleção de elementos estruturais e ~~seus~~ interfaces pelas quais o sistema é composto, juntamente c/ o seu comportamento.

Organização estrutural de sistemas em grandes blocos

Uma arquitetura é definida para satisfazer os requisitos.

A arquitetura do sistema aborda 3's perspectivas de análise:

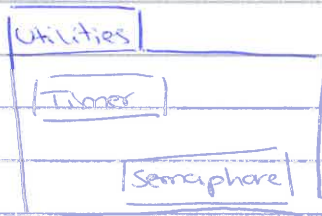
1 Arquitetura Lógica de Software → organização geral dos blocos de software; independente da tecnologia de implementação

2 Arquitetura de Componentes de Software → peças construídas com uma tecnologia concreta; construção modular

3 Arquitetura de Instalação → visão dos equipamentos e configuração de produção

## Arquitetura Lógica:

Packages podem representar agrupamentos muito  $\neq$ 's.



Associação entre pacotes

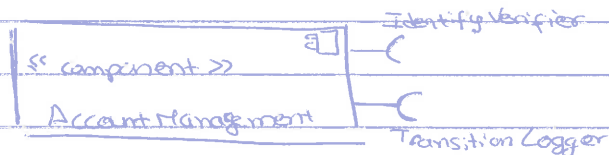


## Componente:

O componente é uma peça substituível, reusável de um sistema maior.

A funcionalidade de um componente é descrita por um conj. de interfaces fornecidas.

Para além de implementar, o componente pode requerer funcionalidades de outros.



Os componentes são implementados c/ tecnologia concreta.

Propriedades desejáveis: Encapsulamento

Reutilizável

Substituível

Solução modular c/ componentes:

Com os componentes pretende-se arquiteturas c/ baixo coupling.



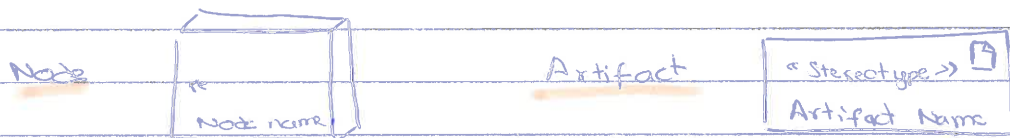
## A arquitetura "física" de instalação:

Explicita a configuração concreta do sistema, no ambiente de produção pretendido.

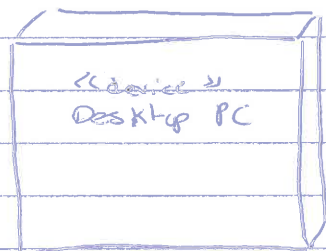
Nós (node) → um equipamento de hardware

Ambiente de execução → um ambiente externo à solução que proporciona o contexto necessário à sua execução

Artefactos (Artifact) → ficheiros concretos que são executados ou utilizados pela solução



Os artefactos são executados em nós



A relação "manifest" permite associar componentes a artefactos.

## Métodos ágeis de desenvolvimento

O desenvolvimento iterativo e evolutivo:

- baseia-se numa atitude de abraçar a mudança e a adaptação como fatores inevitáveis e essenciais

- isso não quer dizer que o desenvolvimento iterativo encoraje um processo cíclico e reativo

## Abordagem ágil

objetivo: Resposta rápida à alteração de condições.

Práticas que equilibram a disciplina e o feedback (ciclos curtos e entrega de valor frequente, integração em contínuo, desenvolvimento orientado por testes)

## Abordagem incremental e iterativa

### Papeis previstos no Scrum:

Scrum Team

Product owner

ScrumMaster

Development Team

Scrum process

(foto)

Scrum: backlog must be prioritized

Scrum velocity:  $\text{Estimated size} / \text{measured velocity} = \text{nº of Sprints}$

### Maquinas de Estado:

#### Notação básica dos Diagramas de Estado:

- Estado → caixas

- ↳ condições em que se encontra o objeto

- Transições → setas

- ↳ evolução de um estado para outro

- Triggers → etiquetas

- ↳ acontecimentos relevantes que causam transição de estado

- Condição de acesso

- Marcador início/fim



Se um objeto responder sempre da mesma maneira a um evento  $\Rightarrow$  é considerado independentemente do estado em relação a esse evento.

Se para todos os eventos de interesse, um objeto reage sempre da mesma maneira  $\Rightarrow$  é um obj. independente de estado.

Por outro lado obj. dependentes de estado reagem de maneira  $\neq$  a eventos, dependendo do seu estado.

## Garantia de Qualidade e os Métodos Ágeis de Desenvolvimento

**Verificação:** Estamos a fazer o sistema da maneira correta?

**Validação:** Estamos a fazer o sistema certo?

Test pyramid

