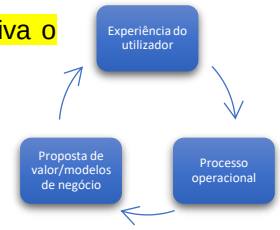
MAS

- Quando se fala em domínio tentamos descrever o contexto de aplicação de um projeto

Sistemas

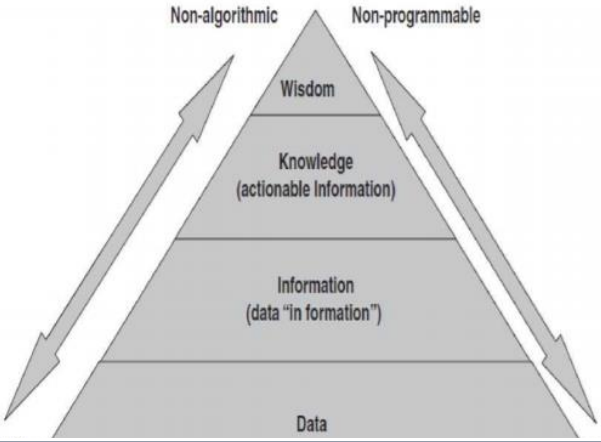
A utilização de TIC (Tecnologias de Informação e Comunicação) melhorou de forma decisiva o desempenho ou proposta de valor de uma empresa.



Dados vs. Informação

Os **dados** são os factos em cru, ou seja, as observações ou as medidas sobre a realidade.
O **conhecimento (do domínio)** explica os padrões que ocorrem nos dados. É a compreensão das relações entre os dados.
A **informação** é obtida no processamento de dados. É a coleção de factos organizados que apresentam mais valor, para além dos factos em si.

The DIKM Model



Deriva da formal ou informal análise da informação dos dados.

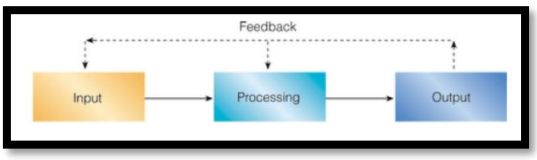
Refere-se à análise de dados que tem de ser coerente e organizada

Simples ponto de observação que carateriza a relação

Experiência do cliente/utente	Processos operacionais da organização	Modelo de negócio (produtos/ serviços de base tecnológica)
Estudar o cliente	Desmaterialização de processos	Extensão do negócio para o digital
Mais tecnologia no momento de venda /compra	Maior flexibilidade no posto de trabalho	Novo modelo de negócio de base tecnológica
Canais de contacto com o cliente	Gestão informada do desempenho/ indicadores	Globalização das operações e parcerias

O que é um sistema de informação?

Um **Sistema é** um conjunto de componentes/partes que interagem para atingir uma finalidade. Tem uma estrutura interna que transforma inputs em outputs, para um fim específico.
Ex: diversos sistemas no corpo humano



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

Exemplo de um sistema digestivo

Inputs -> comida; processo dos elementos -> boca; estômago; outputs -> nutrientes ready to use; Objetivo -> Absorção de nutrientes da comida.

Systems developments life cycle (SDLC)

É o processo de compreensão como um sistema de informação pode suportar as necessidades de negócio projetando um sistema, contruindo-o e entregando-o aos utilizadores (clientes).

Papel do **Analista** -> A chave do analista no SDLC é:

1. Analisar a situação de negócio;
2. Identifica as oportunidades de melhoria;
3. Projeta um sistema de informações para implementá-las;

O objetivo principal de um analista de sistemas não é criar um sistema maravilhoso, mas sim um **sistema organizado** !

SDLC 4 phases

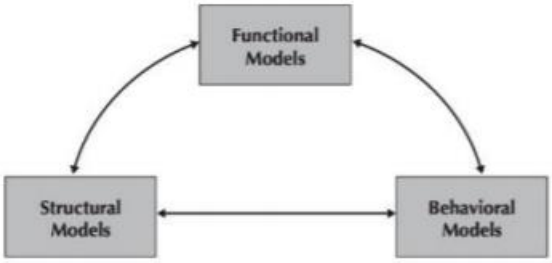
I. Planeamento/ Inicialização
   - ➔ Planeamento
     É o processo fundamental para compreender como **um sistema de informação deve ser construído** ("desenho") e determinar como a **equipa do projeto irá trabalhar** (distribuir tarefas).
   - ➔ Inicialização
     O **valor comercial do sistema** é identificado. A maioria das ideias para novos sistemas vem de fora da área de SI que serão apresentadas a um comité de aprovação de sistemas de informação.

II. Análise
   - ➔ Responde às questões sobre **quem** irá usar o sistema, o que o **sistema irá fazer** e **onde e quando** irá ser usado.
   - ➔ Key steps: Análise de sistemas existentes, recolha de requisitos e conceito de solução.

III. Design
   - ➔ Decide como o sistema irá operar em termos de hardware, software e infraestrutura da rede (sistema de arquitetura de design).
   - ➔ Decide a interface do utilizador, formulário e relatórios (data model design) e os programas específicos, bancos de dados (program design) e aquivos que serão utilizados (selection of frameworks).

IV. Implementação
   - ➔ O sistema é **construído** (ou comprado no caso de um design de software empacotado).
   - ➔ Inclui também a transação para o ambiente de **produção**.

Complemetary Views on a system



Functional -> O que o sistema faz? Comportamento externo de um sistema. (Análise)

Estrutual -> Quais são as partes do sistema? (Design)

Comportamento -> Quais são as operações que o sistema faz? (Análise) ???

O SDLC é concretizado em processos de desenvolvimento.

- ❖ Processo é uma sequência de tarefas que se inicia com uma conceção de uma ideia e termina na finalização, no nosso caso em particular, num *software* desenvolvido.
  Tem 4 fases: conceção, elaboração, construção e transição.

O que inclui um processo?

- ✓ Processos fundamentais;
- ✓ Papéis;
- ✓ Produtos de trabalho;
- ✓ Disciplinas;

- ✓ Ciclo de Vida.

Análise: O processo de caracterização do problema/ produto

Sintomas da indústria de engenharia de software

- Mau desempenho em condições de pico
- Equipas que não comunicam
- Dificuldades em controlar versões do software
- Necessidades sem respostas
- Requisitos que vão sendo alterados
- Módulos que não integram
- Dificuldades em manter e evoluir
- Erros revelados
- Utilizadores descontentes

Boas Práticas

- ✓ Desenvolver de forma iterativa
- ✓ Gestão explícita de requisitos
- ✓ Usar arquiteturas de componentes
- ✓ Usar modelos visuais
- ✓ Práticas de verificação contínua da qualidade
- ✓ Gestão explícita da mudança

Modelação: Linguagem visual de especificação

Os modelos ajudam a gerir a complexidade uma vez que ajudam a visualizar um sistema, como se pretende que venha a ser. Especifica a estrutura e o comportamento do sistema antes de ser implementado. Serve como referência para a construção ("planta"). Documentam as decisões (de desenho) que foram feitas.

Modelação visual ajuda do desenvolvimento

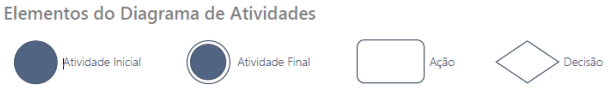
Linguagem de modelação normalizada: UML 2: Unified Modeling Language

Benefícios

A. Promove a comunicação mais clara e sucinta.
B. Mantém o desenho (planeamento)
C. Mostra ou esconde diferentes níveis de detalhe, conforme apropriado
D. Pode suportar, em parte, processos de construção automática (gera a solução a partir do modelo).

Um **processo de negócio** é um conjunto de atividades e tarefas que produzem um determinado serviço ou produto, por um objetivo particulare para um cliente ou mais.

Diagramas de Atividades -> mostram o fluxo de ações



Quando aplicar?

Aplicam-se na modelação de fluxos de trabalho/processos de negócio; na descrição de um algoritmo complexo e na descrição de uma sequência de interações entre atores e sistema.

Pode ser usado para descrever os processos organizacionais existentes/novos:

- Neutro em relação à programação;
- Bom a captar papéis;
- Pode captar o fluxo de dados também.

VANTAGENS:

- Pode ter muitas situações num só diagrama;
- Encoraja no pensamento de novas oportunidades em paralelo com as atividades;

DESVANTAGENS:

- O facto de ter muitas situações pode tornar o diagrama demasiado complexo para leitura.

Use Cases

Os atores interagem com o sistema para realizar objetivos

Caso de utilização (CaU) -> O que é ?

O **caso de utilização** capta quem (ator) faz o quê (interação) com o sistema, com que fim (objetivo).
- > Modo como o sistema será usado para os objetivos.
Descreve um diálogo entre o ator e o sistema. (Narrativa)

1. É uma sequência de ações que um sistema executa e que produz um resultado com valor para o ator em particular (incluindo as variantes relacionadas).
2. Foca a menor porção de atividade (interação) que produz um resultado relevante para o utilizador.
3. Conjunto de cenários relacionados com o mesmo objetivo. Um episódio de utilização mais as variantes.

Elementos do modelo de CaU

Ator: Qualquer entidade externa ao sistema sob especificação, que interage com o sistema.
Cenário: Uma situação/história particular de utilização do sistema, isto é, um caminho possível na execução de um caso de utilização.
Caso de utilização: Conjunto de cenários com o mesmo objetivo. Episódio de utilização + variantes.
Associações: Relacionamentos Atores/CaU, CaU/CaU, Atores/Atores.

Modelos de casos de utilização: fornece o contexto para a descoberta dos requisitos do sistema, ou seja, é um modelo de todas as maneiras úteis de usar um sistema.
Vantagens: Permite compreender rapidamente uma visão global sobre o que o sistema faz!

Para que serve o modelo de casos de utilização? -> Papel ?!

- ➔ Permite ver o comportamento de um sistema
- ➔ O modelo de casos de utilização divide a funcionalidade do sistema em episódios relevantes para os utilizadores.
- ➔ O modelo capta os requisitos do sistema (“O quê?”), não implementando a solução (“como?”).
- ➔ A UML fornece uma visualização e uma descrição dos cenários.

Narrativa

- ❖ Descrever como é que o CaU começa e acaba.
- ❖ Descrever o fluxo de eventos que provocam as ações/ reações. Descrever apenas o cenário “dentro” deste CaU. Não incluir fluxo de outros, nem comportamento externo ao sistema.
- ❖ Clarificar a troca de informação entre Atores e Sistema.
- ❖ Evitar designação vagas, ambíguas.
- ❖ Verificação de qualidade.

Fluxo do CaU

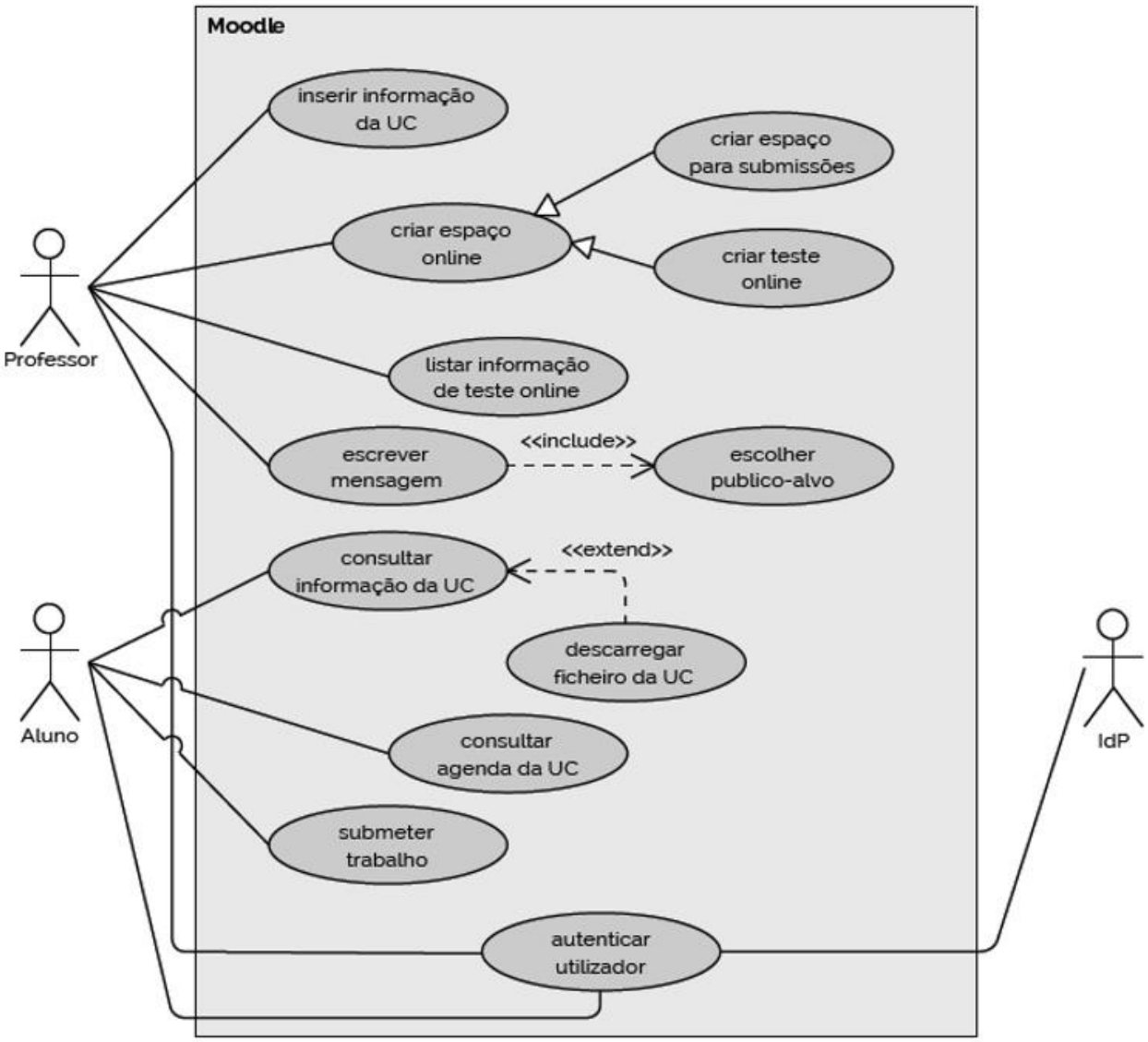
Fluxo-tipo: Guião para o diálogo ator/sistema.
Fluxo-Alternativo: Significam variações “normais”, casos particulares/incomuns, situações de erro e respetivo tratamento.

Como encontrar os CaU?

1. Identificar a fronteira do sistema.
2. Identificar os atores que, de alguma forma, interagem com o sistema.
3. Para cada ator, identificar os objetivos/motivações para usar o sistema.
4. Definir os CaU que satisfazem os objetivos dos atores, dar nomes que refletem a motivação do ator.

Use case:	Brief description:
Create new assigment	The Teaching Staff creates a new Activity of type Assignment, directly inserting it in the page layout. The assignment must define a title and a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from registered students.

Use case:	Add new assigment
Brief description:	The Faculty creates assignments for students, directly inserting it in the course page. The assignment defines a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from students.
Basic flow:	1. Log-in using corporate IdP. 2. Select desired course. 3. Turn editing mode on. 4. Add Assignment activity in the page layout. 5. Configure Assignment activity. 6. Commit changes.
Alternative flows:	Step 1: IdP unavailable. Step 4/5: Instead of a new, empty assignment, the user may reuse an existing one.
Open issues:	Step 3/4. The course is closed. Are changes allowed to past courses? Step 5. The browser does not accept the rich text editor. Default to plain text?



Requisitos -> O sistema deve ? (permitir um pedido de adesão…)

O que é um requisito? Uma declaração do que o sistema deve fazer ou uma característica que deve ter.

Propósito da determinação de requisitos

Converter requisitos de negócios de alto nível em requisitos detalhados que podem ser usados como entradas para a criação de modelos (do sistema).

Atributos de Qualidade

Requisitos que têm de ser selecionados:

- Requisitos essenciais vs facultativos
- Capacidade vs Orçamento vs Tempo
- Enquadramento regulamentar

Os atributos de qualidade são necessários para definir o produto. Um sistema tem funcionalidade e atributos de qualidade:

Requisitos Funcionais

- Captam o comportamento pretendido do sistema. São expressos como serviços, funções ou tarefas que o sistema deve realizar.
- Pode ser captado nos CaU.
- Pode ser descrito com diagramas de comportamento: atividades, sequência.

Requisitos Não Funcionais

- Restrições globais num sistema de software (robustez).
- Também se designam atributos de qualidade.
- Por regra, não afetam apenas um módulo/CaU.

Outros requisitos

FURPS+

Funcionalidade -> capacidade lógicas que um sistema oferece ao cliente. ex: segurança
Usabilidade -> entrega de capacidades documentais ao cliente final.
Reliability/ fiabilidade -> disponibilidade/ tempo que um sistema está ativo.
Performance -> tempo de resposta, tempo de recuperação, temp de inicio de ligação e de encerramento de sessão.
Suporte -> teste, adaptação, compatibilidade, manutenção

A apresentação de requisitos segue um modelo: Norma **ISO/IEC/IEEE 29148:2011**: Systems and software engineering -- Life cycle processes -- Requirements engineering.

CaU contam histórias que mostram os requisitos funcionais em contexto.

CaU Essencias vs Concretos: Estilo essencial para manter as referências à interface com o utilizador de parte e focar na intenção do ator.

Modelo de processo (Open) Unified Process

É necessário um processo de trabalho:
quem faz o **quê**;
quando é que alguma coisa deve ser construída;
e **como** atingir certos objetivos.

Os **requisitos novos/modificados** passam por um processo de **Engenharia de Software** para obterem um **sistema novo**.
O **processo** de engenharia de software é o conjunto das atividades e resultados associados que produzem uma peça de software.

Atividades fundamentais de um processo de Engenharia de Software:

A Especificação de requisito é caracterizada pela definição do problema/produto, pelo estado atual, pelo âmbito, pela visão da solução pretendida e pelo levantamento de requisitos/capacidades.

A **Especificação** do software (ou seja o desenho) é a definição do software a construir e as restrições de operação. Caracteriza-se pelos modelos de solução, apresenta uma estrutura de dados, apresenta também uma arquitetura/organização da solução e interfaces entre sub-sistemas/integrações com sistema externos.

O **Desenvolvimento** do software integra a implementação: escrever o código, desenvolver a base dados e procurar as oportunidades de reutilização.

A **Validação** e garantia de qualidade do software pretende garantir que o software satisfaça as expectativas do cliente. É necessário verificar a correção (realizar testes de módulo/unidade, testes de integração de módulos e testes do sistema num ambiente alvo) e validar a adequação (aceitação por parte do utilizador).

A **Instalação/operação** necessita de uma instalação de software e hardware, uma formação e uma entrada em produção.

A **Evolução/manutenção** do software inicializa uma adaptação e modificação do software para responder à alteração de requisitos do cliente e do ambiente. A **manutenção corretiva** corrige os erros no código, a **manutenção adaptativa** adapta o software às novas condições de operação e a **manutenção evolutiva** melhora/expande o produto.

Os modelos de Processo de Engenharia de Software são descrições abstratas de um processo de Engenharia de software segundo uma interpretação Indicam atividades e produtos a realizar e os papéis na equipa.

Waterfall

Aspetos chave como o resultado de cada fase, 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.

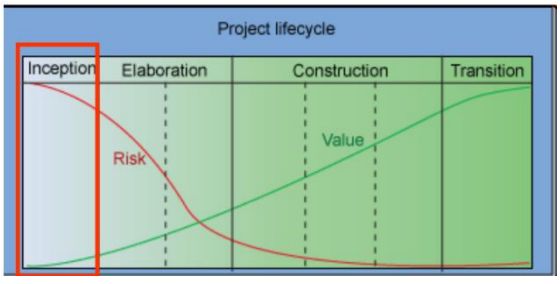
Vantagens: abordagem simples, passos simples, o facto de ser disciplinado. O fim de cada etapa que progride linearmente.

Abordagem "ágil"
Objetivo: Desenvolver software rapidamente para responder às alterações rápidas dos requisitos.

Para atingir a agilidade é necessário: práticas que equilibrem a disciplina e o feedback e aplicar princípios de desenho que favoreça a construção de software

Fazes do projeto

Inception: Saber o **que** contruir
Apresenta uma iteração curta, por norma. É necessário produzir um documento de visão, um caso de negócio e desenvolver e focar-se apenas no que realmente interessa. Desenvolver os requisitos do projeto a alto nível: iniciar os casos de uso (e, é opcional, os modelos de domínio).



Elaboration: Saber **como** contruir
Apresenta várias ou longas iterações. Equilibra, faz referência às principais questões técnicas e riscos de negócios. Fase de testes. Produz e valida a arquitetura executável, é importante definir pontos-chave e implementar as interfaces cuidadosamente.

Construction: **Contrução** do produto
É necessário definir uma estratégia e o código da implementação do produto. Ao realizar vários testes, o objetivo é verificar a evolução da arquitetura.

Transition: A transição, a fase de **estabilizar** e **implementar** o produto Constante atualização das versões do projeto

Análise por objetos: Objetos e Classes

Desenvolvimento por objetos

É uma estratégia para simplificar o espaço do problema, modularizando-o. É um esquema mental comum à análise de requisitos e programação. Facilita à reutilização de software.

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

Observáveis no mundo físico Exemplo: Aluno, Avião.
Conceitos Exemplo: Venda, Reserva.
Abstrações próprias do software Exemplo: Lista ligada, Vetor.

Objetos = **propósito** + **estado** + **comportamento**

Um objeto é uma entidade com uma fronteira bem definida, que encapsula **estado** e **comportamento**. Cada objeto tem a sua **própria identidade**, única, mesmo que o seu estado seja igual ao de outro objeto.

O estado é representado através de atributos e relacionamentos. O estado de um objeto corresponde a uma das condições/ configurações a que é possível o objeto apresentar-se. É normal o estado do objeto mudar ao longo do tempo.

O comportamento é representado através de operações. O comportamento define como é que o objeto age/reage. O comportamento visível/exposto é modelado pelo conjunto de mensagens a que responde (operações).

Objetos = **estado** + **operações**

Sabe os seus atributos e os seus relacionamentos. Sabe o seu comportamento, ativado através de operações.

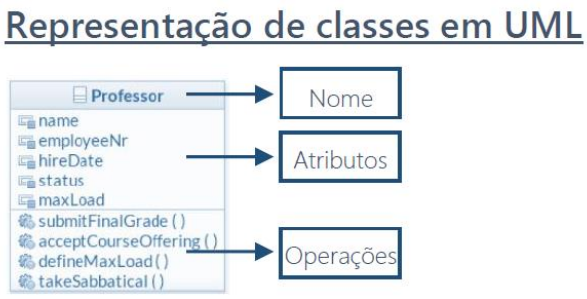
O objeto é uma instância (ocorrência) de uma classe

O que é uma classe?

Uma classe é uma categoria de objetos semelhantes, que partilham os mesmos atributos, operações, relacionamentos e semântica. É uma **abstração**, categoriza objetos semelhantes e enfatiza as características de interesse (e suprime outras).

A relação entre classes e objetos

Uma classe é uma definição abstrata de um objeto, define a estrutura e comportamento de cada objeto daquela classe/categoria. Funciona como um molde para criar objetos. **As classes não são coleções de objetos**.



A relação entre classes e objetos

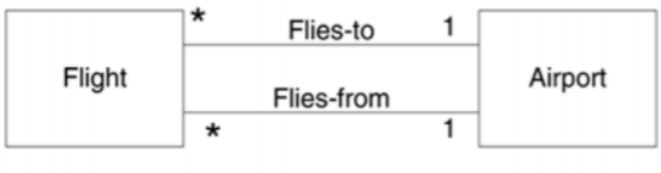
Uma classe é uma definição abstrata de um objeto, define a estrutura e comportamento de cada objeto daquela classe/categoria. Funciona como um molde para criar objetos. **As classes não são coleções de objetos.**

Um **atributo** é uma propriedade de uma classe que descreve a gama de valores que as instâncias podem deter. Uma classe pode ter vários ou nenhum atributo.

Uma **operação** é a implementação de um serviço/ação que pode ser pedida a qualquer objeto de uma classe. É o que a classe sabe fazer. Uma classe pode ter várias ou nenhuma operação. Comuns: comandos e interrogações.

Uma **associação** é a relação semântica entre dois ou mais classificadores que especifica as ligações existentes entre as suas instâncias. Uma relação estrutura que mostra que um tipo de objetos estão ligados a outro tipo de objetos.

Multiplicidade é o número de instâncias de uma classe que se relacionam com uma instância da outra. Para cada associação, avaliar cada um dos extremos.



Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional scalar role)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

Agregação é uma forma especial de associação que modela uma relação de todo-parte, entre o agregador e as suas partes constituintes. Pode ler-se "**É parte de**...". A multiplicidade é usada como em outras associações.

Negavilidade indica a possibilidade de navegar de uma classe de partida para uma classe de chegada, usando a associação.

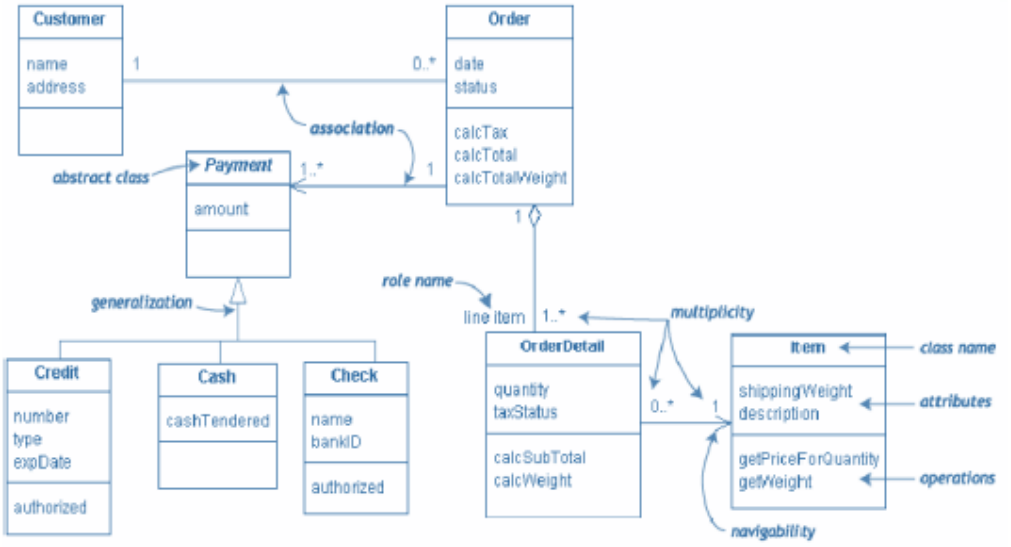
A relação entre classes em que uma especializa a estrutura e/ou comportamento de outra, partilhando todas as características designa-se generalização. Define uma hierarquia em que a subclasse herda das características da superclass (A subclasse pode sempre ser usada onde a superclasse é usada, mas não ao contrário). Pode ler-se "**é um tipo de**".

É relação entre classes em que uma especializa a estrutura e/ou comportamento de outra, partilhando todas as características Define uma hierarquia em que a subclass herda das caracterísitcas da superclass A subclasse pode sempre ser usada onde a superclasse é usada, mas não ao contrório . Pode ler -se "é um tipo de"

A subclasse herda os atributos, operações e relacionamentos da superclasse. A subclasse pode adicionar mais atributos, operações e relacionamentos à base herdada. Redefine as operações da superclasse. A herança põe em evidência as características comuns entre classes.

AQUI

Síntese da notação do diagrama de classes



O modelo do domínio é um mapa para os objetos:

Mostra os conceitos de um problema (**dicionário visual**). Tem uma perspetiva do cartógrafo, mostrar que existe usando os nomes que a população utiliza. Não é um software.

O modelo de domínio em UML é representado com um diagrama de classes sem operações: não tem implementação. Pode representar objetos/conceitos do domínio, associações entre esses conceitos e atributos.

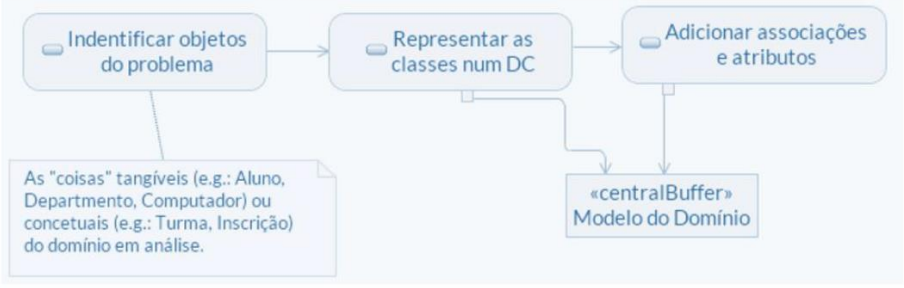
Classes de análise (Assunto de MAS)

- Resultado da análise dos requisitos (analista)
- Neutro em relação à implementação
- Não fornece diretamente o modelo de dados nem classes de programação

Classes em Java (Assunto de POO (e, em parte, de MAS))

- Resultado do desenho/implementação (programador)
- Escritas numa linguagem concreta (OO)

Fluxo para criar o Modelo de Domínio



É possível encontrar as classes: Procurando numa lista de situações comuns, explorando documentos/relatórios existentes na área do problema e analisando os nomes.

UML tipos de diagrama.

Diagramas estruturais.

-> Diagrama de classes;

-> Diagrama de Objetos;

Associação entre conceitos

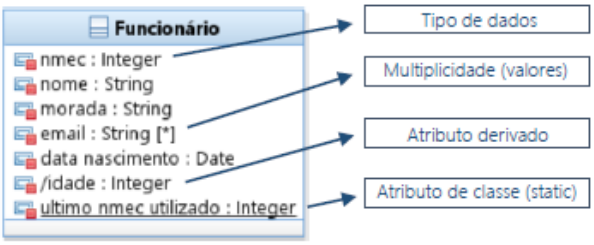
- **Descoberta**
- Requisitos na forma [entidade][verbo]{entidade]
- Formulários/relatórios da área do problema ligam informação de várias entidades
- As anotações também são um elemento de modelação

- **Práticas**
- A generalidade das associações do Modelo de Domínio são binárias
- As associações devem ter um nome para clarificar a interpretação
- Por convenção, lê-se de cima para baixo, da esquerda para a direita

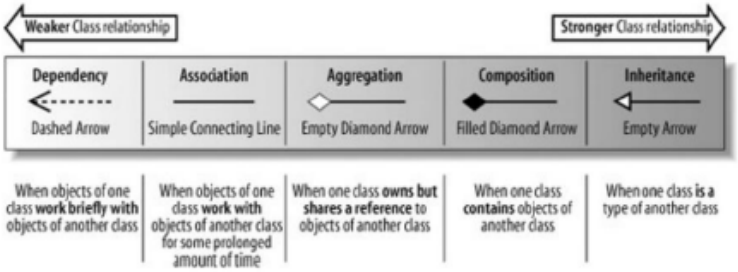
Conceitos:

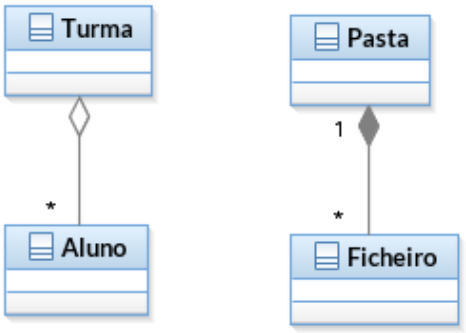
- Atributos de classe são partilhados por todas as instâncias.
- Associação reflexivas relacionam instâncias da mesma classe.
- Classes-associação captam a informação que descreve o relacionamento.
- Indicações para o uso de uma classe-associação no modelo de domínio
    - ✓ Um atributo está relacionado com (a ocorrência de) uma associação.
    - ✓ As instâncias da classe-associação têm um tempo de vida dependente da associação.
    - ✓ Há uma relação de N:M entre dois conceitos e informação que carateriza a própria associação.
- Uma classe abstrata não é instanciada diretamente.
- Classes abstratas facilitam implementações parciais (partes comuns na superclasse).
- No conceito de programação as interfaces são contratos sem implementação e sem estado..
- Modelo de domínio não é o modelo de uma base de dados.

Especificação dos atributos



Cinco tipos diferentes de relacionamento entre classes com diferentes níveis de ligação



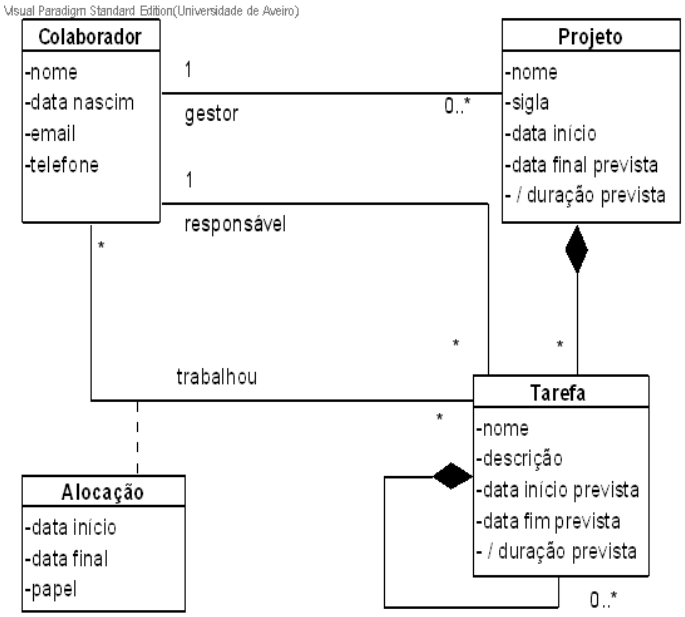


Agregação: A detém partes B de forma não -exclusiva

As instâncias B partilháveis

Composição: os B são partes constituintes de A

Instâncias B não partilháveis



16.

Considere o Diagrama 3:

a) Um Projeto pode ou não ter um gestor.
b) Um Projeto pode agregar sub-projetos.
c) Uma Tarefa pode agregar sub-tarefas.
d) Uma Tarefa pode ser realizada em diferentes projetos.
e) Cada Colaborador é responsável por uma Tarefa.

17.

O Diagrama 3 utiliza uma classe de associação, para caraterizar a alocação a projetos.

a) A classe de associação é uma classe cujas características descrevem uma associação, e não um objeto "normal".
b) A classe de associação facilita a visualização da interdependência entre duas classes.
c) A classe de associação pode ser suprimida, desde que se mova os respetivos atributos para uma das classes associadas.
d) A classe de associação deve ser usada sempre que há uma multiplicidade de muitos para muitos (entre as classes base associadas).
e) Um bom modelo deve evitar a utilização de classes de associação.

18.

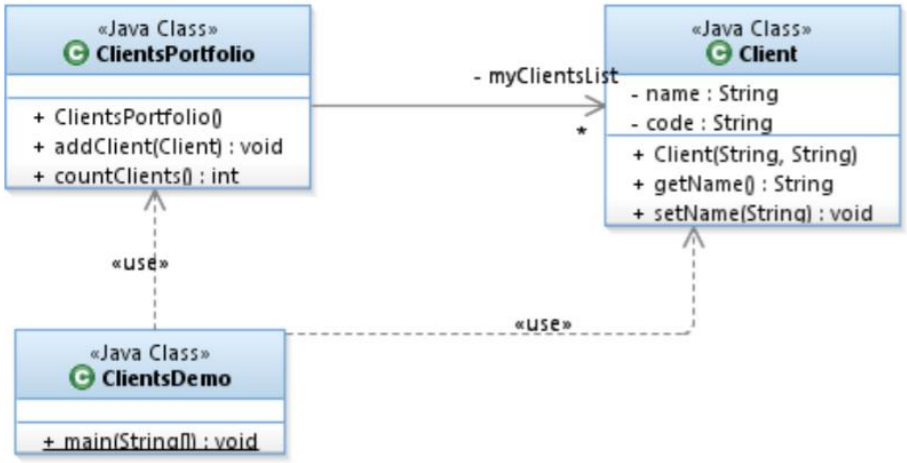
Relativamente ao Diagrama 3, Tarefa e Projeto indicam durações previstas.

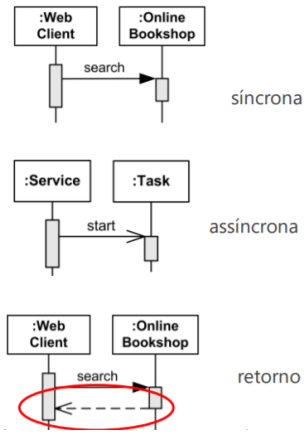
a) Há um erro de notação no atributo associado à duração prevista.
b) A duração prevista pode ser determinada à custa de outros atributos, não deve ser representada na classe.
c) A duração prevista pode ser determinada à custa de outros atributos, é um atributo derivado.
d) A duração prevista deveria ser definida apenas na Tarefa.
e) A duração prevista é obtida por um método e é errado apresentar como um atributo.

Ver powerpoint ! tem exemplos.

```
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();
   }
}
```



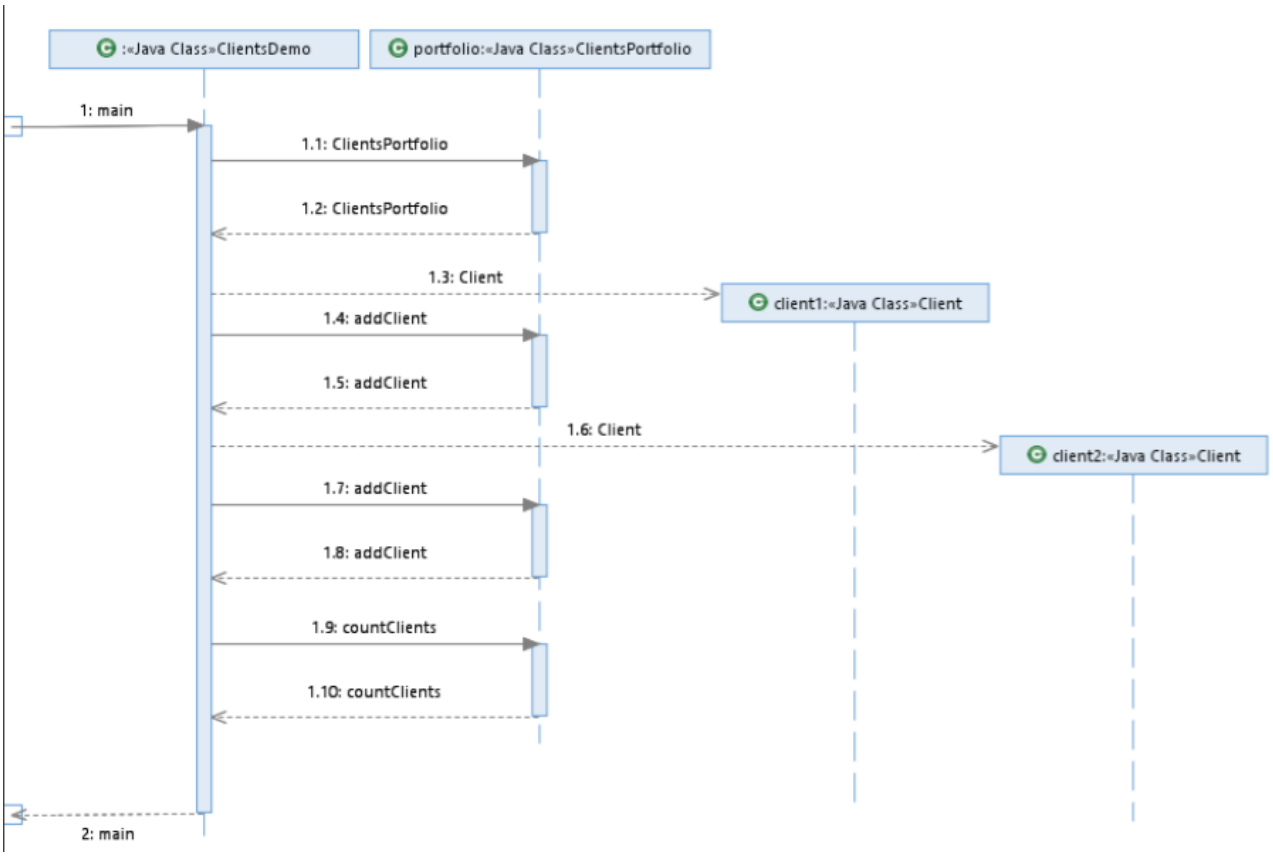



```
public class ClientsDemo {
public static void main(String[] args) {
   ClientsPortfolio portfolio = new ClientsPortfolio();

   Client client1= new Client( "C101", "Logistica
   Tartaruga");
   portfolio.addClient( client1 );

   Client client2 = new Client( "C104", "Jose, Maria &
   Jesus Lda");
   portfolio.addClient( client2 );

   System.out.println( "Clients count: " +
   portfolio.countClients() );
}
}
```



Para representar o actor primário (ou os actores) e o sistema usamos uma *lifeline*.
Os diagramas de sequência de sistema (DSS) **mostram operações de Sistema** para cada CaU. O sistema é modelado por uma classe que o representa globalmente. Opcionalmente, pode-se incluir o texto da descrição do CaU.

Uma **operação de sistema** (OpS) corresponde a um ponto de entrada no sistema, encapsula um conjunto de interações subjacentes entre objetos, necessárias para realizar essa operação.

A etapa seguinte na construção do **modelo dinâmico**: mostra, para cada OpS, a rede de objetos que vai ser ativada. Estes objetos são instâncias de classes que devemos identificar.
Intitulamos de atividade o que designamos por desenho, quando escolhemos as classes que participam na solução. Os diagramas de interação ajudam a distribuir responsabilidades, ou seja, a encontrar métodos.

Ponto essencial: distribuir responsabilidades pelos objetos, seguindo os princípios do desenho por objectos (*Object-Oriented Design*).

Responsabilidades de um objeto

Fazer	**Saber**
• Sobre o estado, como calcular alguma coisa, criar objetos.	• Conhecer o estado interno (escondido)
• Iniciar uma ação noutros objetos. • Coordenar/controlar as ações de outros objetos	• Conhecer os objetos relacionados

UML para programar

- *Forward engineering*: Do modelo para a implementação (código).
- *Reverse engineering*: Da implementação (código) para o modelo.
- *Round-trip engineering*: Transição transparente nos dois sentidos.

Coupling	**Coesão**
Mede a força/intensidade da dependência de uma classe de outras	Mede a força/intensidade do relacionamento dos elementos de uma classe entre si.
A classe C1 está emparelhada com C2 se precisa de C2, direta ou indiretamente.	Todas as operações e dados de uma classe devem estar natural e diretamente relacionados com o conceito que a classe modela
Uma classe que depende de outras 2 tem um "coupling" mais baixo que uma que dependa de 8.	Uma classe deve ter um foco único (vs. responsabilidades desgarradas)

12. Vistas de Arquitetura

A arquitetura do sistema aborda diferentes perpetivas de análise

É a primeira transportadora da qualidade do sistema. Transporta a permormance, a modificabilidade e a segurança. Pode ser alcançada sem uma visão arquitecional unificadora.

Arquiteto	Desenvolve o desenho, a conceção e desenvolvimento da arquitetura.
	Responsável pelas necessidades do pessoal (envolvido ou afetado por qualquer mudança no sistema de informação);
	Faz uma análise de alto nível nos requisitos do sistema, baseada nas necessidades dos utilizadores.
	Garante que foi desenvolvida uma arquitetura robusta.
	Um bom arquiteto traduz os requisitos de forma que sejam inteligíveis para clientes, utilizadores e desenvolvedores.

Arquitetura do sistema	**Organização estrutural do sistema em grandes blocos.** Componentes desenvolvidos e/ou integrados ex:servidores, rede,...
	Uma arquitetura é definida para satisfazer os requisitos. Requisitos não funcionais e restrições de operação são determinantes
	Compromissos e decisões. (fornece uma base para gerenciamento de projeto)

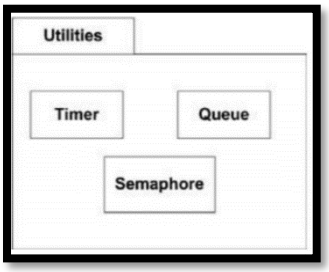
Uma arquitetura pode ser **estática** (aspetos estáticos de uma sistema não mudam como um sistema corre) **e dinâmica** (mudam como um sistema corre)

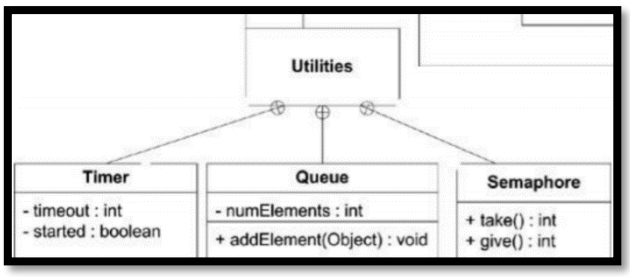
Prespetivas de análise

Arquitetura lógica: organização geral dos blocos de software (packages), independente da tecnologia de implementação.

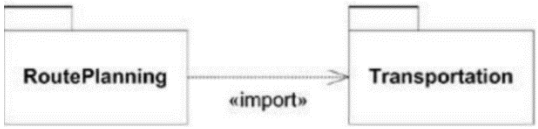
Arquitetura de instalação: visão dos equipamentos e configuração concreta de produção (conectividade, distribuição).

Arquitetura de componentes: peças construídas com uma tecnologia concreta. É normal dividir sistemas complexos em subsistemas mais geríveis. O componente é uma peça **substituível**, **reusável de um sistema maior**, cujos **detalhes** de implementação **são abstraídos**. A funcionalidade de um componente é descrita por um conjunto de interfaces fornecidos. Para além de implementar, o componente pode requerer funcionalidades de outros.

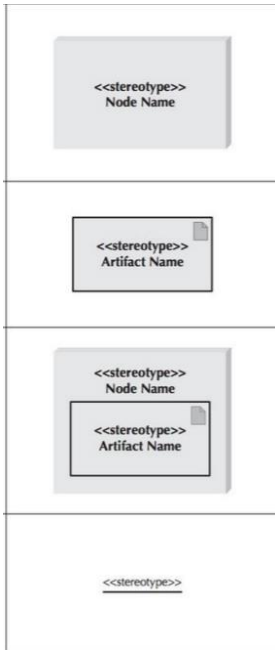




classes contidas num pacote



associação entre pacotes -> import



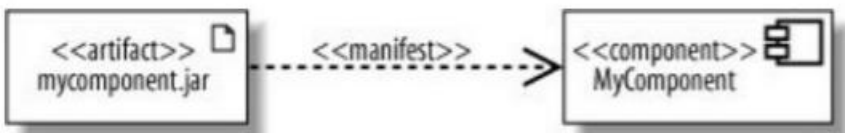
Com os componentes, pretende-se uma arquitetura com baixo "coupling".
A exposição da funcionalidade através de interfaces ajuda a separar o contrato da implementação.

Diagrama de instalação da UML

Nós (*node*): "caixa" Um equipamento de hardware, server, dispositivo de network

Ambiente de execução: Um ambiente externo à solução que proporciona o contexto necessário à sua execução.
ex: SO, servidor web, servidor aplicacional, framework

Artefactos (*artifact*): Ficheiros concretos que são executados ou utilizados pela solução.



A relação “manifest” permite associar componentes a artefactos.