

# Introdução ao UML

Metodologias e Desenvolvimento de Software  
2022/2023

Pedro Salgueiro  
CLAV-256  
pds@uevora.pt

# UML

O que é?

- Unified Modeling Language (UML)
- Linguagem de modelação gráfica
  - Standard
- Conjunto de notações gráficas
- Meta-modelo único
- Descrever e desenhar (ou modelar) sistemas de software
  - Object-oriented
  - Mas não só

# Tipos de utilização

- Depende do utilizador
- **Sketch**
  - Esboços
- **Blueprint**
  - Detalhar um sistema, ou partes
- **Linguagem de programação**
  - Implementar um sistema, ou partes
- Conceptual and software modeling

# UML as Sketch

- Descrever/comunicar detalhes de um sistema
- Nível de abstração elevado
- Tipo de utilização
  - Forward engineering
  - Reverse engineering

# UML as Sketch

## Forward engineering

- Desenhar um diagrama antes de escrever código
- Descrever/discutir ideias e alternativas com a equipa
- Focar no que é importante
- Não pensar no código

# UML as Sketch

## Reverse engineering

- Desenhar um diagrama depois de termos código
- Usar sketches para explicar parte do sistema
- Documentação do sistema
  - Complemento

## Ferramentas

- Simples
- Criar diagramas UML
- informal

# UML as blueprint

Detalhar um sistema

- De forma exaustiva
- Tipo de utilização
  - Forward engineering
  - Reverse engineering

# UML as blueprint

## Forward engineering

- Modelo detalhado
  - sistema
  - partes do sistema
- “Traduzido/programado” pelo programador
- Completo
- especificação de todas as decisões
  - Programador “limita-se” a seguir o modelo
- Abordagem comum:
  - Designer → modelo do interface de subsistemas
  - Programador → detalhes internos de cada subsistema



# UML as blueprint

## Reverse engineering

- Informação detalhada de parte do código de um sistema
  - ex: detalhes de uma classe num modelo gráfico

# UML as blueprint

## Ferramentas

- Mais complexas
- Forward engineering
  - Criar diagramas UML
- Reverse engineering
  - Analisam o código fonte
  - Geram diagramas UML

# UML as programming language

- Maior quantidade de modelos UML
  - Programação cada vez mais mecânica
  - UML para programar o sistema
- Programadores
  - Usam diagramas UML
  - UML é o source code
  - Compilados diretamente
- Ferramentas muito mais complexas e sofisticadas

# UML as programming language

## Model Driven Architecture (MDA)

- Abordagem standard para usar UML
  - UML as a programming language
- Confunde-se com UML
  - “Apenas” usa UML como linguagem base dos modelos

# UML as programming language

## Abordagem MDA

- Trabalho dividido em duas etapas
  1. Platform Independent Model (PIM)
    - Modelo/representação do sistema/aplicação
    - Independente da plataforma/tecnologia
    - UML
  2. Platform Specific Model (PSM)
    - Modelo/representação do sistema/aplicação
    - Dependente da plataforma
    - Um para cada plataforma/tecnologia
    - Pode ser UML
- Modelo PIM transformado em PSM
  - Ferramentas específicas
- Modelo PSM transformado em código
  - Pode ser automatizado

# UML as programming language

## Executable UML

- Parecido com MDA
  - Modelos independentes da plataforma
    - Parecidos com os modelos PIM de MDA
  - Model Compiler
    - Compila o modelo inicial
    - Sistemas executáveis
    - Num único passo
    - Não necessita dos *Platform Specific Models (PSM)*
- Subset do UML
  - Não usa todas as características do UML
  - Mais simples do que UML

# UML as programming language

- Realista?
  - Parece ser bom demais para ser verdade
- Problemas
  - Ferramentas
    - Maturidade suficiente?
- Produtividade
  - Executable UML vs outra linguagem de programação

# Conceptual and software modeling

## Perspetiva de software

- Elementos UML
  - Mapeados diretamente para elementos de software
  - Exemplo:
    - Ferramentas que geram UML a partir de código
    - Software mapeado diretamente para UML

## Perspetiva conceptual

- Elementos UML
  - Descrição de conceitos de um domínio de aplicação
  - Não estamos a falar de elementos de software
  - Estamos criar um “vocabulário” para falar de um domínio específico
  - Exemplo: Usar os diagramas UML para perceber o significado dos termos e conceitos associado ao problema



# Notações e meta-modelos

UML define:

## 1. Notações

- Elementos gráficos dos modelos
- Sintaxe gráfica da linguagem de modelação
  - Representação de cada conceito: classes, relações, multiplicidade, etc...
- Problemas
  - O que significa uma classe, uma relação ou multiplicidade?

## 2. Meta-modelo

- Define os conceitos da linguagem
  - Como deve ser usada

# Notações e meta-modelos

## Linguagens gráficas

- Pouco rigor
  - Notação depende da “intuição” em vez da definição formal
- Embora muito informal
  - Continua a ser muito útil

## Importância do meta-modelo

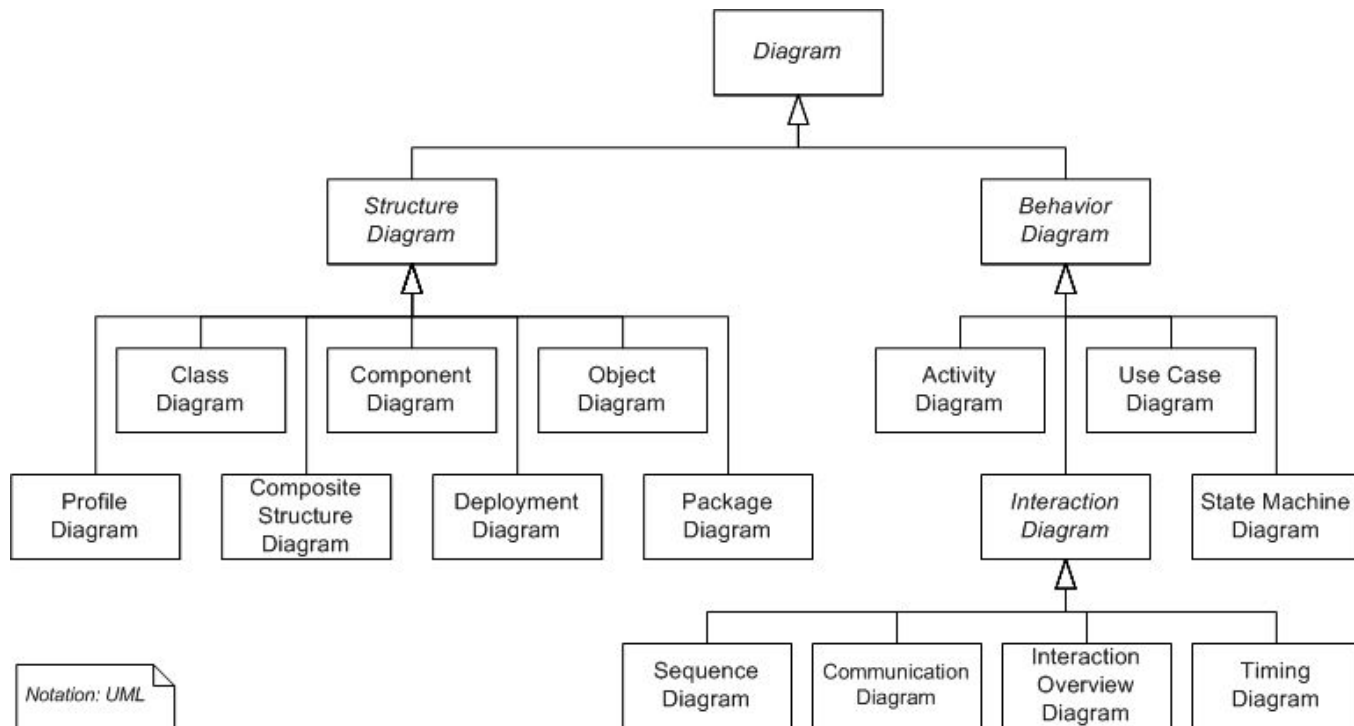
- Depende da utilização
- Sketching
  - Apenas notação
- Blueprinting
  - Notação
  - Um pouco de meta-modelo
- UML as programming language
  - Notação
  - Muito meta-modelo

# Diagramas UML

UML V2 define 13 diagramas

- **Activity**
- **Class**
- Communication
- Component
- Composite structure
- Deployment
- Interaction overview
- **Object**
- Package
- **Sequence**
- **State machine**
- Timing
- **Use case**

# Diagramas UML



# UML válido

O que é?

- Resposta simples:
  - “O que está definido como bem formado na sua especificação”

Na prática

- Não é tão simples
- Standard
  - Muito complexo
  - Está aberto a várias interpretações
- Pode ter diferentes interpretações
  - Tem diferentes utilizações
- Depende da utilização

# UML válido

## Importância

- *Sketching ou blueprinting*
  - Pode usar-se
  - Mas é pouco importante
  - **Mais importante** → **bom design**
- UML as a programming language
  - Essencial
  - Ou o sistema não funcionará corretamente

# Significado do UML

- Existe uma especificação detalhada
  - UML bem formado ou correto
- Significado
  - Não existe especificação para o seu significado
- Diagrama UML
  - Não existe tradução exata para source code
  - Consegue-se ter uma “ideia geral” de como seria o código
  - Suficiente
  - Detalhes de implementação → responsabilidade da equipa de desenvolvimento

# Será suficiente?

Grande conjunto de diagramas

- Modelar diferentes aspetos de um sistema
- Conseguem definir um sistema de uma forma muito completa
- Conjunto incompleto
  - Recorrer a outros tipos de diagramas



# Começar por onde?

- Grande conjunto de diagramas
  - Apenas um pequeno subset é usado
  - Raramente são usados todos diagramas
- Que diagramas?
  - utilização
  - sistema/aplicação
- Diagramas mais usados
  - classes, sequências, atividades, use cases, objetos, transição de estados

# UML e processos de desenvolvimento

## Análise de requisitos

- Use cases:
  - Descrevem como é que os utilizadores interagem com o sistema
- Diagramas de classes:
  - Usando uma perspetiva conceptual, podem ser usados para construir um vocabulário sobre o domínio do sistema
- Diagramas de atividades:
  - Workflow/fluxo de trabalho na empresa, mostrando como é que o software interage com atividades humanas. Mostrar o contexto dos use cases, bem como detalhes de use cases complexos
- Diagramas de estados:
  - Se o sistema tiver um life cycle interessante, com diferentes estados e eventos que fazem mudar o estado
- Nunca incluir nada técnico!

# UML e processos de desenvolvimento

## Design

- Diagramas de classes
  - A partir de uma perspetiva do software, podem mostrar classes do sistema e como estão interligadas
- Diagramas de sequências:
  - Workflow/fluxo de trabalho na empresa, mostrando como é que o software interage com atividades humanas. Mostrar o contexto dos use cases, bem como detalhes de use cases complexos
- Diagramas de estados:
  - Se o sistema tiver um life cycle interessante, com diferentes estados e eventos que fazem mudar o estado

# UML e processos de desenvolvimento

## Documentação

- Complemento à documentação
- Compreensão global do sistema
  - Não fazer diagramas detalhados do sistema (opinião)
  - Documentação detalhada deve estar no código
  - Focar aspetos importantes
- Package diagram:
  - mapa lógico do sistema
- Diagramas de classes:
  - apenas os aspetos importantes de cada package
- Diagramas de interação:
  - ajudar a compreender alguns aspetos dos diagramas de classes
- Máquinas de estados:
  - ajudar a perceber o ciclo de vida das classes, apenas para classes mais complexas

# Bibliografia

- UML Distilled A Brief Guide to the Standard Object Modeling Language. Martin Fowler. 3rd edition. Addison-Wesley Professional. 2003. Capítulo 1.
- Software Engineering. Ian Sommerville. 10th Edition. Addison-Wesley. 2016. Capítulo 5.