Introdução ao UML

Metodologias e Desenvolvimento de Software

Pedro Salgueiro
pds@di.uevora.pt
CLV-256

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
    - Programming language
      - Implementar um sistema, ou partes
    - Conceptual and software modeling

UML as Sketch

- Descrever/comunicar detalhes de um sistema
- Nível de abstracção 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 (completeness)**
  - 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
  - Diagramas UML
    - source code
  - Compilados directamente
- Ferramentas muito mais sofisticadas

UML as programming language

- Model Driven Architecture (MDA)
  - Standard
    - *UML as a programming language*
  - Confunde-se com UML
    - “Apenas” usa UML como linguagem base dos modelos
  - MDA
    - Platform Independet Model (PIM)
      - Modelo/representação do sistema/aplicação
      - Independente da plataforma/tecnologia
      - UML
    - Platform Specific Model (PSM)
      - Modelo/representação do sistema/aplicação
      - Dependente da plataforma
      - Um para cada plataforma

UML as programming language

- *Executable UML*
  - Parecido com MDA
    1. Modelos independentes da plataforma
    2. Model Compiler
       - Compila o modelo inical
       - Sistemas executável
       - 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?
  - Problemas
    - Ferramentas
      - Maturidade suficiente?
    - Produtividade
      - Executable UML vs outra linguagem de programação

Notações e meta-modelos

- UML define
  - Notações
    - Elementos gráficos dos modelos
    - Sintaxe gráfica da linguagem de modelação
      - Representação de cada conceito: classes, relações, etc…
  - 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
  - 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 struture
  - Deployment
  - Interaction overview
  - **Object**
  - Package
  - **Sequence**
  - **State machine**
  - Timing
  - **Use case**

UML “legal”

- O que é?
  - Resposta simples:
    - “O que está definido como bem formado na sua especificação”
  - Na prática
    - Não é tão simples
    - Standard
      - Está aberto a várias interpretações
    - Pode ter diferentes interpretações
    - Depende da utilização

UML “legal”

- 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 correctamente

- Suficiente?

- Grande conjunto de diagramas
  - Modelar diferentes aspectos 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, actividades, use cases, objectos, 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 perspectiva conceptual, podem ser usados para construir um *vocabulário* sobre o domínio do sistema
  - **Diagramas de actividades:** Workflow/fluxo de trabalho na empresa, mostrando como é que o software interage com actividades 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!**

- Design

  - **Diagramas de classes:** a partir de uma perspectiva 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 actividades 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

- 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*