Modelagem de Software e Visão Panorâmica da UML

Karin Becker
Engenharia de Software N
Instituto de Informática - UFRGS

Modelo de Software

Modelos

- Uma abstração do sistema segundo um certo **ponto de vista** e **nível de abstração**
  - aspectos essenciais do sistema

- Modelos visam:
  - Entender um problema complexo
    - Ex: cronograma de uma obra
  - Testar uma entidade física antes de lhe dar forma
    - ex.: modelos de aviões testados em túneis de vento
  - Comunicação com partes envolvidas
    - ex.: plantas baixas
  - Visualização
    - ex.: maquetes

Rede Pert : Gestão de Projeto

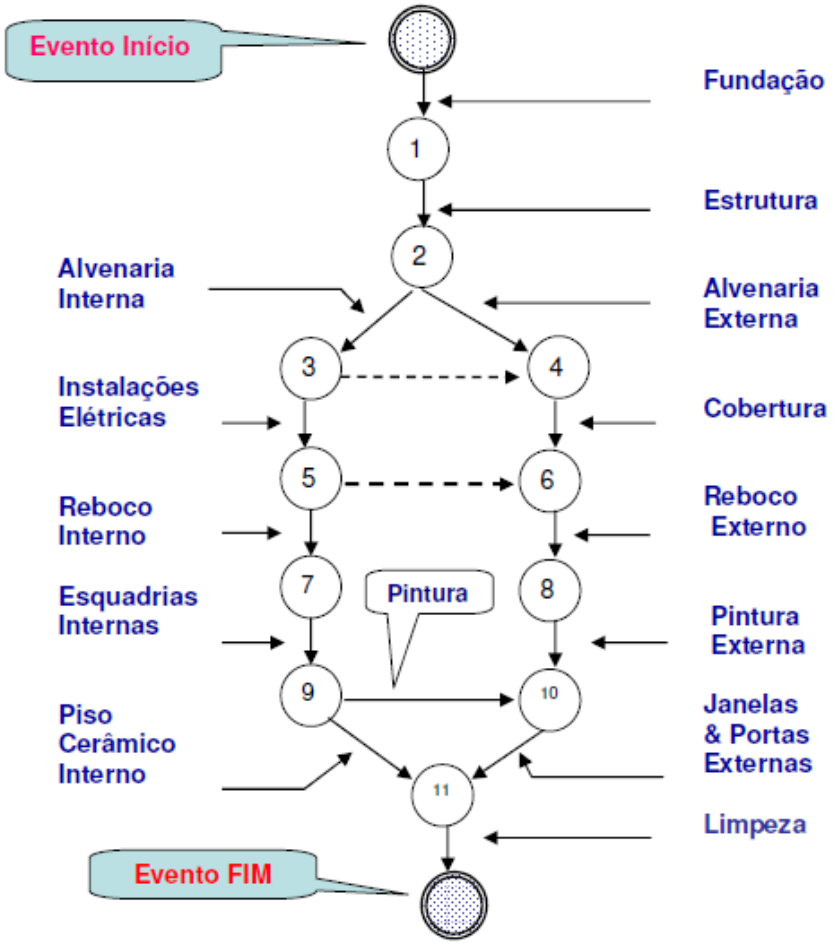


Fig.6.5.a – Exemplo de Rede Pert-CPM

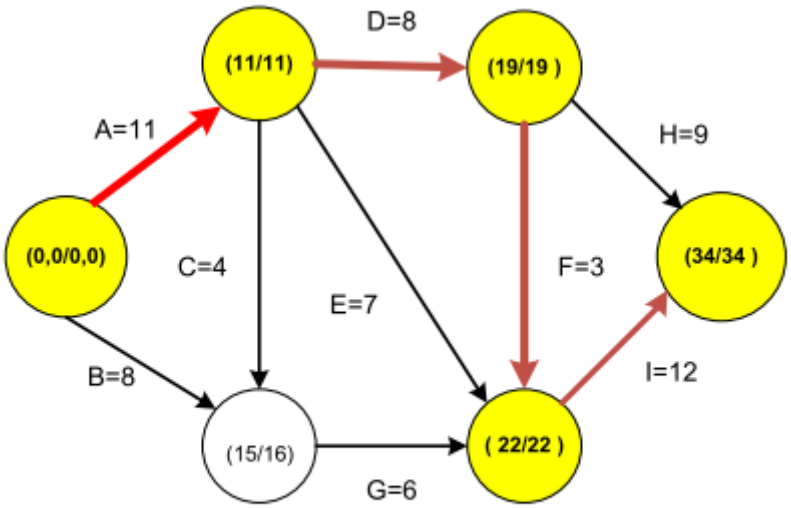
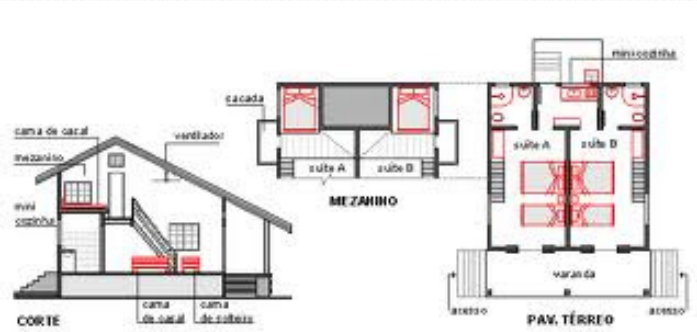
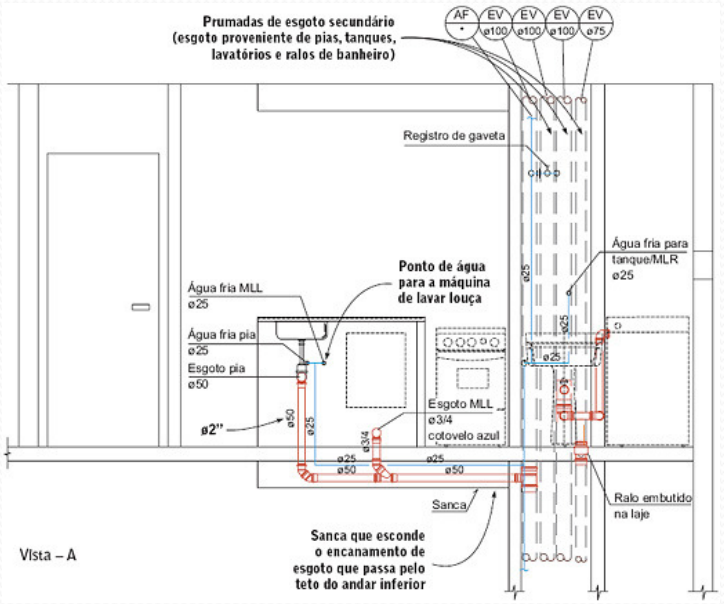


Fig.6.20 – Determinação do Caminho Critico

Arquitetura





Modelos de Software

- Gerenciamento da complexidade pela decomposição do sistema (do mundo real ou do software) em pedaços compreensíveis
- Comunicação com as várias pessoas envolvidas no processo de desenvolvimento de software
  - Caso de Uso, Estórias, Texto : Clientes, analistas, eng. testes

Arquiteto, Projetista, eng. teste,
Suporte, Usuário, etc …

  - Java, C++, Ruby, Python, Cobol: Programadores

Modelos de Software

- Código contém um nível de detalhe muito grande
- Impróprio a Abstração
  - Perder a noção do todo
  - Dificuldade de concentrar no que é importante
  - Dificuldade de lidar com a complexidade
  - **Pensar, entender !!!!!!**
  - Decisão específica em função de uma limitação/estilo da linguagem (ou do programador)
- documentos textuais e narrativos cansam e desestimulam;
  - Impreciso, ambíguo
  - Depende do estilo de quem o descreve
  - Pode sofrer de efeitos de (falta de/excesso) padronização

Modelo de Software

- Por quê investir em modelagem?
  - Estruturar processo de solução de um problema
  - Explorar múltiplas soluções (sem implementá-las)
  - Permitir abstrações para gerenciar complexidade e ocultar detalhes
  - Diminuir riscos de cometer erros
  - Confiabilidade pelo rigor e consistência entre as visões do sistema.
- Tarefa crítica no desenvolvimento de software é ATRIBUIR responsabilidades a componentes de software adequadamente **(ENTENDER, PENSAR, TESTAR IDEIAS)**
  - Inescapável (mesmo em projetos apressados)
  - Profundos efeitos na qualidade do Software
    - Robustez, Manutebilidade, Reusabilidade

Modelo de Software

- Objetivos de modelos:
  - Apoiar as atividades do processo **DURANTE** o desenvolvimento
    - permite organização de idéias para reflexão
    - Discussão e teste de idéias
    - Comunicação
  - Documentar o projeto (**DURANTE** ou **APÓS)**
    - Processos baseados em documentação
    - Engenharia reversa
  - Apoiar a Comunicação entre membros da equipe e usuários
    - membro-membro, membro-usuário

Modelos de Software

- Aceitação pelo usuário
  - Ausência de um modelo visível ao usuário faz com que ele dê conformidade a soluções incompletas ou mesmo erradas
  - Facilita a interação com o usuário
  - Obs: Opinião não é compartilhada pela tendência ágil

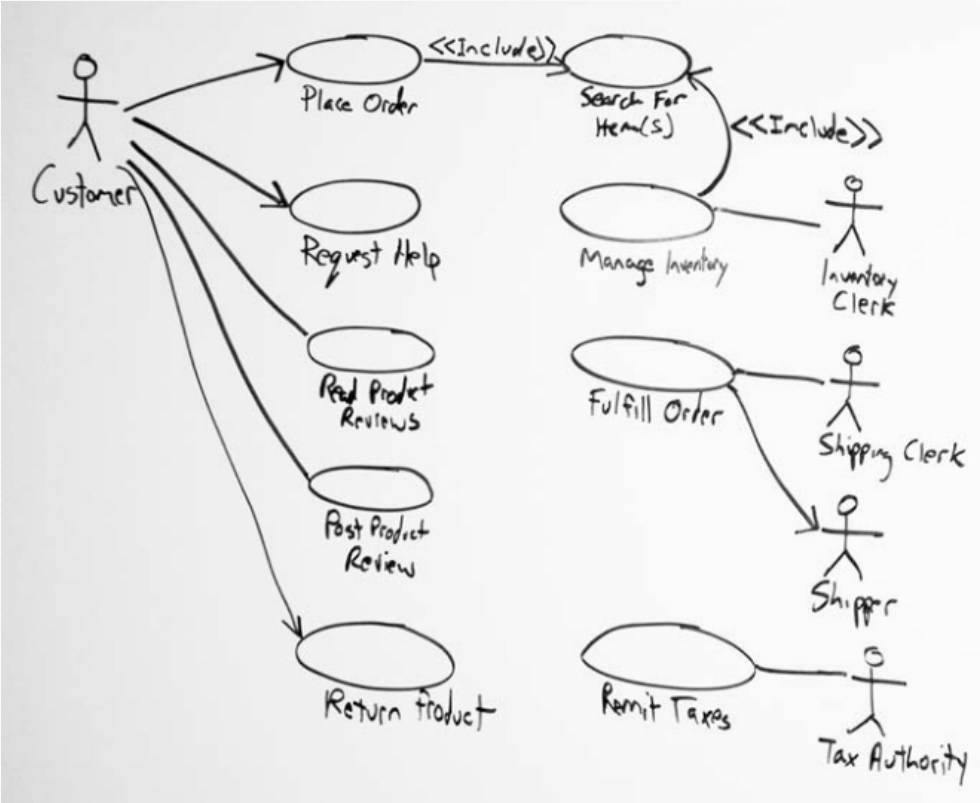
- Ciclo de vida muito comprido, Manutenção
  - O usuário modifica suas necessidades em função da dinâmica do mundo real
  - Dificuldade de abstrair o que é importante no código, entender impacto, planejar
  - as pessoas envolvidas podem não permanecer até o fim
  - Modelo é um documento importante

Princípios da Modelagem Ágil

- Modelar com um propósito
  - Entender
  - Comunicar
  - De acordo com seu propósito, modelos são temporários ou permanentes
- Modelos devem ser compreensíveis
  - Definir público alvo
  - Mais ou menos capricho
  - Estilo
  - Nível de detalhe
- Use as ferramentas corretas
  - Modelos se complementam
  - Permitem expressar um aspecto específico

Exemplo

- Você vai pedir a um cliente que aprove seu documento de requisitos
- Você está discutindo com seus colegas quais seriam os requisitos



Princípios da Modelagem Ágil

- Modelos devem ser simples
- Modelo precisa ser suficientemente preciso e detalhado
  - De acordo com seu propósito
  - Ex: mapa
  - **Investir o tempo que agrega valor**
  - O preço da burocracia
- Modelo precisa agregar valor
  - Ferramentas certas
  - **Retorno do tempo investido**
  - **Modelo não é necessariamente documento**
    - mas pode vir a ser

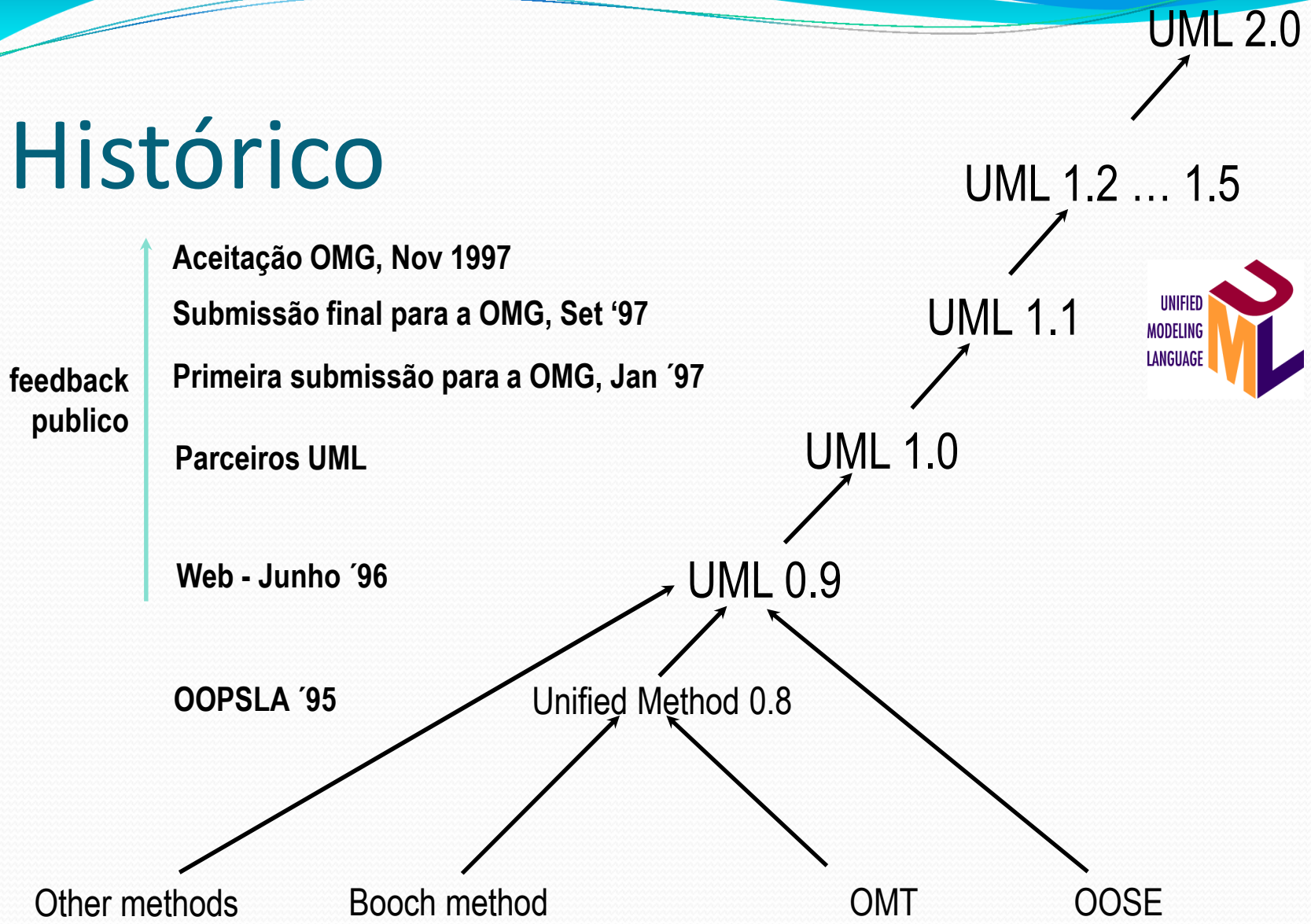
UML – Visão Panorâmica

Contexto

- Mercado fragmentado
- Falta de padronização
- Resistência da indústria e usuários em investir em OO

"it was missing a *lingua franca* for modeling"

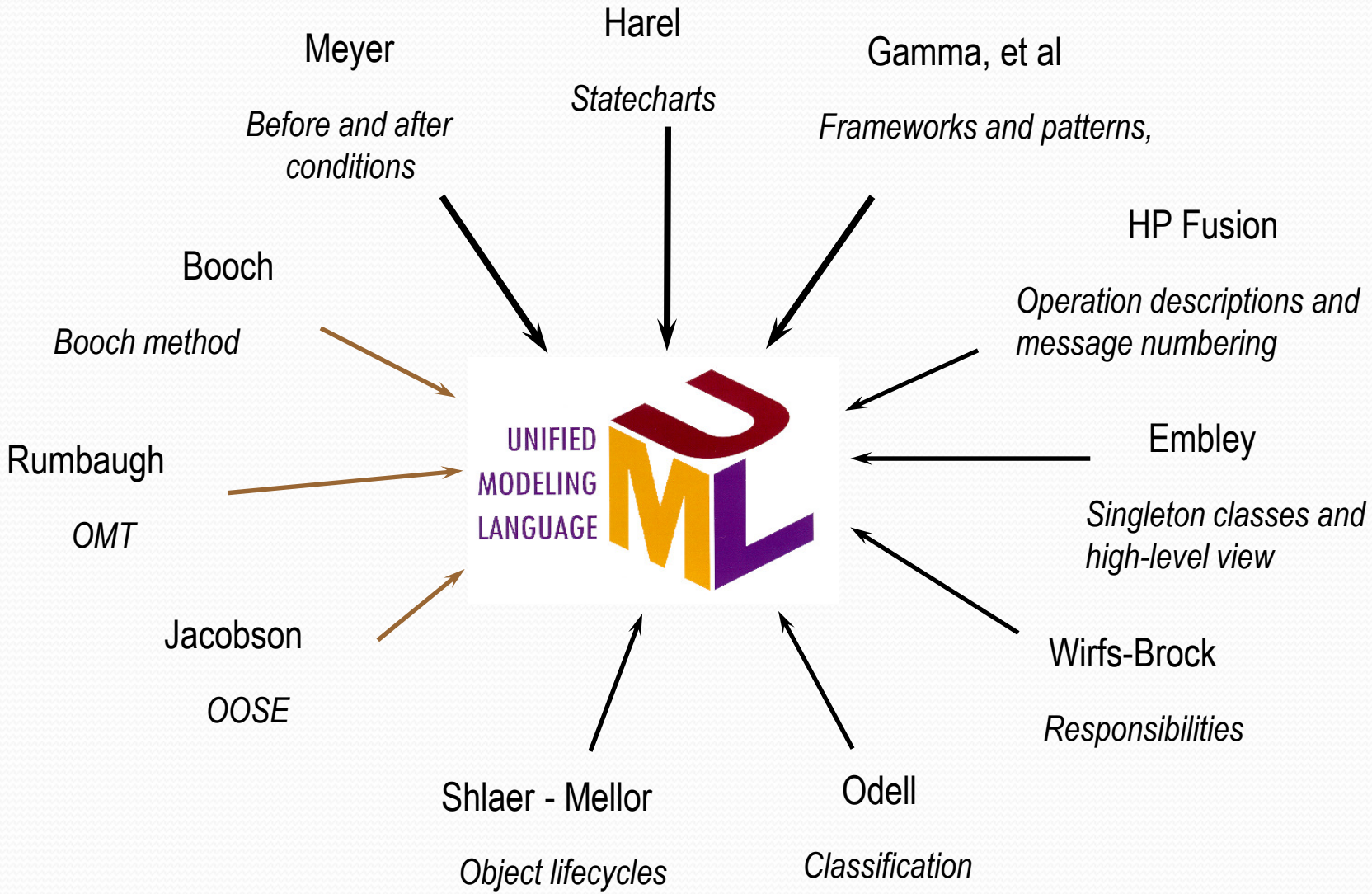
Histórico


UML 2.0
UML 1.2 … 1.5
UML 1.1
UML 1.0
UML 0.9
Unified Method 0.8
Other methods
Booch method
OMT
OOSE
Aceitação OMG, Nov 1997
Submissão final para a OMG, Set '97
Primeira submissão para a OMG, Jan ´97
Parceiros UML
Web - Junho ´96
OOPSLA ´95
feedback publico
UNIFIED MODELING LANGUAGE

Histórico

- Rational Software: UML 0.9 (1996)
  - Rumbaugh & Booch, + Jacobson
- 1996: OMG Request For Proposal (RFP)
  - Consórcio : UML 1.0 (Jan. 1997)
    - DEC, HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI, Systemhouse, Microsoft, Oracle, Rational Software, TI, and Unisys
  - Consórcio : UML 1.1 (Set. 1997)
    - + IBM & ObjecTime; Platinum Technology; Ptech; Taskon & Reich Technologies; and Softeam
  - revisões
    - UML 1.2 (Junho/1998)
    - UML 1.3 (Final de 1998): estável
    - UML 1.4 (set 2001), UML 1.5 (março 2003): revisões menores
  - UML 2.0: revisão profunda, 2005 (anunciada desde 2000 ....)

Contribuições para a UML



Alguns Parceiros UML

- Rational Software Corporation
- Hewlett-Packard
- I-Logix
- IBM
- ICON Computing
- Intellicorp
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle
- Platinum Technology
- Taskon
- Texas Instruments/Sterling Software
- Unisys

Visão Geral da UML

- A **UML** é uma linguagem destinada a:
  - visualizar
  - especificar
  - construir
  - documentar

os artefatos de um sistema complexo de software.

Artefatos: diagramas, modelos e documentos.

Objetivos

- Descrever modelos de sistema - do mundo real e de software - baseado em conceitos de objetos.
  - Fornecer uma linguagem de modelagem OO visual fácil, pronta para uso, permitindo amplas facilidades de modelagem
  - Fornecer mecanismos de extensibilidade e especialização de conceitos de base
  - Independência de processos e linguagens de programação
    - abranger todo o ciclo de vida
    - diferentes tecnologias de implementação
  - Integrar melhores práticas em desenvolvimento OO de sistemas

Vantagens

- Padronização
  - impulso no desenvolvimento e adoção de ferramentas para desenvolvimento OO de software
  - portabilidade, interoperabilidade
- Bom compromisso entre conceituação e flexibilidade:
  - ampla variedade notacional
  - facilidade de extensão/personalização
  - facilidade de evolução (conceitual, tecnológica)

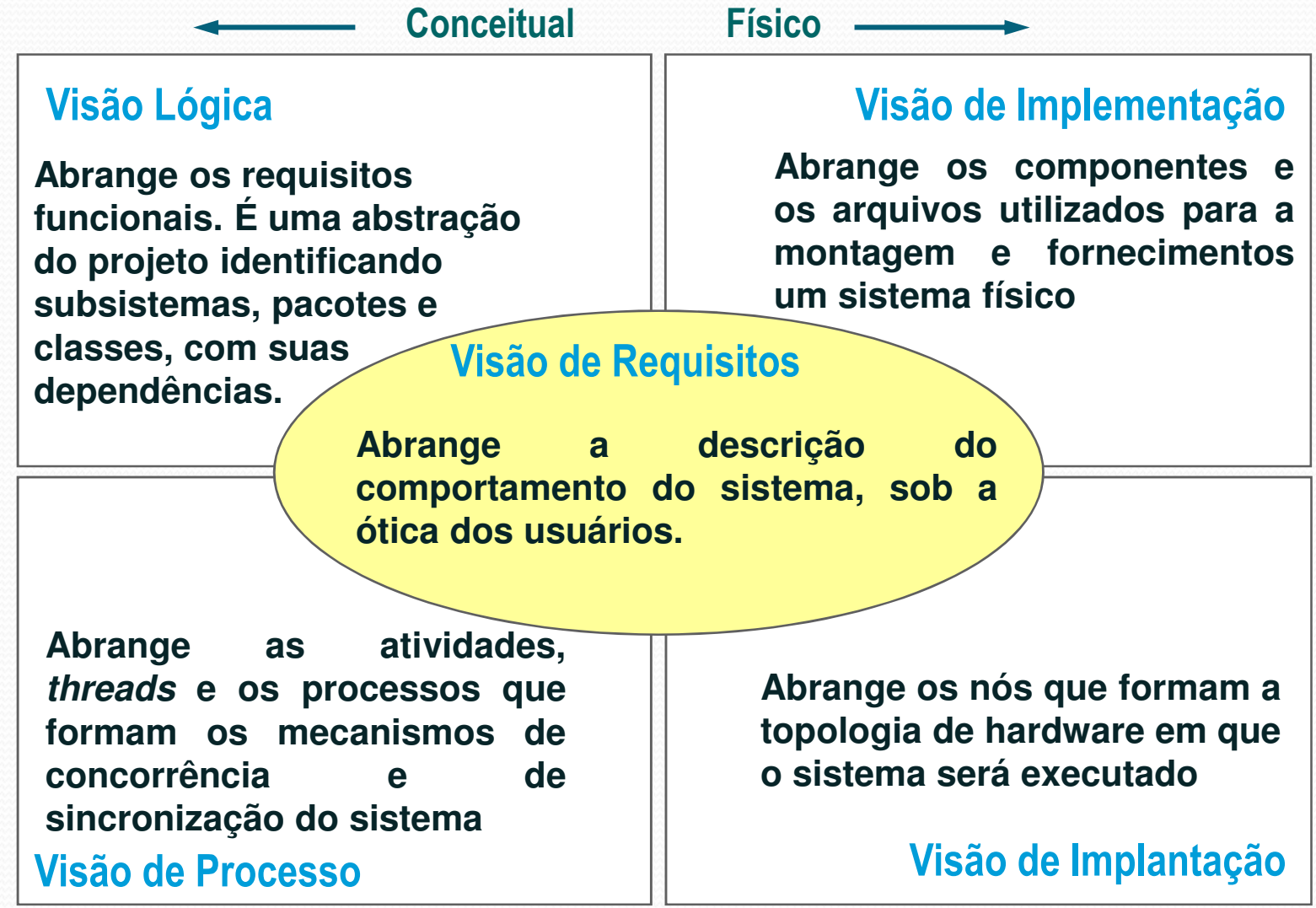
UML: “resistências”

- Qualidade de Modelos
  - Rigor, simplicidade, expressividade, ortogonalidade
- UML
  - Muitos modelos
  - Não há modelos suficientes
  - Construtores e diagramas não são ortogonais
  - Semântica é ambígua
    - “semântica” guia definição de ferramentas (diagramadores e repositórios)
  - Dificuldade de manter consistência entre modelos
  - Não há metodologia ou técnica associada
  - Diagramas têm afinidade com a OO, mas nem sempre

Termos e Conceitos

- **Sistema**: coleção de subsistemas organizados para a realização de um objetivo e descritos por um conjunto de modelos.
- **Subsistema**: representa uma partição dos elementos de um sistema maior em partes independentes.
- **Modelo**: são abstrações semanticamente fechadas de um sistema, representando uma simplificação autoconsistente e completa da realidade.
- **Diagramas**: apresentação gráfica de um conjunto de elementos.
- **Visão**: abrange um subconjunto de itens que pertencem a um modelo, cujo foco esta voltado para um único aspecto do sistema.

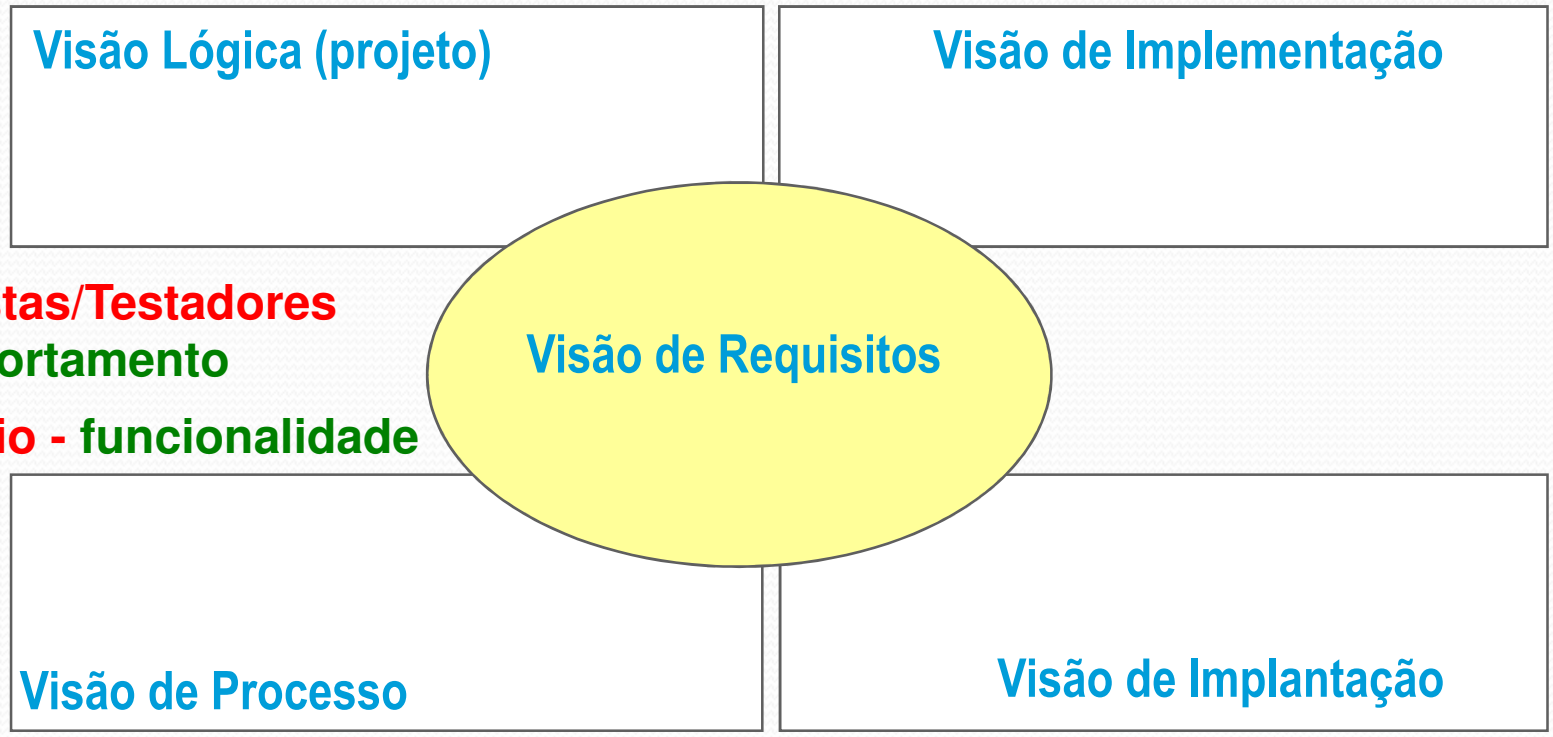
Arquitetura 4 + 1



Arquitetura 4 + 1

Analistas/Projetistas – estrutura/funcionalidade

Programadores – gerenciamento de Software



Analistas/Testadores
Comportamento

Usuário - funcionalidade

Integradores de Sistemas
desempenho, escalabilidade, fluxo

Engenharia de Sistemas
Topologia, Entrega, Instalação, Comunicação

Diagramas Estruturais

- **Diagrama de Classes**
  - mostra um conjunto de classes, interfaces, e seus relacionamentos
  - visões: lógica e processo
- **Diagrama de Objetos**
  - mostra um conjunto de objetos e seus relacionamentos (instanciação do diagrama de classes)
  - visões: lógica e processo
- **Diagrama de Pacotes**
  - sistema divididos em agrupamentos lógicos mostrando as dependências entre estes
  - Visões: quase todas

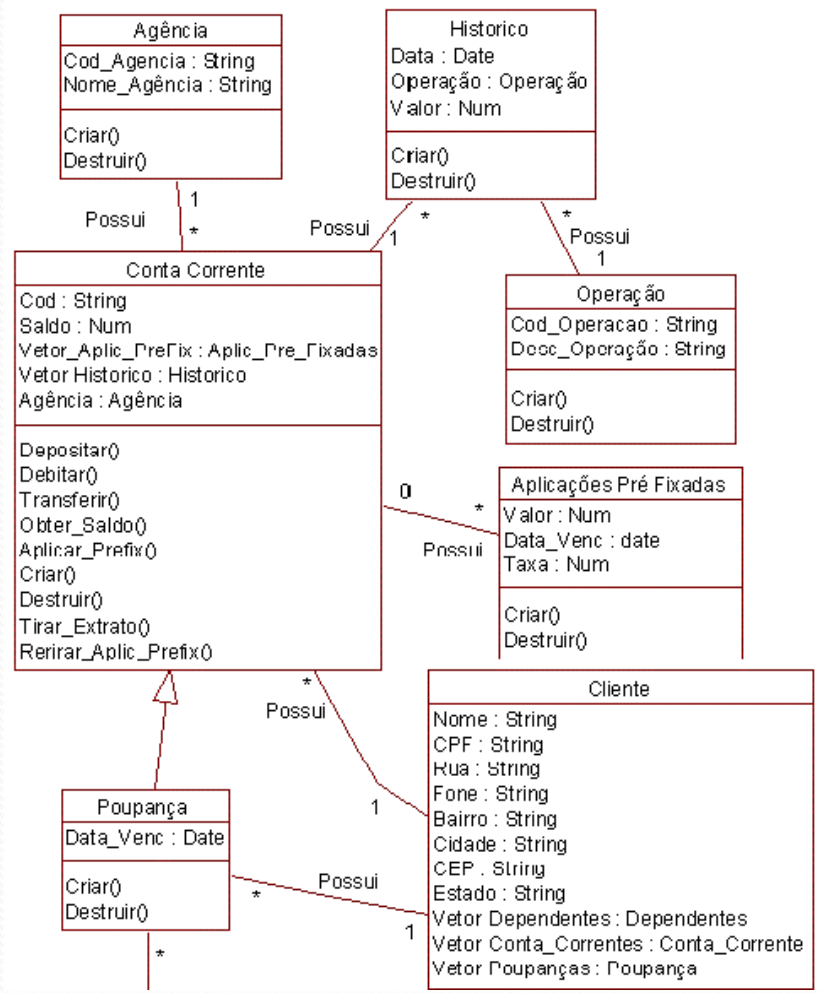


Diagrama de Classes

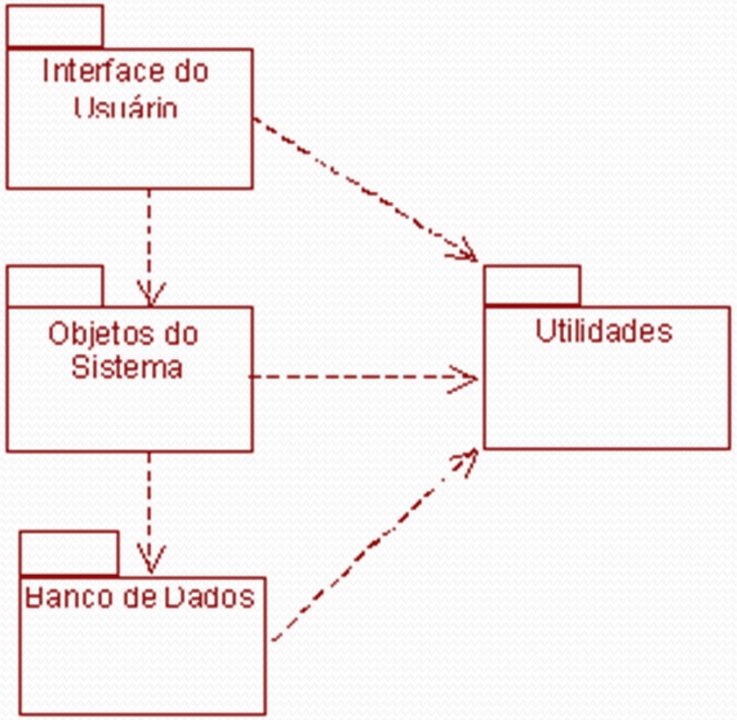


Diagrama de Pacotes

Diagramas Estruturais

- **Diagrama de Implantação**
  - mostra a modelagem dos aspectos físicos de um sistema (configuração em tempo de execução)
  - visão de implantação
- **Diagrama de Componentes**
  - mostra um conjunto de componentes e seus relacionamentos
  - visão de implementação

Diagramas Comportamentais

- **Diagrama de Casos de Uso***
  - mostra um conjunto de atores e casos de uso
  - organiza e modela o comportamento do sistema
  - visão de caso de uso
- **Diagrama de Interação**
  - Subtipos com diferentes ênfases
    - Sequência: enfatiza o ordenamento das mensagens trocadas entre os objetos
    - Comunicação : enfatiza a organização estrutural dos objetos que trocam mensagens
    - Tempo: enfatiza a interação na escala do tempo, mostrando as condições que mudam no decorrer desse período.
  - visões: lógica e processo

* Alguns consideram estrutural

Diagrama de Casos de Uso

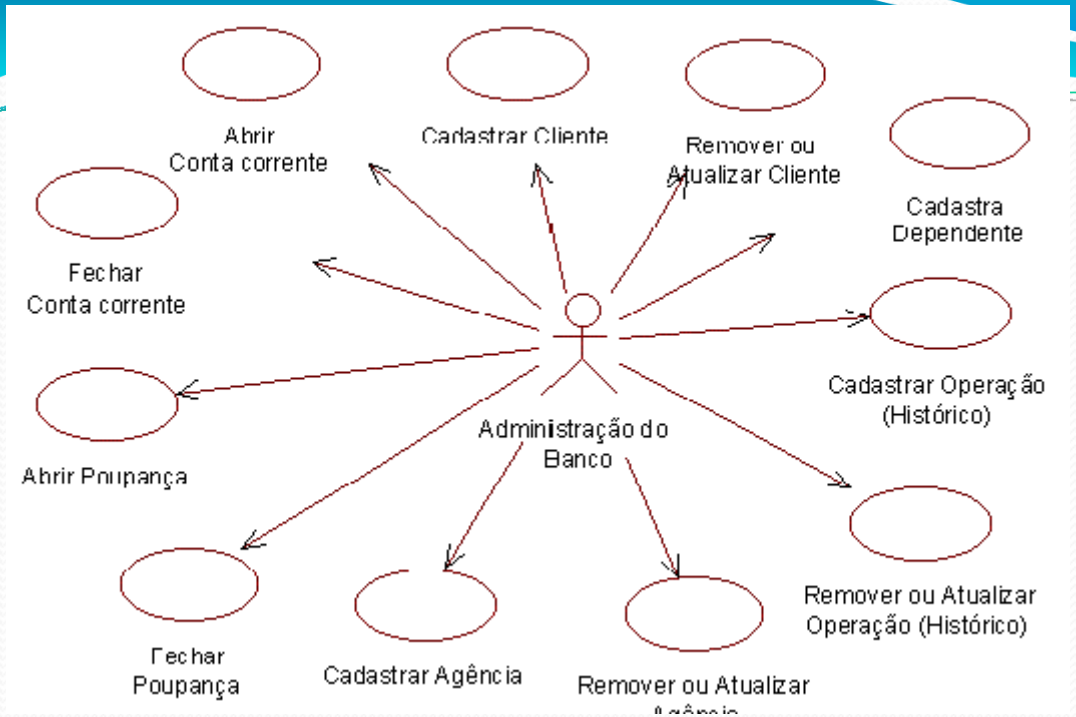
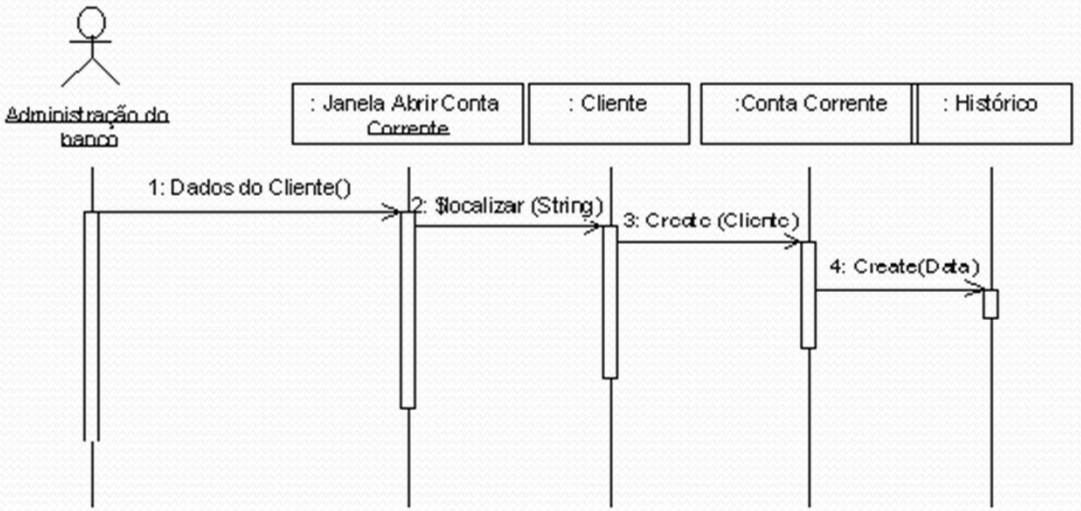


Diagrama de Sequência



Diagramas Comportamentais

- **Diagrama de Atividade**
  - enfatiza o fluxo de uma atividade a outra no sistema
  - visões: requisitos, lógica e processo

- **Diagrama de Estados**
  - enfatizam o comportamento de um objeto de acordo com um conjunto de eventos
  - particularmente útil na construção de sistemas reativos
  - visões: lógica e processo

Modelando software com UML: um cenário comum

- Representar as relações do sistema com o mundo real e suas maiores funções usando Casos de Uso e Atores
- Ilustrar realizações de Casos de Uso com Diagramas de Interação e/ou Diagramas de Atividades
- Representar a estrutura estática de um sistema usando Diagramas de Classes
- Modelar o comportamento de objetos com Diagramas de Interação e/ou Diagramas de Estado
- Revelar a arquitetura de implementação física com Diagramas de Componentes e Implantação

UML : Visão Pragmática

- "You can model 80% of most problems by using about 20% UML. You learn those 20%. For the rest, see the official UML definition" (www.uml.org)
  - Aprenderemos nesta disciplina as principais características dos principais diagramas
- UML é um **conjunto de notações**
  - **Tem dicas para usar, mas não é metodologia**
- Nesta disciplina
  - utilizaremos o arcabouço do processo unificado para relacionar modelos/diagramas com atividades do desenvolvimento de software
  - complementaremos com técnicas propostas por Larman (prescritivas) e Ambler (modelagem ágil) para modelar
  - praticaremos o uso de ferramentas CASE
  - insistiremos no uso correto da notação, e na produção de modelos completos, corretos e preciso

Para saber mais

- Larman, Craig. Utilizando UML e Padrões - Uma Introdução à Análise e ao Projeto Orientados a Objetos, Bookman.

  Descreve passo a passo UM processo de Análise e Projeto Orientados a Objetos utilizando a notação UML. Aborda também o uso de padrões de projeto.

  As duas primeiras edições são mais objetivas e sucintas, a terceira é mais focada em desenvolvimento iterativo e ágil.

- Ambler, S. , *Modelagem Ágil*, Bookman, 2004.

  Descreve a modelagem segundo a filosofia ágil, contextualizando-a em RUP e XP

- Ambler, S. , *The Elements of UML 2.0 Style* , Cambridge, 2005.

  Discute cada modelo, com dicas de bom uso. Bom para iniciantes, mas se concentra na notação.

- Fowler, M. ; Scott, K. UML Essencial, Bookman, 2005.

  Livro de referência sobre UML mas descreve apenas a notação e os modelos e não o processo de construí-los.

  Está um pouco defasado, pois considera a UML 1.x