MODELAGEM COM A UML
(UNIFIED MODELING LANGUAGE)

- BREVE HISTÓRICO
- CARACTERÍSTICAS
- CONCEITOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS
- MODELAGEM DE ANÁLISE E DE PROJETO

I. BREVE HISTÓRICO

Em fins dos anos 80 e início dos anos 90 vários métodos orientados a objetos surgiram, entre eles os métodos de Grady Booch, Jim Rumbaugh (OMT) e Ivar Jacobson.

A UML foi uma tentativa de unificar as notações destes três métodos. Foi concebida por esses profissionais.

A idéia era produzir um padrão, com as melhores práticas adotadas pela indústria, levando mais desenvolvedores a modelar seus sistemas de software antes de construí-los.

Histórico

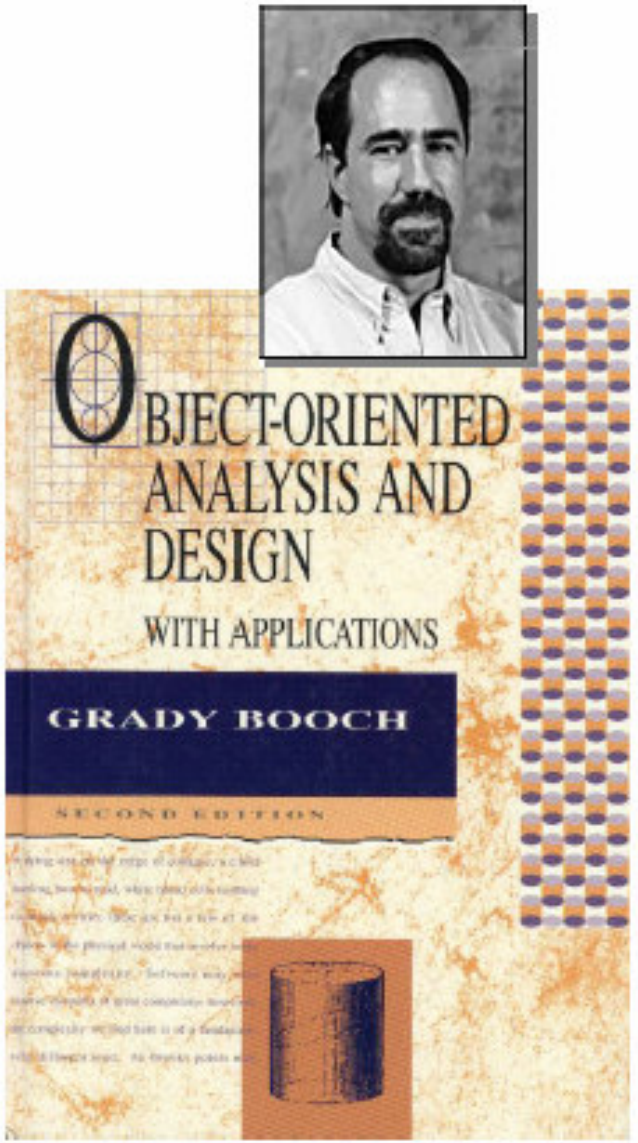
- Diversas metodologias e métodos surgiram para apoiar OO
  - Evolução a partir de linguagens C++ e SmallTalk
  - Anos 80-90: diversidade de autores
  - Anos 98-2000: unificação em torno de UML

Histórico

- Grady Booch
  - Um dos pioneiros da OO
  - 1980: ênfase em técnicas de projeto para Ada
  - 1992-1994: livros
    - *Object-Oriented Design with Applications*
      - projeto de programas em C++ e Ada

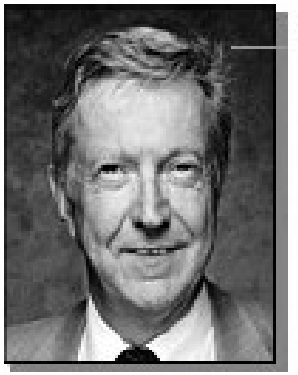
Histórico

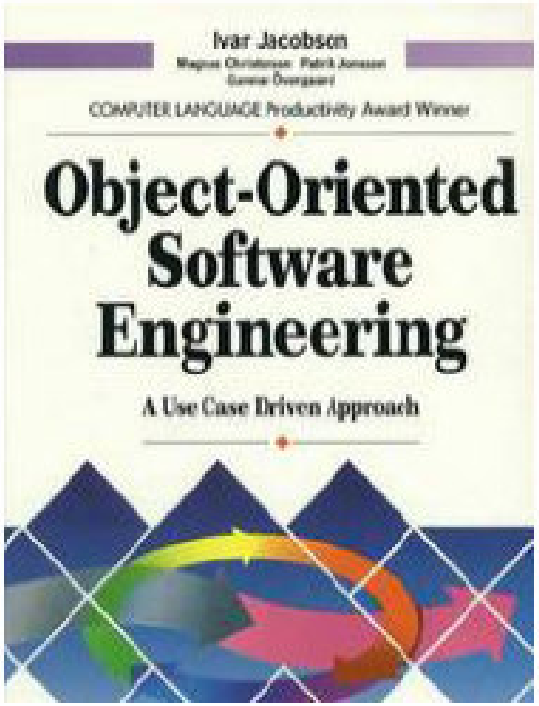
- 1994: *Object-Oriented Analysis and Design with Applications*
  - texto sobre conceitos de OO e modelagem de objetos
  - projeto de várias aplicações-exemplo com diferentes linguagens da época
  - base de UML
- 1998: Fundação da Rational



Histórico

- Ivar Jacobson
  - Modelagem OO baseado em Casos de Uso
  - *Objectory*



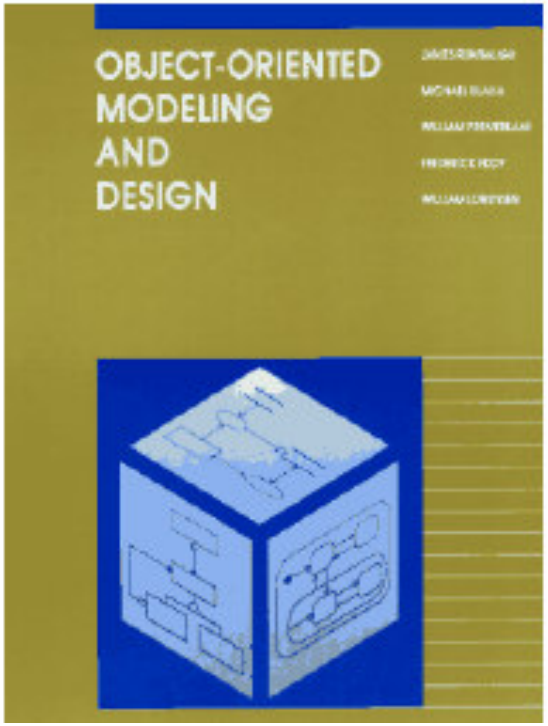


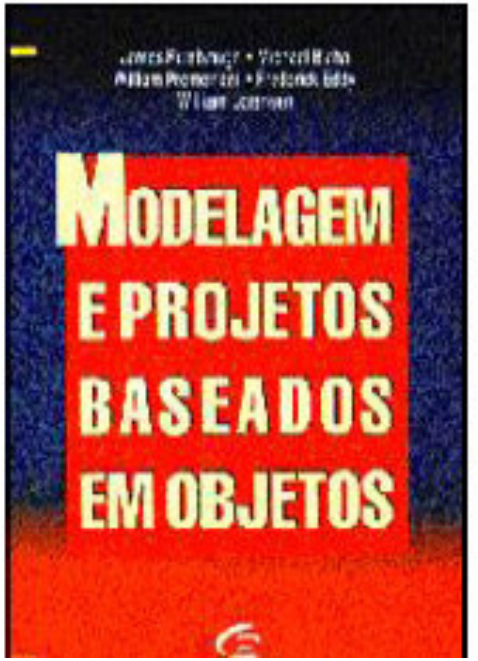
Histórico

- James Rumbaugh
  - Object Modeling Technique (OMT)
  - Desenvolvida na GE
  - Metodologia baseada em notações pré-existentes (ER, DTE, DFD)
  - Clara distinção entre as três visões do problema

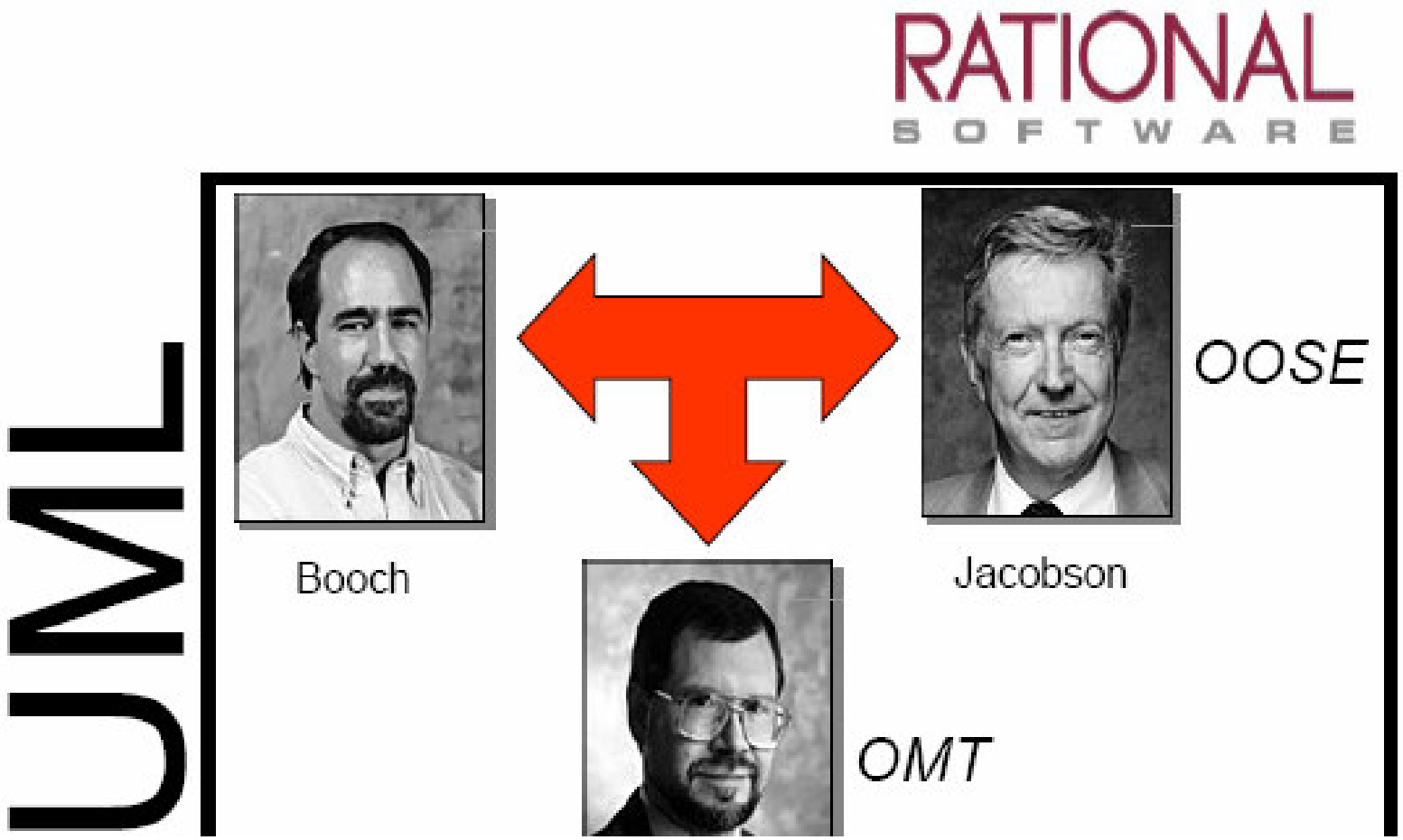
Histórico

- James Rumbaugh (cont.)

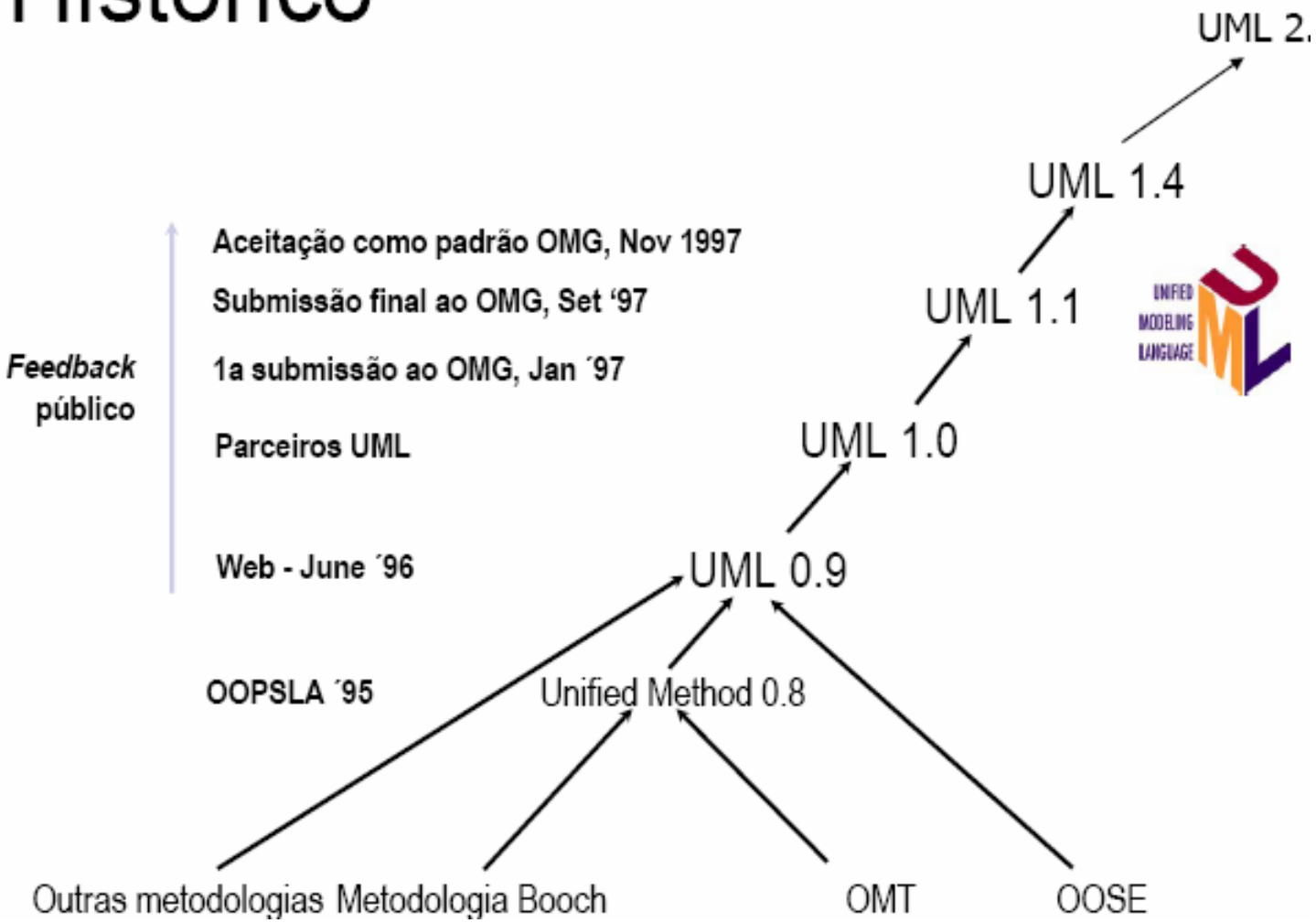




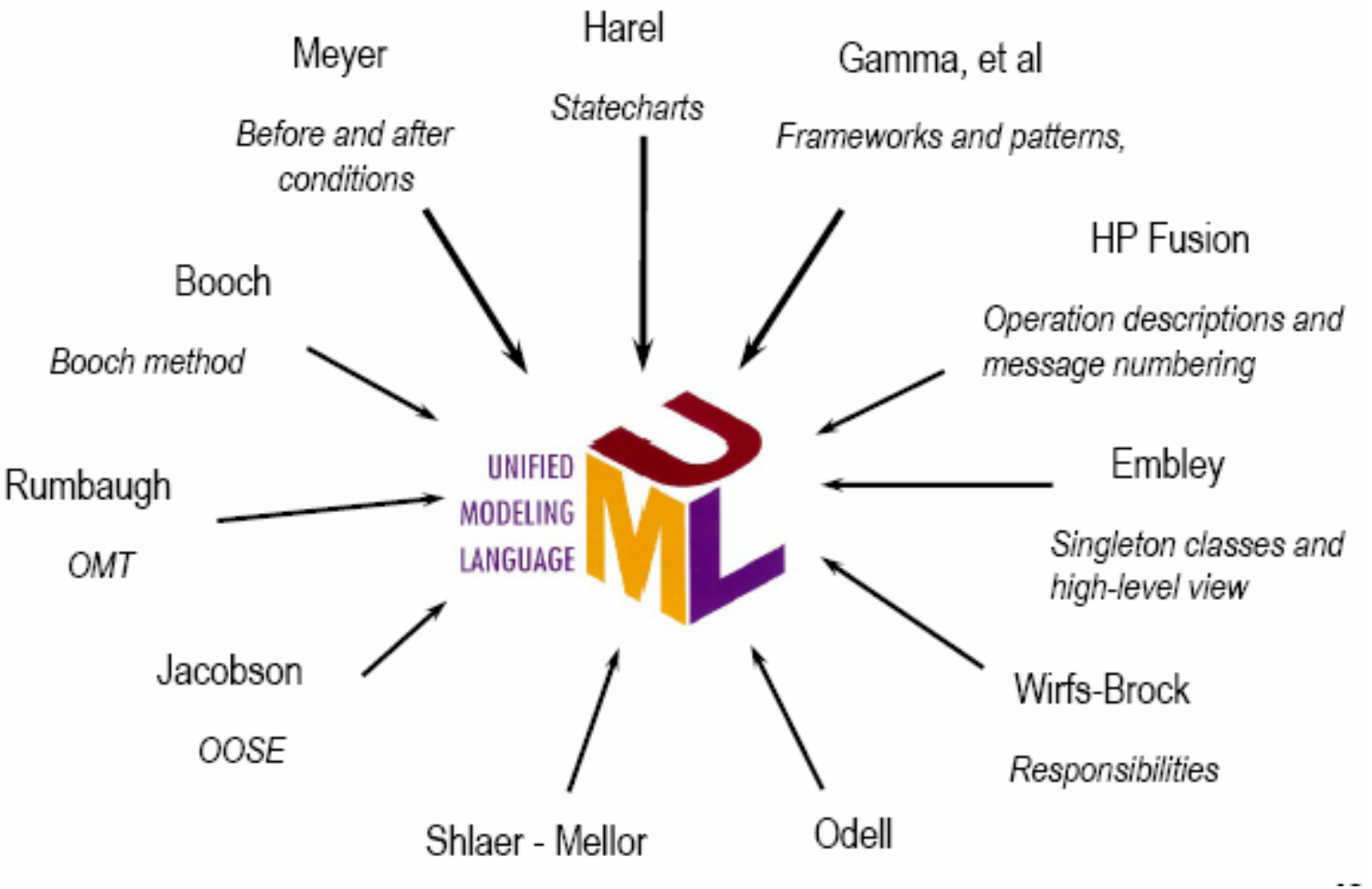
Histórico



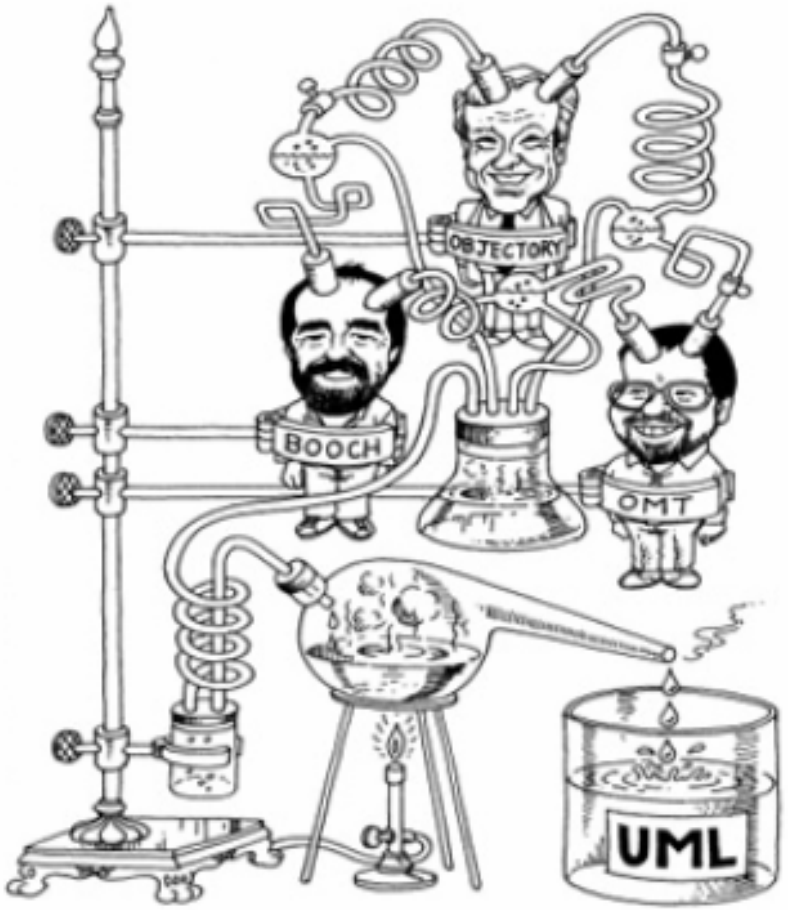
Histórico

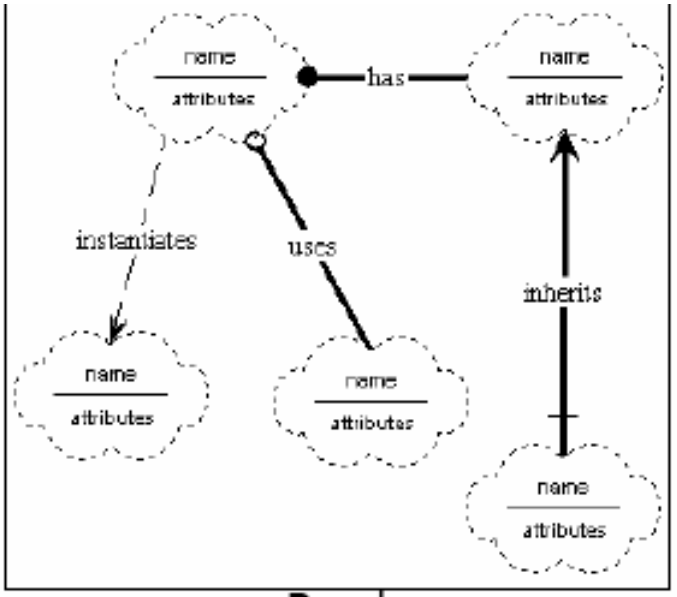

UML 2.
UML 1.4
UML 1.1
UML 1.0
UML 0.9
Aceitação como padrão OMG, Nov 1997
Submissão final ao OMG, Set '97
1a submissão ao OMG, Jan ´97
Parceiros UML
Web - June ´96
Feedback público
OOPSLA ´95
Unified Method 0.8
Outras metodologias
Metodologia Booch
OMT
OOSE

Histórico

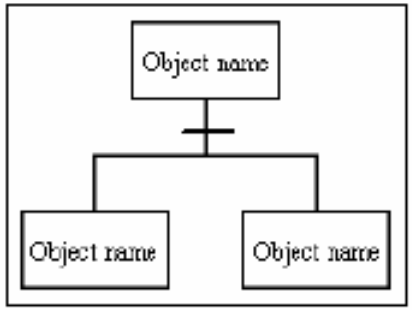

Meyer
Before and after conditions
Harel
Statecharts
Gamma, et al
Frameworks and patterns,
HP Fusion
Operation descriptions and message numbering
Booch
Booch method
Rumbaugh
OMT
UNIFIED MODELING LANGUAGE
UML
Embley
Singleton classes and high-level view
Jacobson
OOSE
Wirfs-Brock
Responsibilities
Shlaer - Mellor
Odell

Histórico

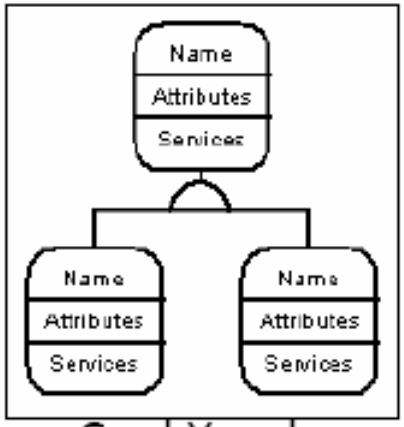




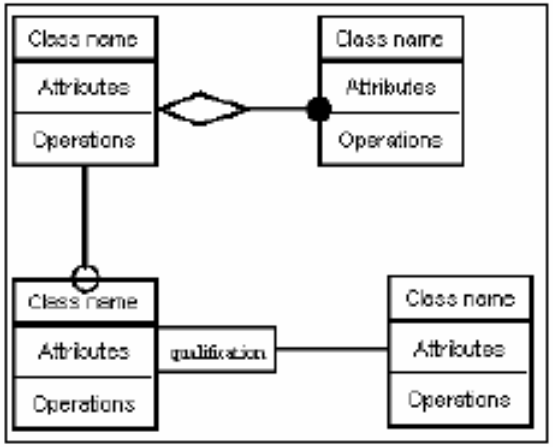
Booth



Schlaer-Mellor



Coad-Yourdon



OMT

II. CARACTERÍSTICAS

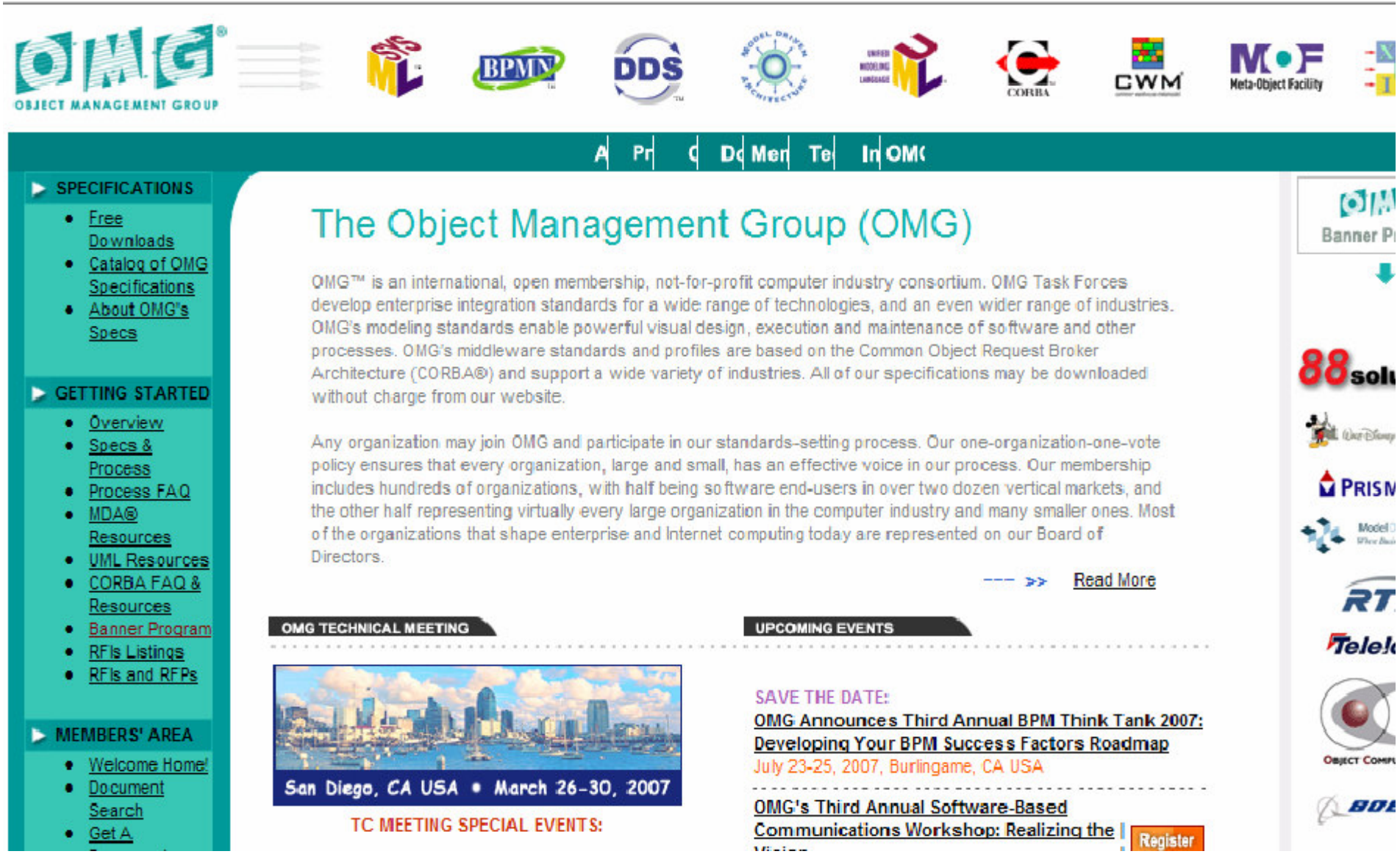
- A UML não é um método e pode ser utilizada por diferentes processos de desenvolvimento de software

- A UML foi reconhecida pelo OMG (Object Management Group) como uma linguagem de modelagem padrão.

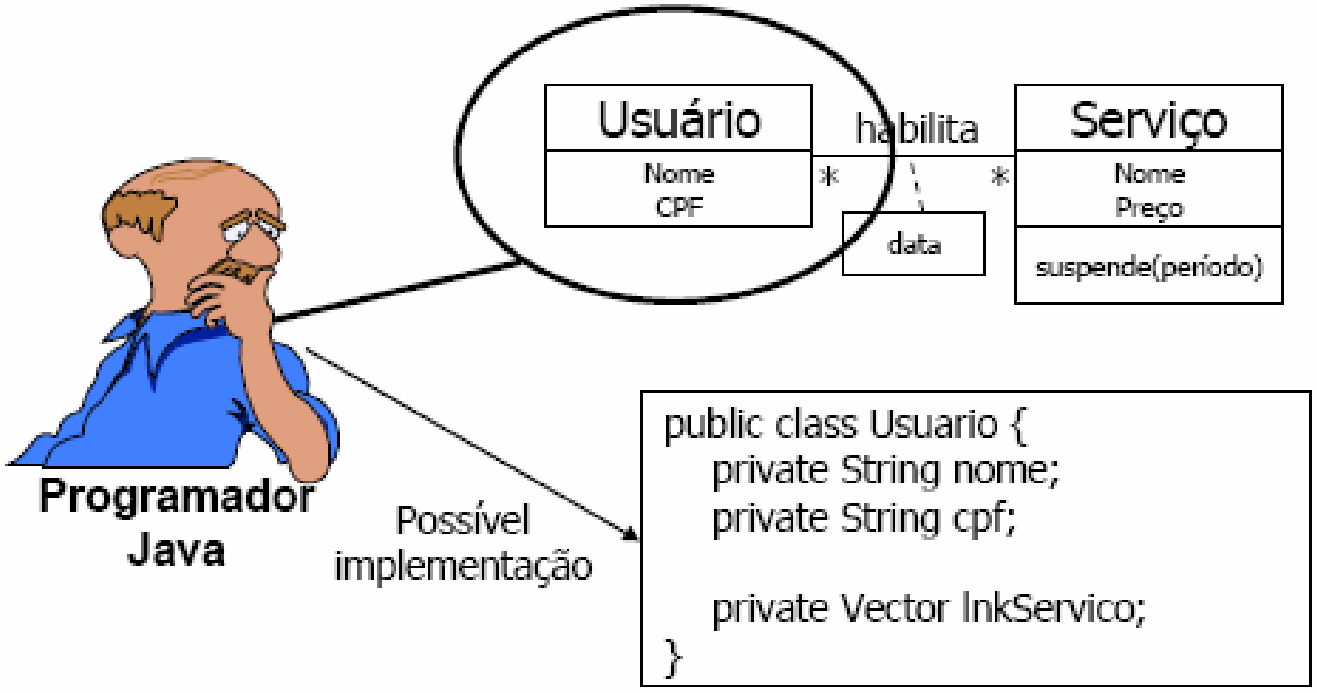
(OMG – uma associação aberta que desenvolve e mantém especificações utilizadas pela indústria da computação)

- Obtenha a especificação da UML em http://www.omg.org

- A UML utiliza conceitos de orientação a objetos.


OBJECT MANAGEMENT GROUP
BPMN
DDS
CORBA
CWM
Meta-Object Facility
SPECIFICATIONS
Free Downloads
Catalog of OMG Specifications
About OMG's Specs
GETTING STARTED
Overview
Specs & Process
Process FAQ
MDA® Resources
UML Resources
CORBA FAQ & Resources
Banner Program
RFIs Listings
RFIs and RFPs
MEMBERS' AREA
Welcome Home!
Document Search
The Object Management Group (OMG)
OMG™ is an international, open membership, not-for-profit computer industry consortium. OMG Task Forces develop enterprise integration standards for a wide range of technologies, and an even wider range of industries. OMG's modeling standards enable powerful visual design, execution and maintenance of software and other processes. OMG's middleware standards and profiles are based on the Common Object Request Broker Architecture (CORBA®) and support a wide variety of industries. All of our specifications may be downloaded without charge from our website.
Any organization may join OMG and participate in our standards-setting process. Our one-organization-one-vote policy ensures that every organization, large and small, has an effective voice in our process. Our membership includes hundreds of organizations, with half being software end-users in over two dozen vertical markets, and the other half representing virtually every large organization in the computer industry and many smaller ones. Most of the organizations that shape enterprise and Internet computing today are represented on our Board of Directors.
Read More
OMG TECHNICAL MEETING
San Diego, CA USA • March 26-30, 2007
TC MEETING SPECIAL EVENTS:
UPCOMING EVENTS
SAVE THE DATE:
OMG Announces Third Annual BPM Think Tank 2007: Developing Your BPM Success Factors Roadmap
July 23-25, 2007, Burlingame, CA USA
OMG's Third Annual Software-Based Communications Workshop: Realizing the
Register

- De propósito geral
  - Não está presa a uma linguagem de programação



Finalidades do UML

-Visualizar

-Especificar

-Construir

-Documentar

Elementos do UML

-Itens

-Relacionamentos

-Diagramas

Itens do UML

-Estruturais
-Comportamentais
-Agrupamento
-Anotacionais

Itens Estruturais do UML (parte estática)

-Classes (conjunto de objetos com caract. Comuns)
-Interface (serviços de uma classe ou componente)
-Colaborações (comportamento colaborativo)
-Caso de Uso (sequência de ações)
-Classes Ativas (objetos com threads)
-Componentes (pacotes físicos de elementos lógicos)
-Nó (recurso computacional)

Itens Comportamentais do UML (parte dinâmica)

-Interação (intercâmbio de dados)
-Máquina de Estados
    -Estados
    -Transições
    -Eventos
    -Atividades

Itens de Agrupamento do UML (organizacional)

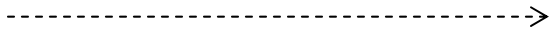
-Pacotes

Itens Anotacionais do UML (explicativo)

-Nota

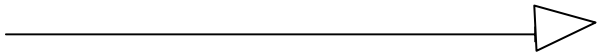
Relacionamentos do UML

-Dependência (relacionamento semântico de dois itens)

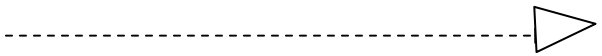


-Associação (relacionamento estrutural)

-Generalização (hierarquia)



-Realização (contrato de uma das partes)



Diagramas do UML

-Classes
-Objetos
-Casos de Uso
-Sequência
-Colaborações
-Gráfico de Estados
-Atividades
-Componentes
-Implantação

III. CONCEITOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

- Na programação orientada a objetos os dados a serem processados e os mecanismos de processamento destes dados devem ser analisados em conjunto.

- Assim, programadores que utilizam o paradigma de programação orientada a objetos criam e usam objetos.

- Na abordagem orientada a objetos os dados são subdivididos em **objetos**

- Cada objeto tem sua própria identidade. Assim, dois livros, no sistema de venda de livros, por mais semelhanças que contenham constituem cada um, um único objeto

- Objetos com a mesma estrutura de dados (atributos), com o mesmo comportamento (operações) e relacionamentos são agrupados numa classe

- Assim, uma classe Livro descreve o que é comum em todos os livros no contexto de um determinado sistema.

Exemplo:

```
import java.util.*;

class Livro
{
        private String isbn;
        private String titulo;
        private GregorianCalendar dataPublicaçao;
        private int quantidade;
        private float preço;
        private GregorianCalendar dataAlteraçaoPreço;

        Livro (String cod,String tit,GregorianCalendar dataPubl,int quant,
                        float   pr, GregorianCalendar dataAltPr)
        {
                isbn = cod;
                titulo = tit;
                dataPublicaçao = dataPubl;
                quantidade = quant;
                preço = pr;
                dataAlteraçaoPreço = dataAltPr;
        }
```

```
	public void alteraPreço (float percentual, GregorianCalendar dataAltPr)
	{
		preço = preço + percentual/100 * preço;
		dataAlteraçaoPreço = dataAltPr;
	}

	public String toString ()
	{
		String umLivro;
		umLivro = "ISBN - " + isbn +  "\n"
				+ "Titulo - " + titulo + "\n"

				+ "Data de publicacao - "+ dataPublicaçao.get(Calendar.DATE)+ "/"
				+  (dataPublicaçao.get(Calendar.MONTH) + 1) + "/"
				+  dataPublicaçao.get(Calendar.YEAR) + "\n"

				+ "Quantidade -  " + quantidade + "\n"
				+ "Preco - " + preço + "\n"

				+ "Data da última alteracao de preco - "
				+    dataAlteraçaoPreço.get(Calendar.DATE) + "/"
				+  (dataAlteraçaoPreço.get(Calendar.MONTH) + 1) + "/"
				+  dataAlteraçaoPreço.get(Calendar.YEAR) + "\n";

		return umLivro;
	}
}
```

- Devemos pensar em um objeto como algo que tem responsabilidades. Objetos devem ser responsáveis por si mesmos e ter essas responsabilidades claramente definidas.

- No nível conceitual um objeto deveria ser pensado desta forma: um objeto é um conjunto de responsabilidades.

- Como os objetos têm responsabilidades e são responsáveis por si próprios, deve haver um modo de informá-los sobre o que devem fazer.

- Objetos dispõem de dados para informá-los sobre si mesmos e métodos para implementar funcionalidades.

- Alguns desses métodos podem ser invocados por outros objetos. A coleção desses métodos é denominada interface pública do objeto.

Comparando com a Progr. Orientada a Procedimentos:

- Na Progr. Orientada a Procedimentos é identificada a tarefa a ser realizada e através de refinamentos sucessivos, quebra-se essa tarefa em subtarefas menores, e estas em subtarefas ainda mais simples até que estas subtarefas estejam simples o suficiente para que possam ser implementadas.

- Após a implementação destas tarefas elas costumam ser combinadas para formar procedimentos mais complexos.

Na Programação Orientada a Objetos há três conceitos fundamentais:

- Encapsulamento ou Ocultação de informação
- Herança
- Polimorfismo

Encapsulamento ou Ocultação de Informação

- Encapsulamento → consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, dos detalhes internos da implementação daquele objeto, que ficam ocultos dos demais objetos.

- Pode-se desejar modificar a implementação de um objeto para melhorar o desempenho ou retirar um erro, dentre outros motivos. O encapsulamento facilita a realização dessas alterações, já que a implementação de um objeto pode ser modificada sem que isso afete as aplicações que o utilizam.

- Na orientação a objetos um objeto encapsula dados, operações, outros objetos, constantes e outras informações.

- A idéia é que os usuários desse objeto possam acessá-lo através de um conjunto de interfaces cuidadosamente documentadas, controladas e padronizadas.

- Através do envio de mensagens pode-se solicitar a esses objetos que façam algo. Por exemplo pode-se enviar a um objeto livro uma mensagem de atualização de preço. Objetos são responsáveis por fornecer informações sobre si mesmos.

- C++, por exemplo, permite a restrição ao acesso a campos e métodos em classes por intermédio de quatro modificadores de acesso: public, private, protected e sem modificador.

    - Public: o campo ou método declarado com este modificador pode ser acessado ou executado a partir de qualquer outra classe.

    - Private: o campo ou método declarado com este modificador só pode ser acessado, modificado ou executado por métodos da mesma classe.

    - Protected: funciona como o modificador private, exceto que classes herdeiras ou derivadas também terão acesso ao campo ou método com este modificador.

Herança

- O mecanismo de herança é apropriado para relações **"é um tipo de"** entre classes.

- A herança permite que uma classe herde atributos e comportamento de outra.

- Considere que em um sistema de controle de consultas médicas dois tipos de pagamento podem ser realizados em uma consulta: através de convênio ou particular

- Todos os pagamentos estão relacionados a uma consulta mas só o pagamento de convênio está relacionado ao convênio correspondente. Já no caso de pagamento particular, deverá ser anotado como foi realizado o pagamento (dinheiro, cheque).

- Usando o mecanismo de herança, podemos declarar as classes PagamentoConvenio e PagamentoParticular como sendo um tipo de Pagamento. Assim:
  - PagamentoConvenio e PagamentoParticular herdam todos os campos e métodos da classe Pagamento.
  - A classe herdeira poderá acrescentar campos e métodos à classe original.

Herança Múltipla

- Imaginar a seguinte situação:
  - Personagem
    - Sofrem transformação espacial
    - Recebem mensagens

Problemas da Herança Múltipla

- Ambigüidade (conflitos de métodos e atributos)
- Topologia (diamond shape / herança virtual)
  - Ex. Mover veículos
- Problemas de Arquitetura

Polimorfismo

- Através do polimorfismo é possível se referir a diferentes derivações de uma classe do mesmo modo, obtendo no entanto o comportamento da classe derivada a que se está referindo.

- Podemos, por exemplo, escreve um método que receba uma instância da classe ObjetoGeometrico e ele é capaz de processar instâncias de qualquer classe que seja sua herdeira, como Retângulo ou Círculo.

Ex:

- Suponha que temos um método imprimir que recebe um uma instância da classe ObjetoGeometrico e calcula a área do objeto e imprime o valor obtido.

- Em tempo de execução poderá ser processada uma instância de um retângulo ou de círculo. Mas no código é feita uma referência a uma instância de ObjetoGeometrico.

IV. MODELAGEM DE ANÁLISE E DE PROJETO

Como já estudamos na 1ª parte do curso, podemos construir os modelos de análise e projeto.

Vamos estudar a UML aprendendo como elaborar esses dois modelos.

MODELO DE ANÁLISE

De acordo com a abordagem de Pressman o modelo de análise é construído na Elaboração, atividade da Engenharia de Requisitos, a partir das informações obtidas nas atividades de Concepção e Levantamento de requisitos. Nessas duas atividades é elaborado o diagrama de casos de uso.

Para elaborar o modelo de análise de acordo com a abordagem orientada a objetos, utilizando a UML, vamos estudar os seguintes diagramas:

- diagrama de casos de uso
- diagrama de classes
- diagrama de *packages*
- diagrama de estados
- diagrama de atividades
- diagrama de seqüência.

MODELO DE PROJETO

O modelo de projeto inclui representações de

- dados,
- arquitetura,
- interface,
- componentes e
- implantação

Este modelo é o principal produto produzido durante o projeto de software.