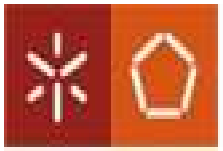


Universidade do Minho
Departamento de Informática

DESENVOLVIMENTO
DE
SISTEMAS SOFTWARE

3º ANO - LEI

2008-2009

OBJECTIVOS

- Associar o desenvolvimento de Sistemas Software aos usuais processos e métodos de Engenharia, neste caso, da Engenharia de Software;
- Modelos, Processos e Métodos;
- Estudo particular e utilização do UP (Unified Process);
- Modelação Orientada aos Objectos e Modelação Visual;
- Estudo da Unified Modeling Language (UML);
- Estudo de uma ferramenta de modelação em UML;

FUNCIONAMENTO

- 2H teóricas e 2H práticas laboratoriais por semana;
- 1 projecto obrigatório, em grupo (3-5), a entregar por fases;
- 1 exame teórico final (com recurso);
- Notas práticas do ano anterior não serão “congeladas”;
- Nota Final:

  Exame [>= 9.0 --- 50%]
  Trabalho Prático [>= 10.0 --- 50%]
  Nota: .5 * Exame + .5 * Trabalho – 2 *(Trabalho - Exame)/Exame [>= 10.0]

- Docentes:

Prof. F. Mário Martins (teóricas) fmm@di.uminho.pt
Prof. António Nestor (práticas) anr@di.uminho.pt
Prof. José Creissac (práticas) jfc@di.uminho.pt

Página da disciplina: sim.di.uminho.pt/disciplinas/dss0809

PLANO - TEÓRICAS

Aula	Data	Matéria Teórica
1	23-09-2008	Introdução à disciplina de DSS. Conteúdos programáticos. Objectivos. Avaliação. Introdução aos Sistemas Software. Sistemas de Informação. Processos de desenvolvimento e Modelos. A importância da Modelação. História da Modelação por Objectos (UML).
2	30-09-2008	Modelação por Objectos em UML. Classificação dos Diagramas por Visões do Sistema. Diagramas de Estrutura. Diagramas de Comportamento. Diagramas de Interação de Componentes. Apresentação da Metodologia de Modelação. Metodologia centrada em Use-Cases. Domínios semânticos.
3	07-10-2008	Estudo dos Use-Cases. Sintaxe e semântica. Include e extend. Cenários: sucesso e excepção. Domínios semânticos (ligação aos use-cases).
4	14-10-2008	Diagramas de Actividade. Swim-lanes. Tratamento sistemático dos use-cases. Refinamento sistemático dos use-cases para diagramas de actividade.
5	21-10-2008	Diagramas de Actividades (continuação)
6	28-10-2008	Diagramas de Sequência.
7	04-11-2008	Diagramas de Sequência (continuação). Modularidade em diagramas de sequência. Diagramas de Colaboração.
8	11-11-2008	Diagrama de Classes e Interfaces.
9	18-11-2008	Diagrama de Classes e Interfaces (continuação). Diagramas de Composição de Estrutura.
10	25-11-2008	Diagramas de Estado.
11	02-12-2008	Packages e Deployment.
12	09-12-2008	Síntese do método proposto. Exemplos.
13	16-12-2008	Outros diagramas UML.
	05-01-2009	RECEPÇÃO DOS PROJECTOS

BIBLIOGRAFIA

- G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.
- J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
- Martin Fowler. *UML Distilled*, 3rd. Ed., Addison-Wesly, 2004.
- Scott W. Wembler, *The Elements of UML 2.0 Style*, Cambridge University Press, 2005.
- R. Pressman. *Engenharia de Software*, 6th. Ed., McGraw Hill, 2005.
- M. Nunes e H. O´ Neill. Fundamental do UML, 2ª Ed., FCA, 2003.
- Notas Teóricas (na página da disciplina).

What is a software engineer?

- Is it simply programming?
- According to the US Bureau of Labor Statistics,
  - → ***Computer systems software engineers*** *primarily write, modify, test, and develop software to meet the needs of a particular customer. They develop software systems for control and automation in manufacturing, business, and other areas.*

IEEE COMPUTER SOCIETY

Slide 9

© Stephen Seidman, "The Path to Software Engineering Professionalism", Join´08, Sept., U. Minho

Programmer vs Software Engineer

Programmer	*Software Engineer*
Writing *code*	Developing *systems*, often large and highly complex
Using techniques learned from individual *experience*	Applying widely accepted techniques based on *proven knowledge*
Building products that *work*	Building products that *you can depend on*

IEEE COMPUTER SOCIETY

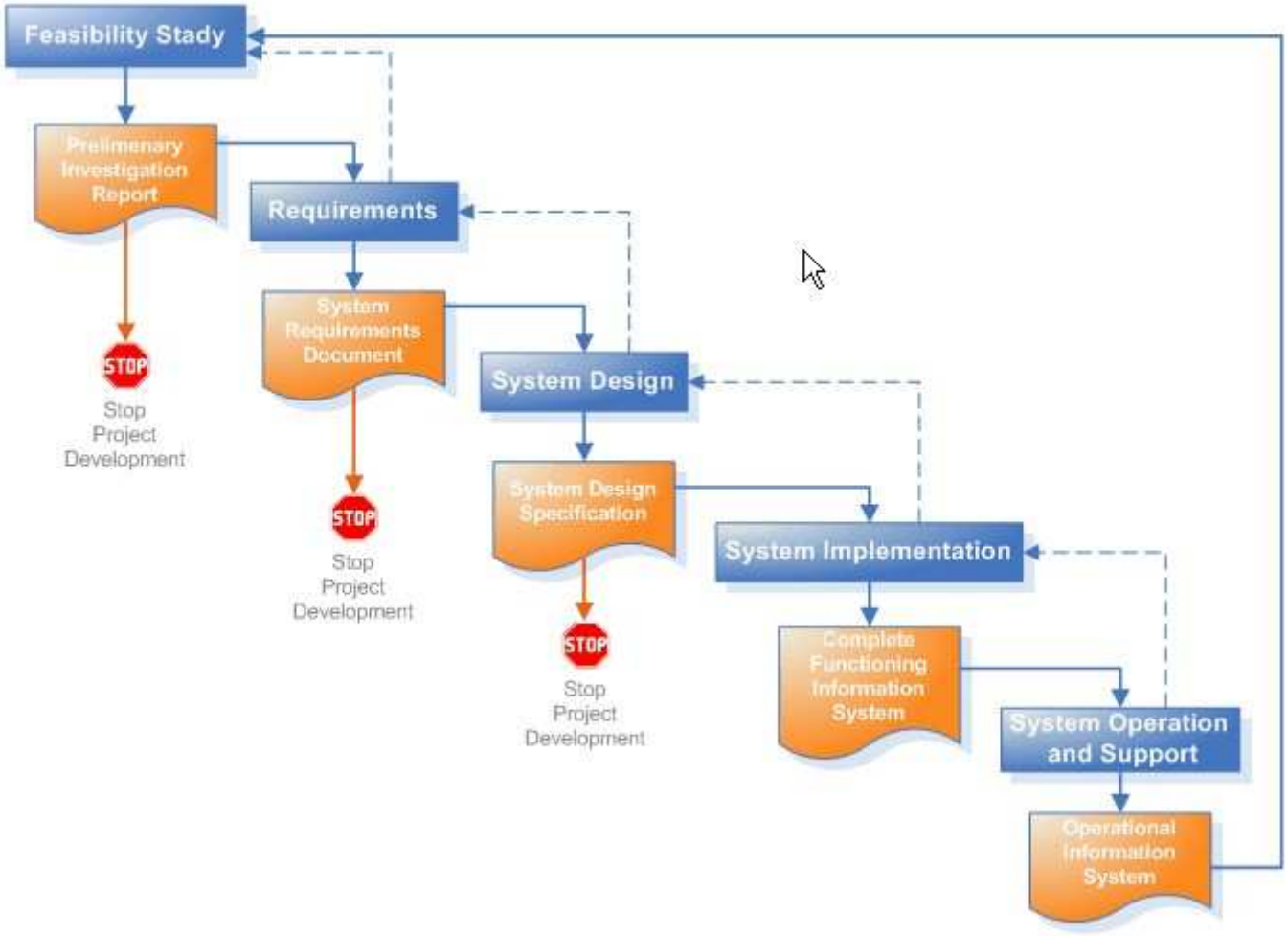
Slide 13

© Stephen Seidman, "The Path to Software Engineering Professionalism", Join´08, Sept., U. Minho

MOTIVAÇÃO - IMPORTÂNCIA

Há etapas típicas, bem definidas, tradicionais até, no projecto de Sistemas Software, ainda que apresentadas de forma diferente.

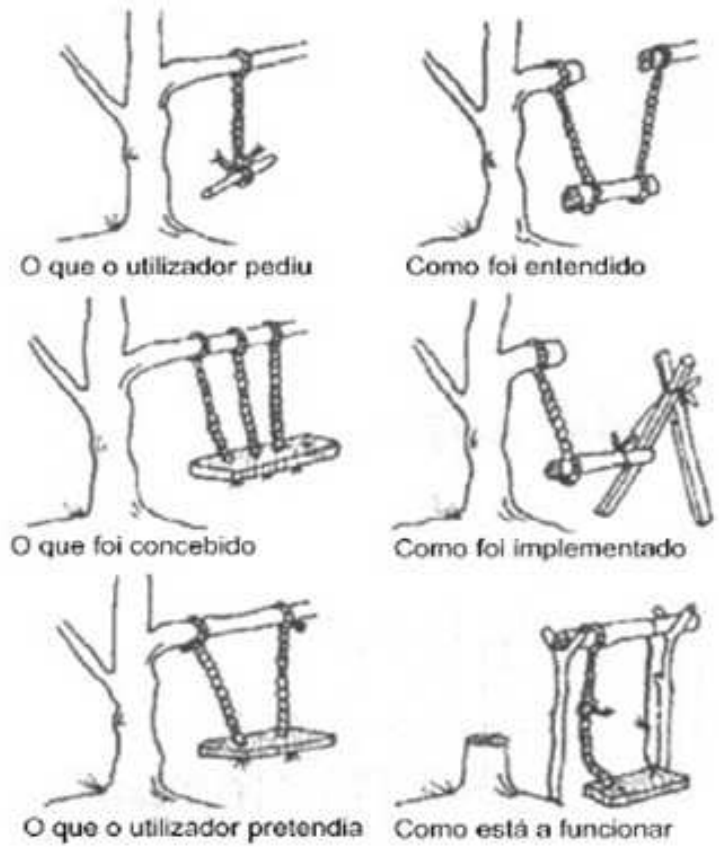
MOTIVAÇÃO - IMPORTÂNCIA



Mas, como podemos ver as fases são exactamente as mesmas.

MOTIVAÇÃO - IMPORTÂNCIA

Desenvolver Sistemas Software não é trivial ...

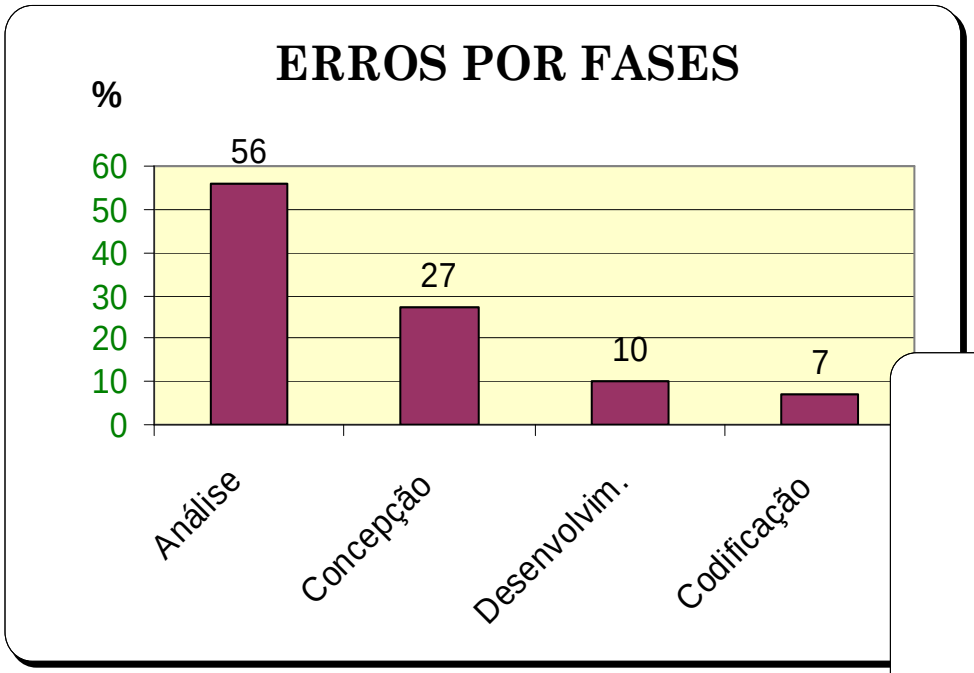


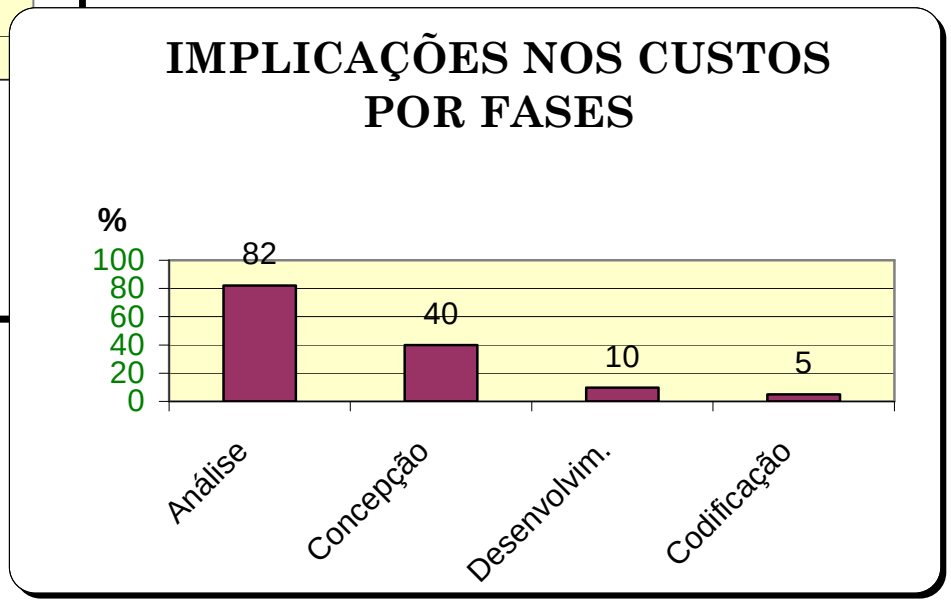
- Riscos associados aos requisitos.
- Riscos tecnológicos.
- Riscos de competência.
- Riscos políticos.

A comunicação entre os clientes, os membros da equipa de projecto, os futuros utilizadores, etc., é um problema que conduz aos maiores erros por má interpretação.

MOTIVAÇÃO - IMPORTÂNCIA

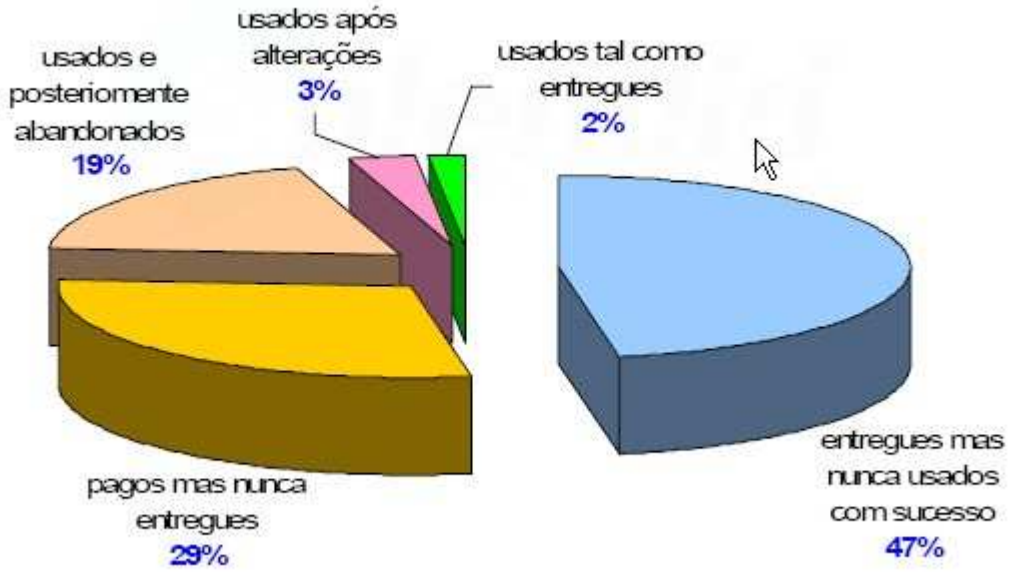
Sendo de salientar os erros de análise e custos implicados !!





MOTIVAÇÃO - IMPORTÂNCIA

E a história tem sido muito pouco satisfatória …

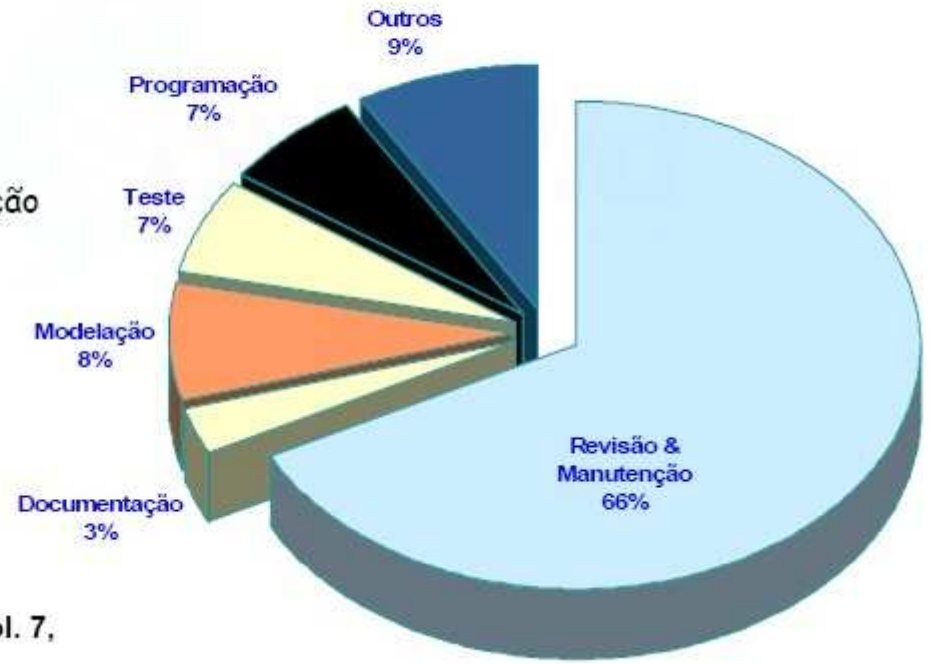


MOTIVAÇÃO - IMPORTÂNCIA

Com custos bens definidos mas enormes

os custos envolvidos num projecto de software

- Modelação
- Programação
- Teste
- Doumentação
- Revisão e Manutenção
- Outros



Source: DP Budget, Vol. 7, No. 12, Dec. 1998

MOTIVAÇÃO - IMPORTÂNCIA

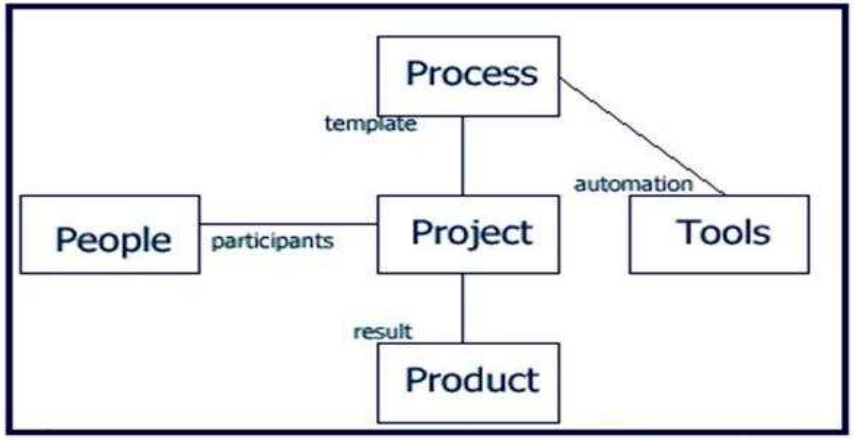
O desenvolvimento de software ainda tem muito de arte e muito pouco de verdadeira Engenharia.

Quando um software de computador é bem-sucedido – quando satisfaz as necessidades das pessoas que o usam, tem desempenho sem falhas por um longo período, é fácil de modificar e ainda mais fácil de usar, ele pode e efetivamente modifica as coisas para melhor. Mas, quando o software falha – quando os seus utilizadores ficam insatisfeitos, quando tem tendência a erros, quando é difícil de modificar e ainda mais difícil de usar –acontecem coisas desagradáveis. Todos nós desejamos construir software que torne as coisas melhores evitando os problemas que espreitam na sombra dos esforços mal sucedidos.
Para obter sucesso, precisamos de disciplina e método quando o software é projetado e construído. Precisamos de uma abordagem de engenharia.

R. Pressman, Engenharia de Software, McGraw Hill, 6ª. Ed., 2005.

Abordagem de Engenharia ao Desenvolvimento de Sistemas Software – Questões importantes

1. Definir um Processo
2. Usar Modelos abstractos do Sistema a conceber e implementar
3. Possuir Métodos rigorosos
4. Usar Ferramentas de apoio ao projecto



MOTIVAÇÃO - IMPORTÂNCIA

Um Processo de Desenvolvimento de Software consiste de uma estruturação das várias disciplinas ou fases que estão contidas na filosofia de desenvolvimento de software adoptada por uma dada organização para o desenvolvimento do produto sistema software.

Mas, fundamentalmente, consiste em definir QUEM no projecto está a fazer O QUÊ, QUANDO o deve fazer e DURANTE quanto tempo, e como se devem atingir os objectivos definidos.

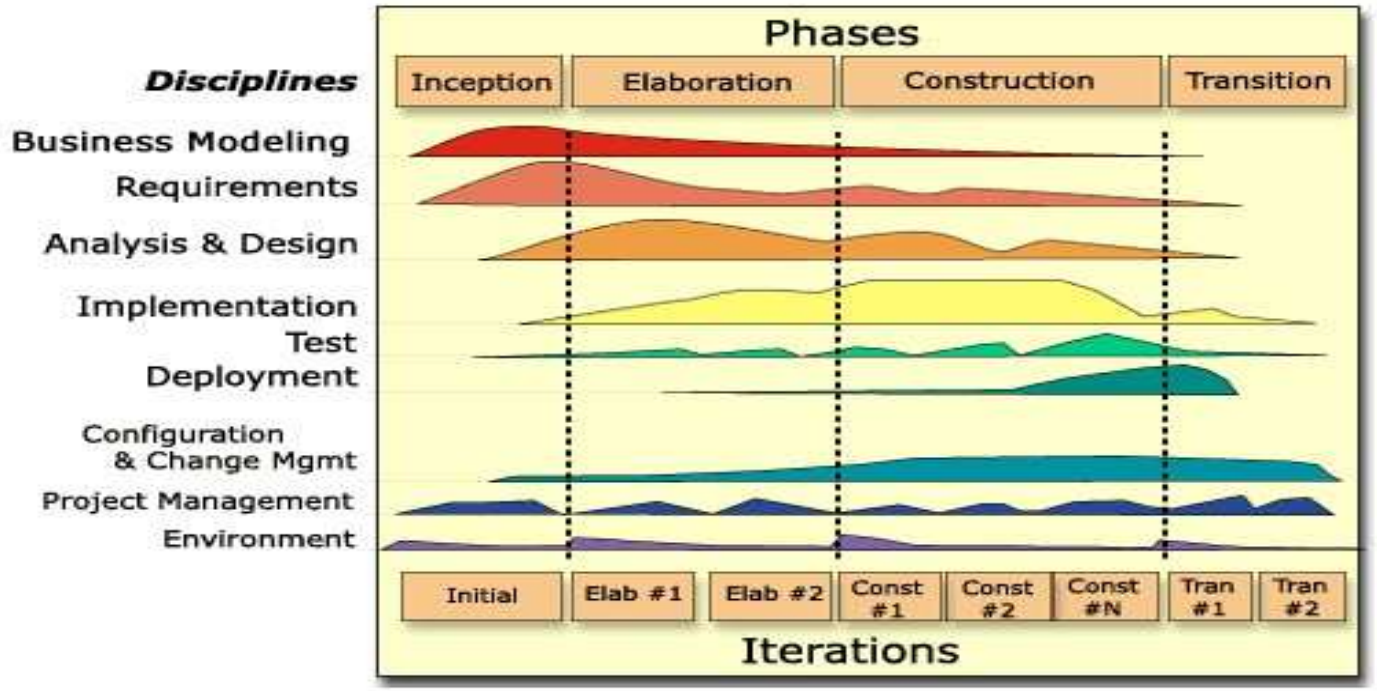
MOTIVAÇÃO - IMPORTÂNCIA

A nossa abordagem de Engenharia ao Desenvolvimento de Sistemas Software, passa por algumas ideias fundamentais, a saber:

- Adoptar o **Rational Unified Process** (RUP) como processo de base para o desenvolvimento;
- Seguindo o RUP, apostar na **Modelação Orientada aos Objectos**;
- Seguindo o RUP, usar **UML** (standard da OMG), como notação de modelação;
- Definir alguma **metodologia** na utilização dos modelos UML.
- Realizar o **desenvolvimento integrado e coerente de todas as camadas** do sistema software, desde a **camada de dados** até à **camada interactiva**.

ABORDAGEM EM DSS

Tal como noutras áreas, talvez a táctica esteja correcta, mas são a "visão" (a estratégia) e a "dinâmica" (o processo) as questões que são fundamentais. Seguiremos uma estratégia Orientada aos Objectos e uma dinâmica parcialmente alinhada pelo RUP ("Rational Unified Process") mas também pelos processos AGILE, não rígidos.



RUP

UML: Unified Modeling Language

Objectivos: Modelação, Comunicação, Teste e Documentação das várias facetas/aspectos de um sistema *software intensive*

Linguagem de modelação visual (+ algum texto)

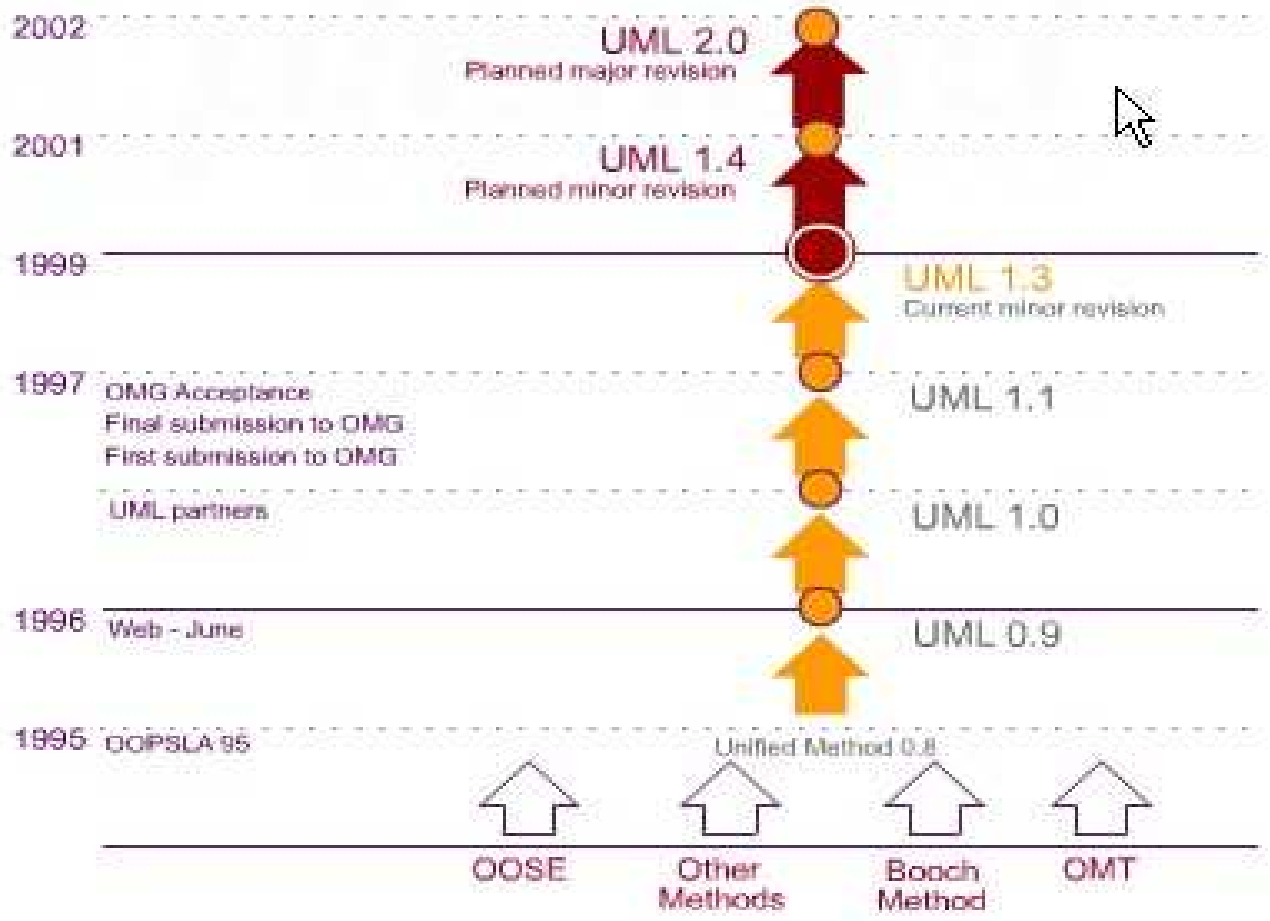
É um standard de facto

v1.1 – 1997
v1.3 – 1999
v2.0 – 2005

Não é metodologia/processo: não diz quem deve fazer o quê, quando e como; É rigorosa mas não formal; Pode ser usada por diferentes metodologias; É, hoje, a base da designada Model Driven Software Engineering (MDSE).

UML - EVOLUÇÃO

Timeline



Diagramas de UML2

Diagramas de Estrutura

- Diagrama de Objectos
- Diagrama de Classes
- Diagrama de Componentes
- Diagrama de Instalação
- Diagrama de Pacotes
- Diagrama de Estrutura Composta

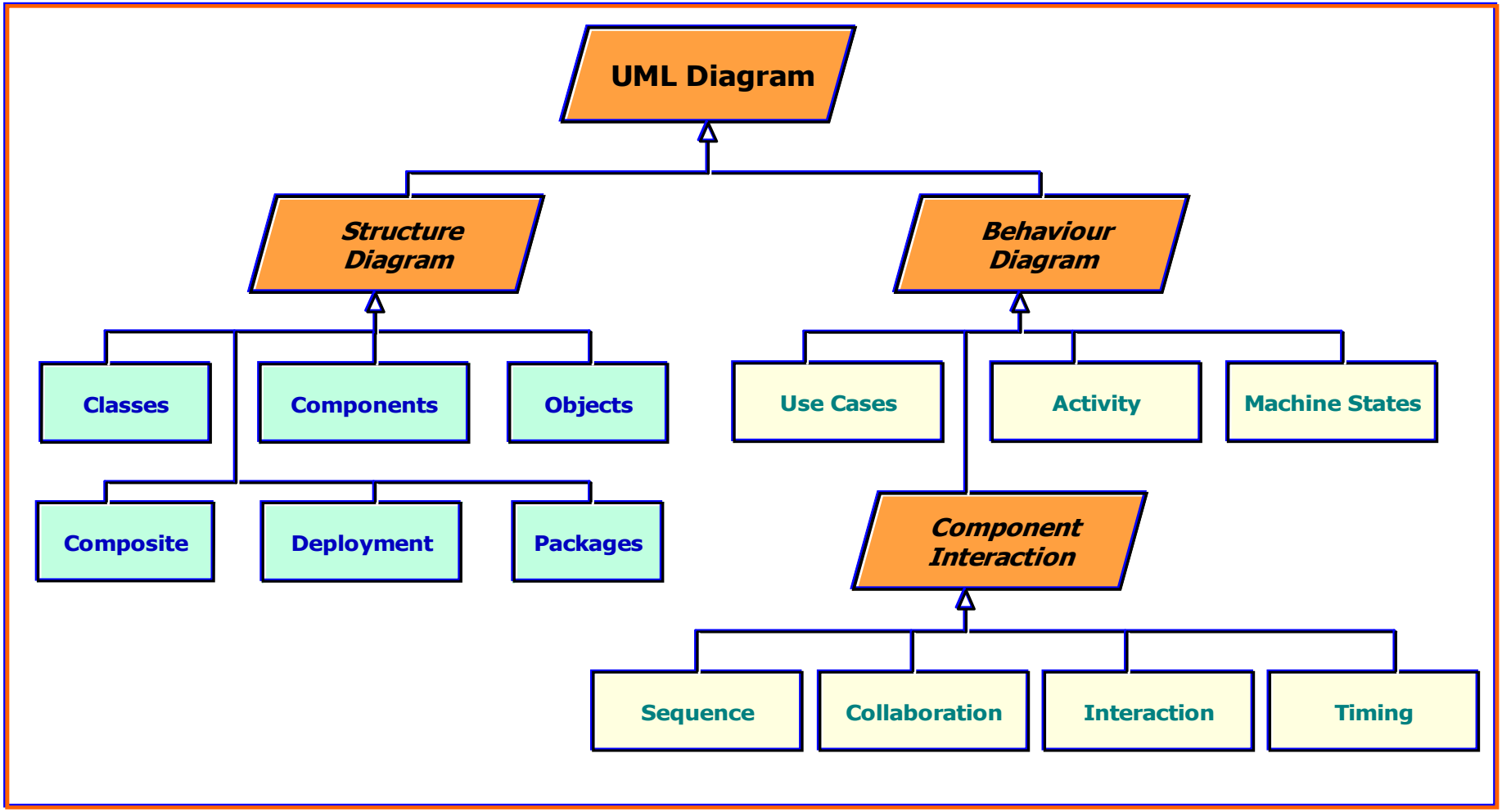
Diagramas de Comportamento

- Diagrama de Casos de Uso
- Diagrama de Actividade
- Diagrama de Estados

Diagramas de Interacção

- Diagrama de Sequência
- Diagrama de Interacção
- Diagrama de Colaboração
- Diagrama de Temporização

UML: DIAGRAMAS e MODELOS



VISÕES FUNDAMENTAIS: ESTRUTURAL e COMPORTAMENTAL

E DEPOIS METODOLOGIA …

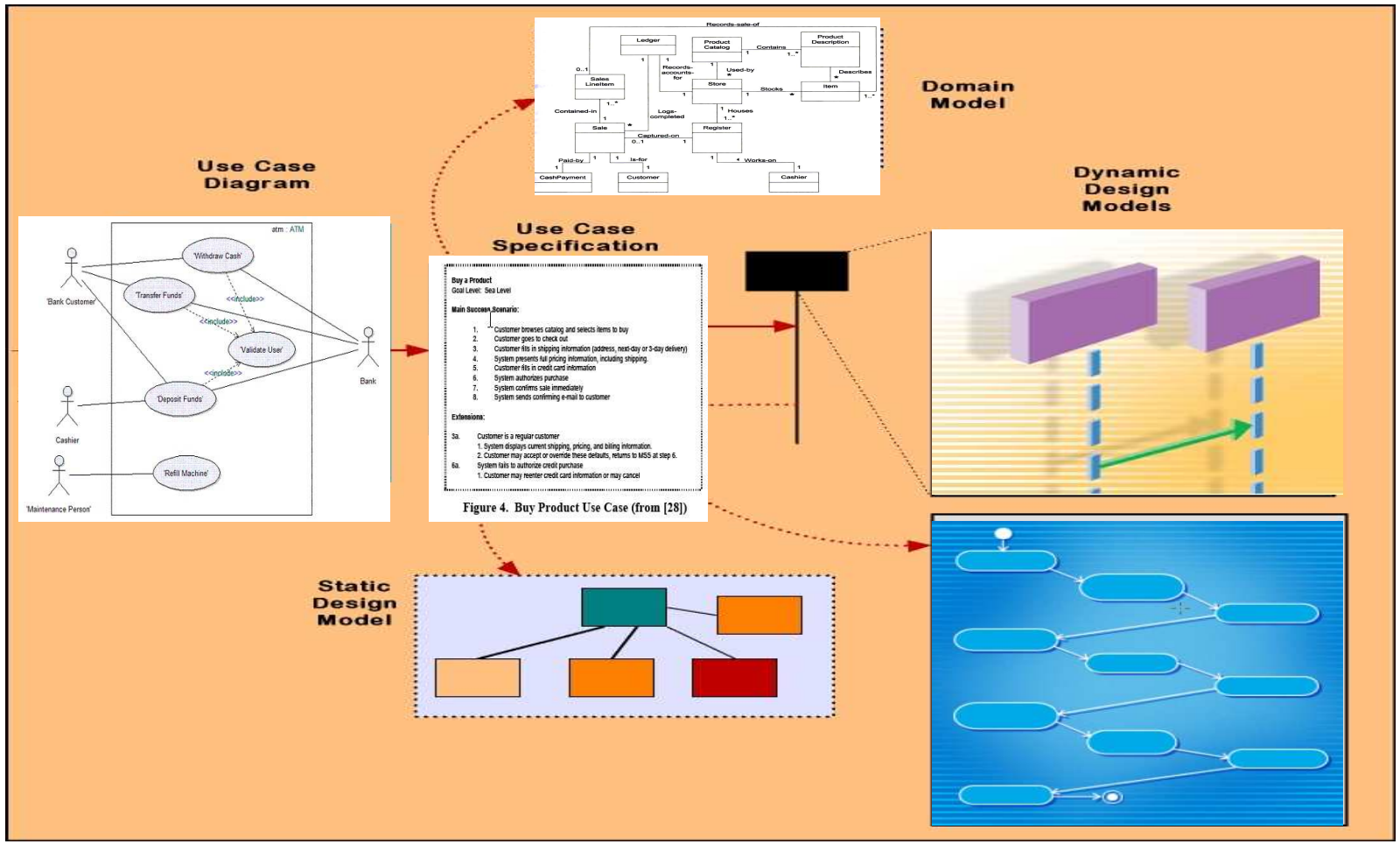


Figure 4. Buy Product Use Case (from [28])

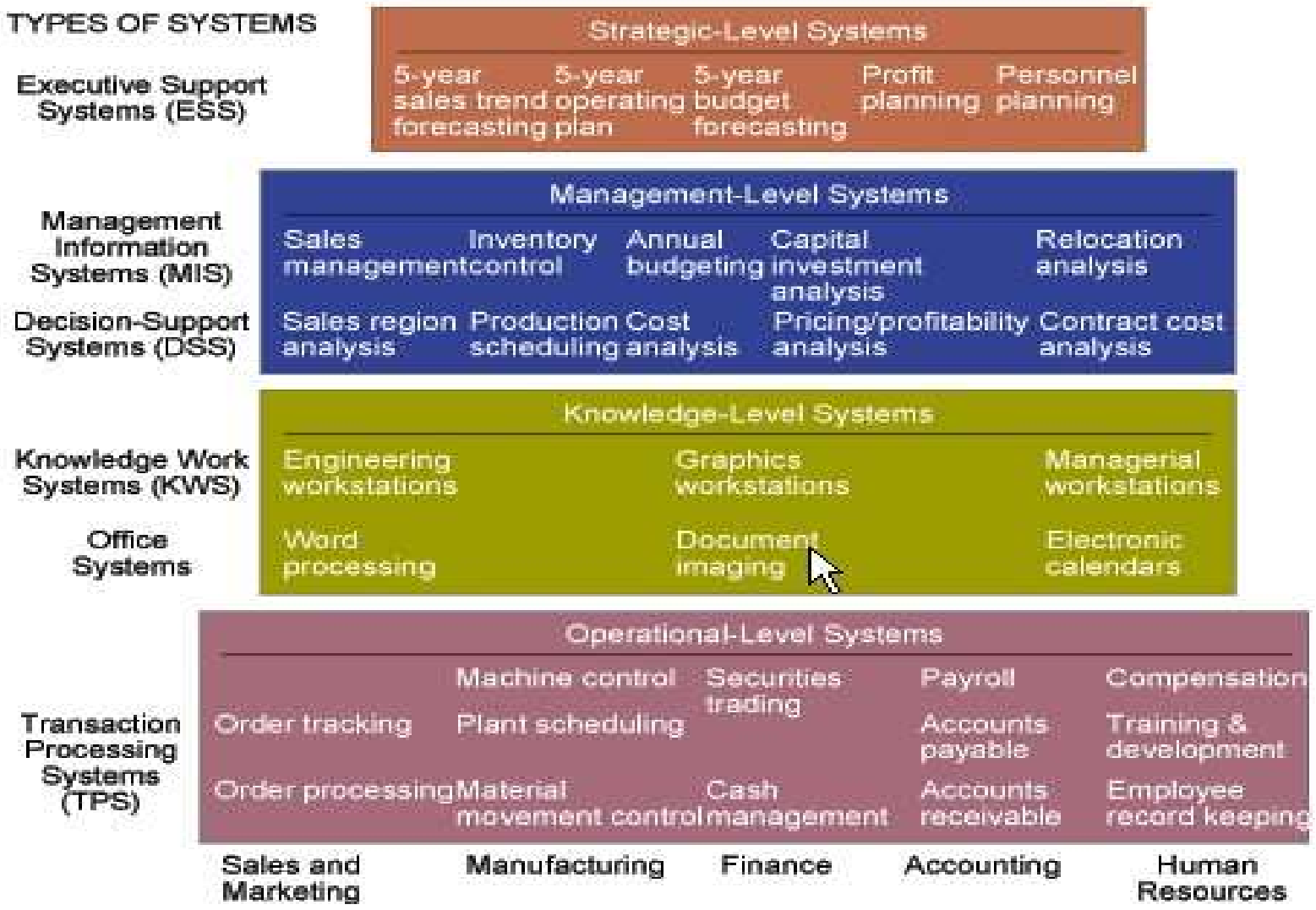
Domain Model + Use Case Model são depois refinados sistematicamente nos outros modelos, estruturais e comportamentais, idealmente diferenciando objectos que são de camadas distintas (dados, computacional, negócio e IU).

O QUE É UM SISTEMA DE INFORMAÇÃO ?

Um Sistema de Informação é hoje entendido como um sistema computacional, ou seja, um conjunto de componentes de hardware e de software, em geral "software intensive", ou seja, fundamentalmente com a "inteligência" residente no software e a capacidade de processamento residente no hardware e na sua respectiva arquitectura, mas que tem por objectivo crucial fornecer um conjunto de procedimentos para o registo, o tratamento, a análise e a apropriada disponibilização de informação relevante para os diferentes níveis de responsabilidade de gestão e decisão, típicas de uma organização moderna.

Os diferentes níveis de responsabilidade e de necessidade de informação/conhecimento dentro das organizações, e, em consequência, os diferentes tipos de SI necessários às organizações, estão hoje muito bem caracterizados.

CONHECIMENTO CLÁSSICO



CONHECIMENTO CLÁSSICO

Fig 1.2
Relation of information systems to levels of organization

As actividades de gestão e de decisão, influenciam e modificam as actividades ao nível funcional (de processamento de informação), porque delas necessitam e dependem (cf. melhor conhecimento).

CONHECIMENTO CLÁSSICO

Em resumo, qualquer que seja o seu tipo ou missão, os SI são criados sempre com os seguintes objectivos em mente:

- Apenas existem para auxiliar (em eficácia e eficiência) a organização cliente;
- Quem deve definir requisitos e objectivos é a organização cliente, mas os projectistas podem "dar ideias" durante a fase de concepção;
- Antes de criar o SI os projectistas devem tentar compreender o melhor possível a organização, o seu "negócio" e a sua estrutura de gestão;
- Devem também compreender de forma rápida quais as pessoas dentro da organização que vão ser os reais utilizadores do SI (cf. os 3 níveis que se apresentaram antes);
- Devem compreender qual a informação relevante que flui na organização e que parte dela vai passar a integrar o SI (cf. "domain model");
- Finalmente, é necessário compreender o problema (requisitos de todos os tipos, funcionais ou não) antes de desenvolver a solução.

CONHECIMENTO CLÁSSICO

Compreendida a missão (desenvolver Sistemas Software úteis e eficazes usando procedimentos de Engenharia), compreendido o enquadramento (para as organizações, sejam de comércio, serviços, ou indústria), e conhecendo até "alguma história" de insucesso, o que precisamos de saber e aprender para que se possa inverter tal história e, de facto, deixar a arte e enveredar pela engenharia e, assim, por um maior rigor ?

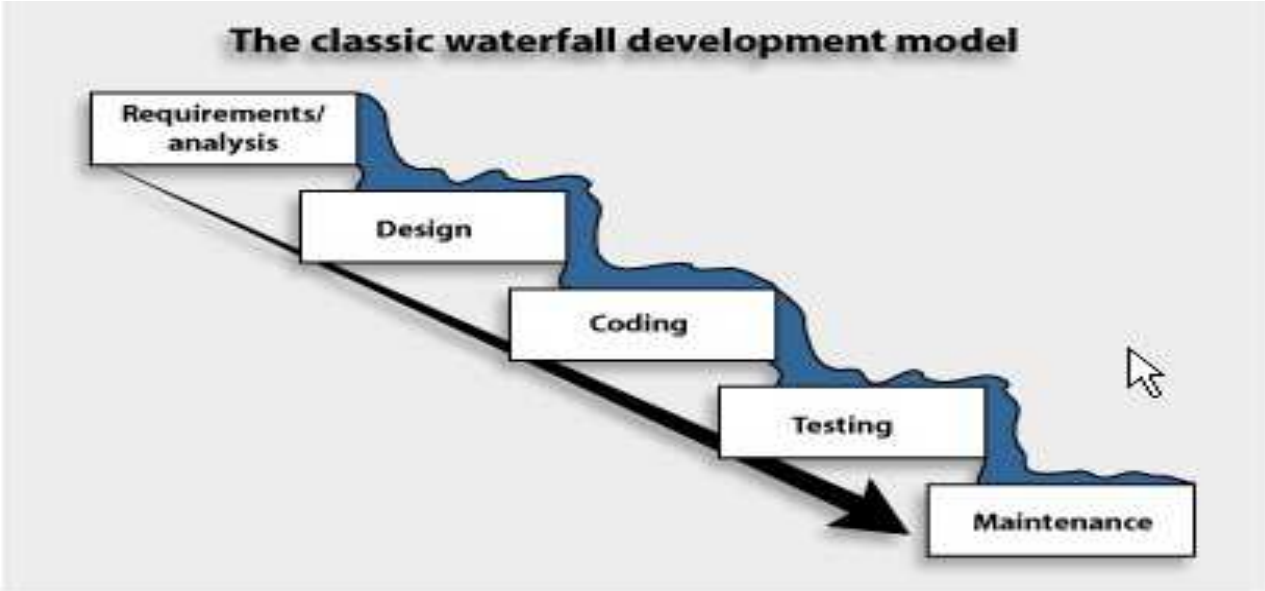
1.- Adoptar um Processo de desenvolvimento de Sistemas Software que nos garanta tais metas, em especial depois de conhecermos a história dos processos desenvolvidos; O processo adoptado deverá dar, justificadamente, muitas mais garantias de sucesso no desenvolvimento dos actuais e futuros SI (ou Sistemas Software);

2.- Adoptar/Criar Métodos, ou seja, regras sobre "como fazer", desde como fazer a captura de requisitos até como fazer a instalação, os testes e a manutenção;

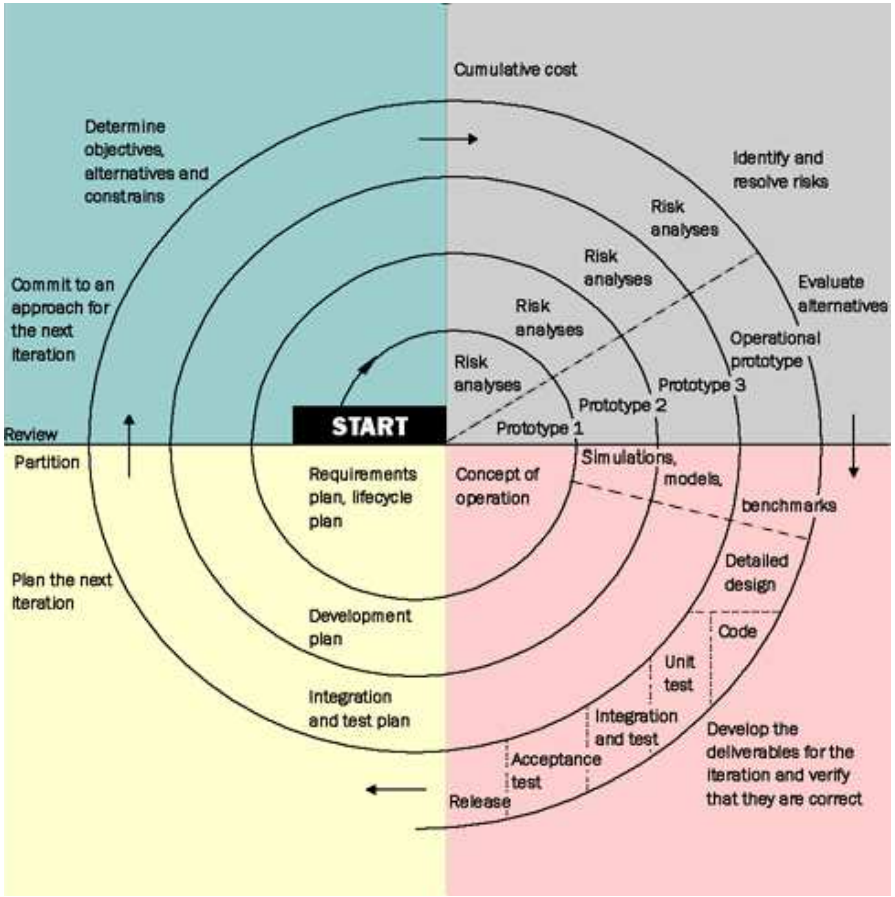
3.- Adoptar Ferramentas que representam o suporte automático ou semi-automático aos processos e aos métodos.

MODELOS DO PROCESSO DE SOFTWARE

São todos relativamente consensuais quanto às várias fases do processo, apenas diferem quanto à dinâmica de execução de tais fases. Vamos definir tais fases com base num modelo mais antigos, o modelo de desenvolvimento sequencial puro, por isso designado em cascata ou "waterfall".



Assume pura sequência de fases, sem retorno, e que tudo corre bem logo à 1ª : é irrealista !!

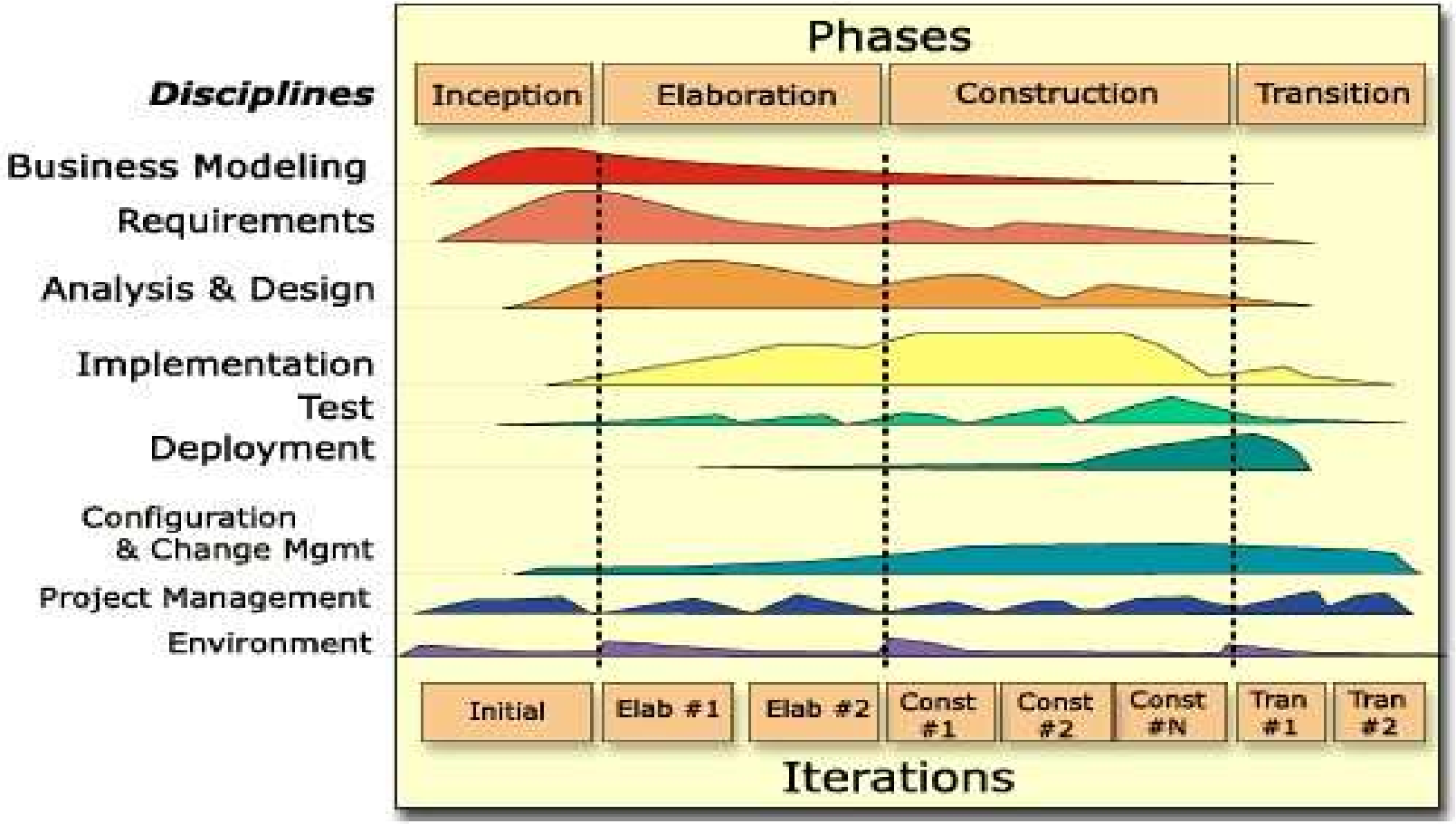


Modelo em Espiral

As várias fases de projecto são realizadas de forma iterativa, ou seja, procura-se garantir que só se transita para uma fase mais estável depois de fixadas as fases anteriores.

Não é um modelo fácil, porque impõe uma estrututura e uma dinâmica de modificação de projecto, em geral, não suportada pela metodologia e pelas ferramentas actuais.

CONHECIMENTO CLÁSSICO



O RUP (Rational Unified Process) será para nós a abordagem seguir.

AS DISCIPLINAS SÂO CONSENSUAIS:

CAPTURA DE REQUISITOS: Depois de os estudos de viabilidade, etc., em geral não realizados, indicarem que o Sistema é viável e útil para a organização, haverá que realizar a captura (percepção e entendimento) de todos os requisitos do mesmo, quer do ponto de vista funcional (o que deve ser capaz de fazer, funções e serviços), quer de outros pontos de vista não funcionais (qualidades e restrições de desenvolvimento, tais como compatibildade de tecnologias, prazos de entrega, etc.).

ANÁLISE E MODELAÇÃO CONCEPTUAL: No dia a dia, usamos muitas vezes sistemas abstractos que nos ajudam a compreender, de forma sintéctica, partes específicas dos complexos sistemas físicos. Por exemplo, um Mapa do Metro de uma dada cidade (abstracção), auxilia-nos a relacionar as diferentes linhas do metro com as ruas da cidade (sistemas físicos) que o mapa representa. Mas o mapa é, naturalmente apenas um sistema abstracto, um modelo, uma síntese da realidade.

CONHECIMENTO CLÁSSICO

AS DISCIPLINAS SÂO CONSENSUAIS:

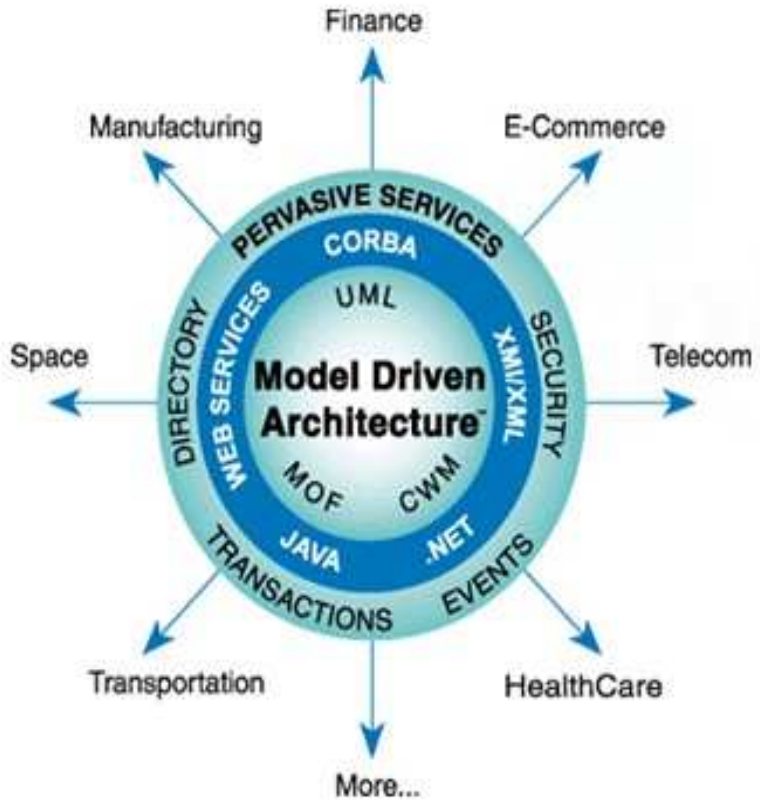
CONCEPÇÃO/DESIGN/ESPECIFICAÇÃO: Fase na qual, após todas as análises realizadas na fase anterior, se determina, "escreve", de forma mais ou menos rigorosa, o que o Sistema Software "deve ser capaz de fazer", informação que é fundamental para os programadores, assim sejam eles capazes de compreender as especificações que lhes são passadas. Na prática, nada disto acontece …

ETC: …

Mas há algumas perspectivas mais concretas, em especial baseadas numa abordagem "por objectos", a todo o processo de desenvolvimento de software, tal como proposto pelo OMG ("Object Management Group"), de credibilidade universal.

OBJECT MANAGEMENT GROUP

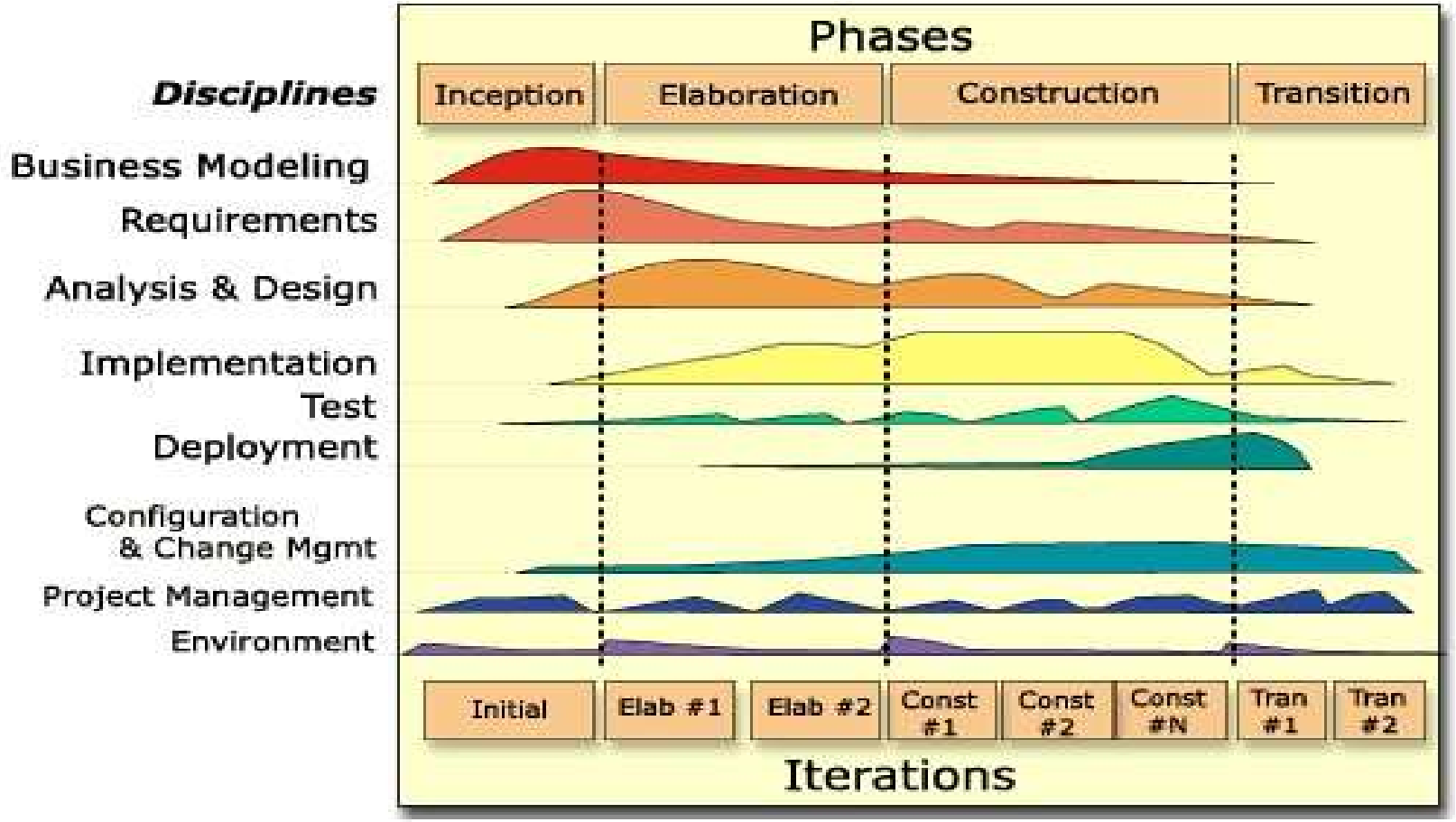
OMG Model Driven Architecture



How Systems Will Be Built

OMG's Model Driven Architecture ® (MDA ®) provides an open, vendor-neutral approach to the challenge of business and technology change. Based on OMG's established standards, the MDA separates business and application logic from underlying platform technology. Platform-independent models of an application or integrated system's business functionality and behavior, built using UML and the other associated OMG modeling standards, can be realized through the MDA on virtually any platform, open or proprietary, including Web Services, .NET, CORBA ®, J2EE, and others. These platform-independent models document the business functionality and behavior of an application separate from the technology-specific code that implements it, insulating the core of the application from technology and its relentless churn cycle while enabling interoperability both within and across platform boundaries. No longer tied to each other, the business and technical aspects of an application or integrated system can each evolve at its own pace - business logic responding to business need, and technology taking advantage of new developments - as the business requires.

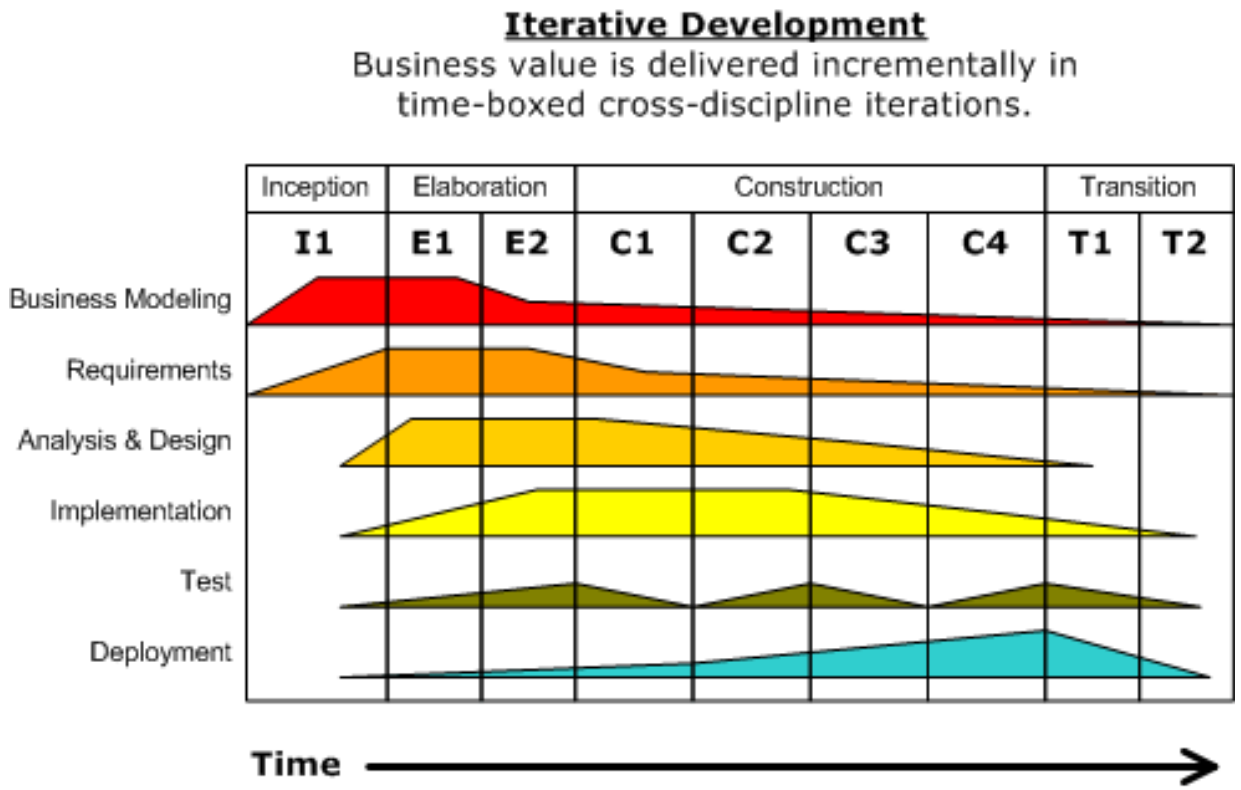
O QUE DIZ O RUP …



O RUP (Rational Unified Process) será para nós a abordagem seguir.

O QUE DIZ O RUP …

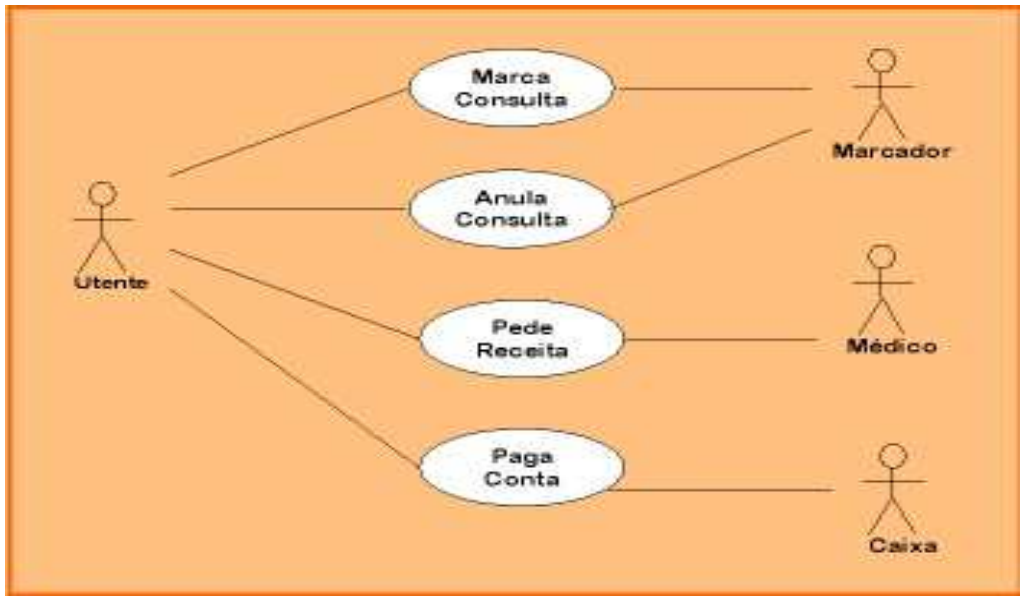
- **Processo de Desenvolvimento de Software que é iterativo e incremental**
  - As várias fases são divididas em séries de mini-fases que correspondem a sucessivas versões mais completas dos sistemas.



O QUE DIZ O RUP …

RUP é "use-case driven"

- Os "Use Cases" (Casos de Uso) são instrumentos fundamentais do processo, porque na fase de captura de requisitos são os "use cases" que irão representar os requisitos de funcionalidade do sistema e definir as interacções com o sistema, necessárias para deste se obter tal funcionalidade.



O QUE DIZ O RUP …

RUP é "use-case driven"

"Use case driven" means writing the user manual first, then writing the code. This practice reinforces the fundamental notion that a system must conform to the needs of the users, instead of your users conforming to the system. [Doug 2001]

Use Case: Fazer Telefonema
Pré-condição: Telefone ligado e em descanso
Comportamento normal:
1. Utilizador marca número e pressiona OK
2. Telefone transmite sinal de chamada
3. Utilizador aguarda
4. Telefone estabelece ligação
5. Utilizador fala
6. Utilizador pressiona tecla C
7. Telefone desliga chamada

Comportamento Alternativo:
3. Telefone Transmite sinal de ocupado
4. Utilizador pressiona tecla C
5. Telefone cancela chamada

Comportamento Alternativo:
3. Telefone cancela chamada

Pós-condição: Telefone ligado e em descanso

Especificação da sequência de interacções que são necessárias para se obter o serviço

O QUE DIZ O RUP …

- **RUP baseia-se na criação de múltiplos modelos usando UML**

Um modelo é uma representação simplificada de um aspecto da realidade existente ou a construir, com um propósito específico;

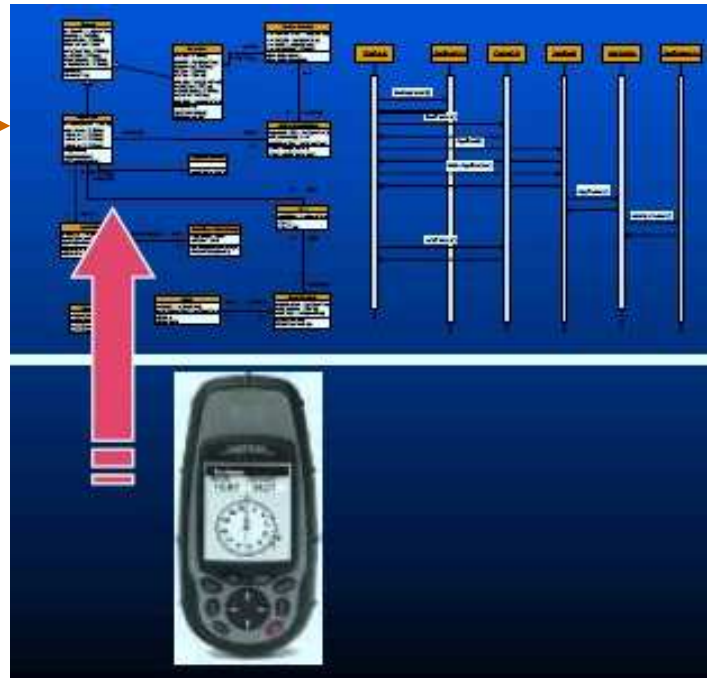
Específico da engenharia: modelar qq. coisa ainda não existente para melhor a criar.

Estrutura + Comportamento

Nota: 1 aspecto => 1..* modelos

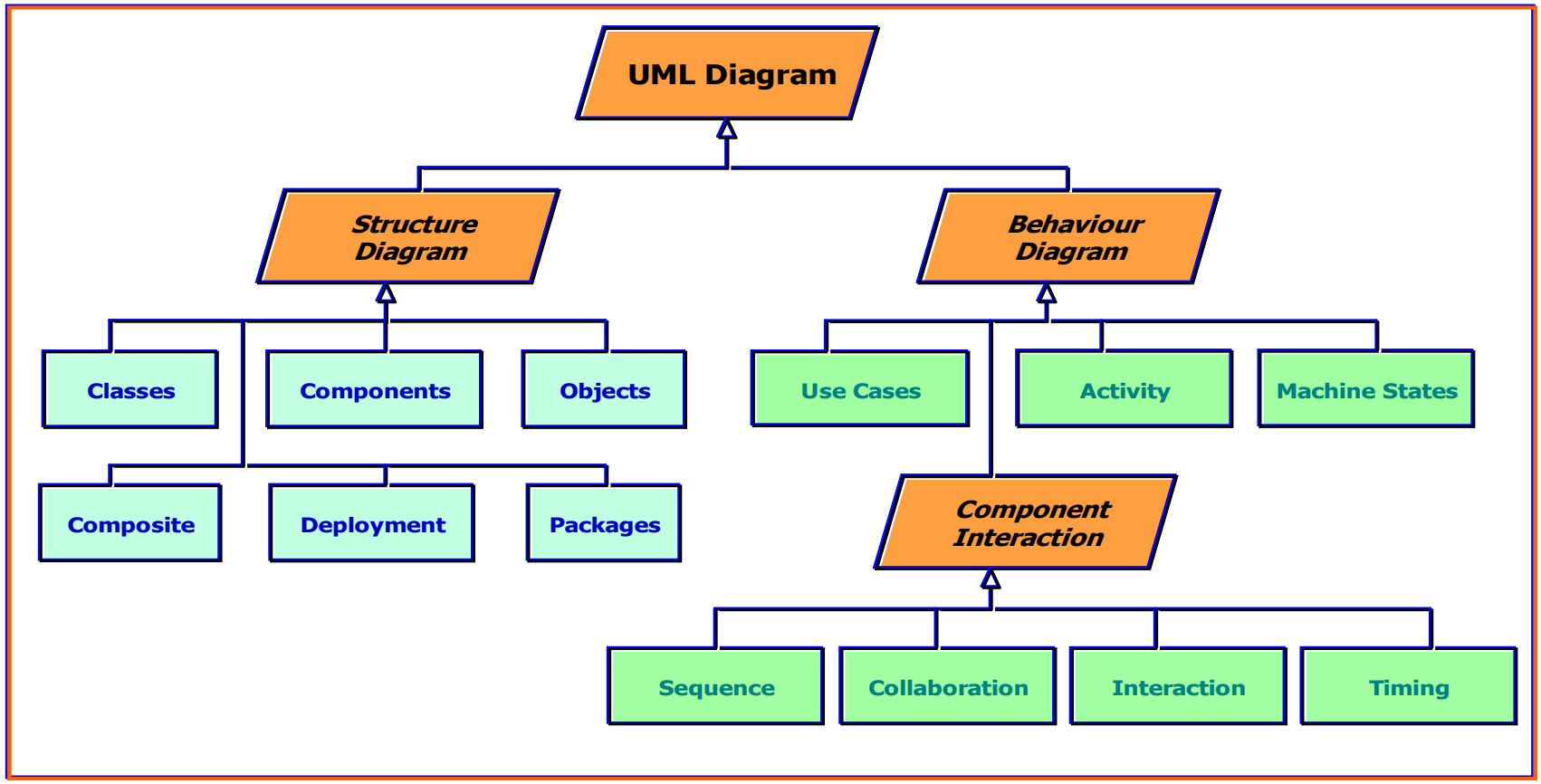
Mm: Espaço de modelação

Mr: Mundo real

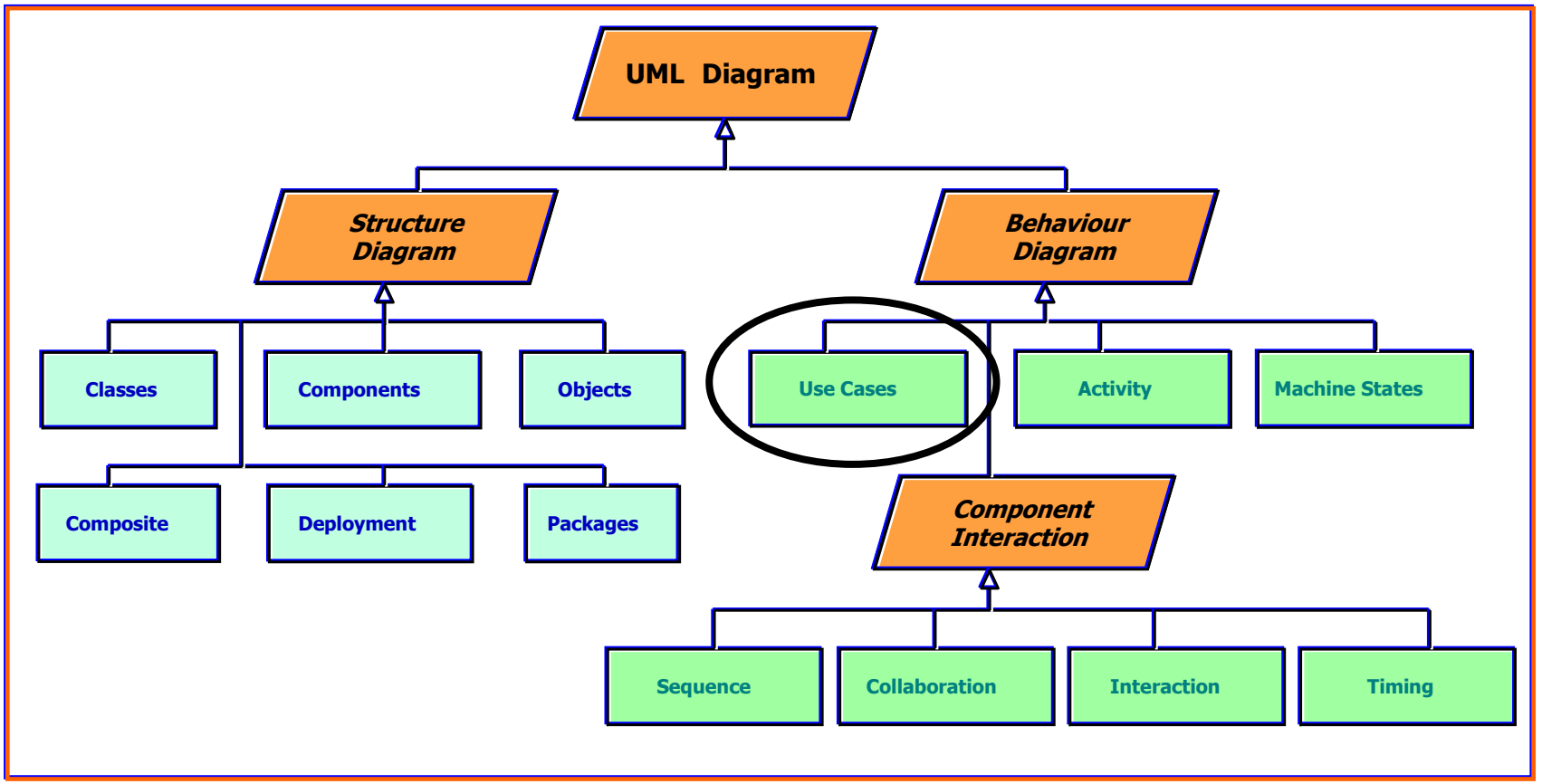


Modelos são conjuntos de diagramas + texto;

Diagramas são vistas de um modelo;

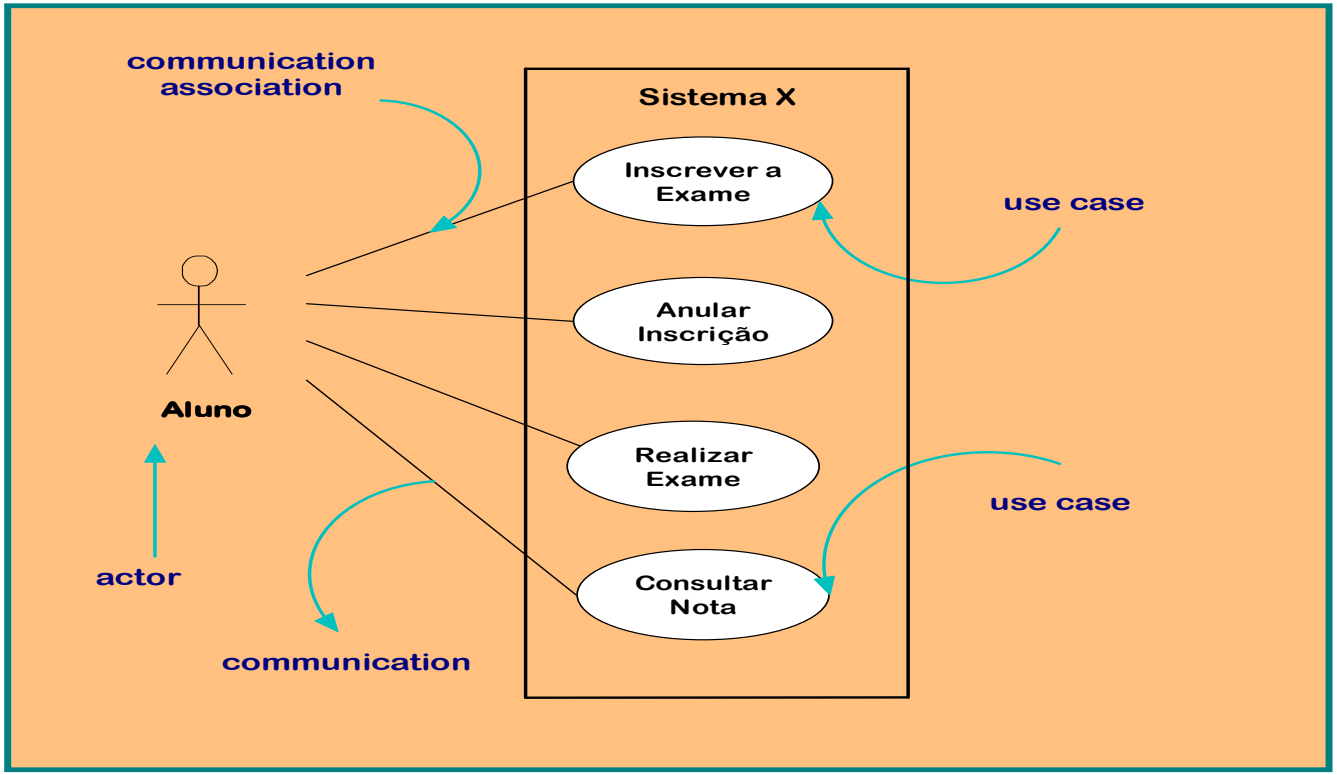


USE CASES

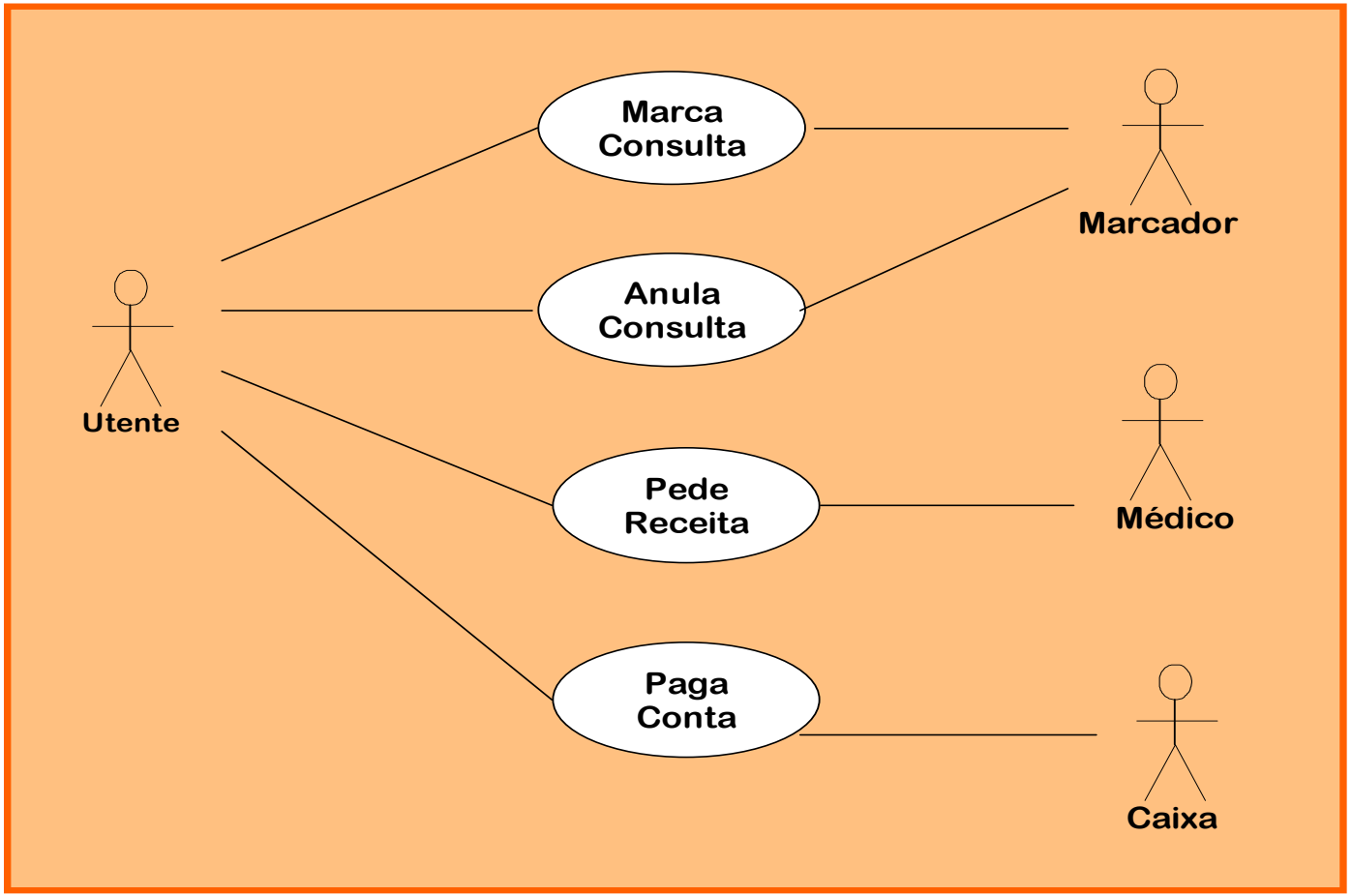

UML Diagram
Structure Diagram
Behaviour Diagram
Classes
Components
Objects
Use Cases
Activity
Machine States
Composite
Deployment
Packages
Component Interaction
Sequence
Collaboration
Interaction
Timing

USE CASES: Elementos

- Diagramas de Use Case especificam O QUE um sistema deve funcionalmente fazer, tal como observável de um "ponto de vista" externo (humano ou não, p.e., outro sistema).
- Conceitos: Actores, Use Cases e Sujeito (sistema).

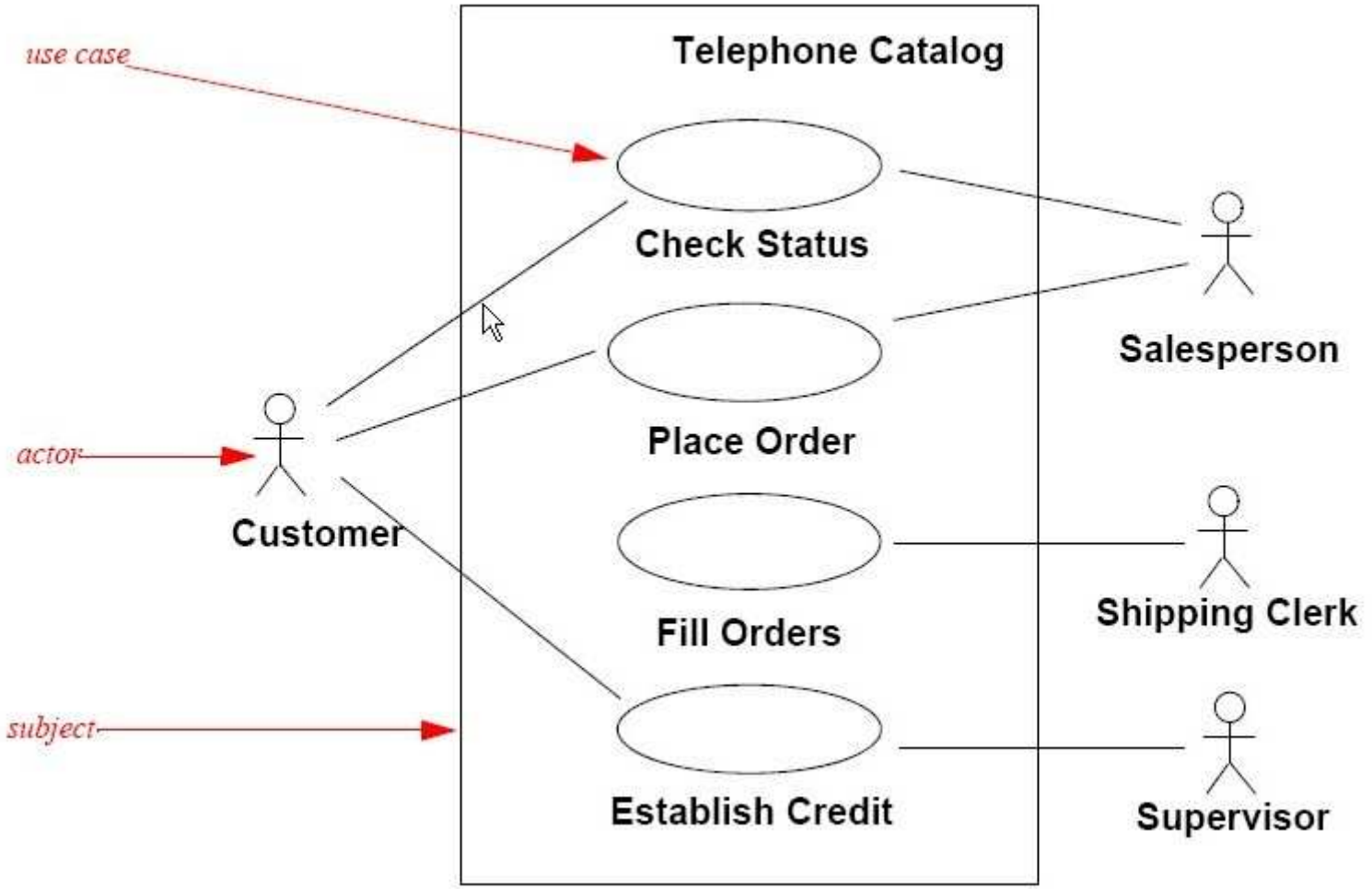


USE CASES

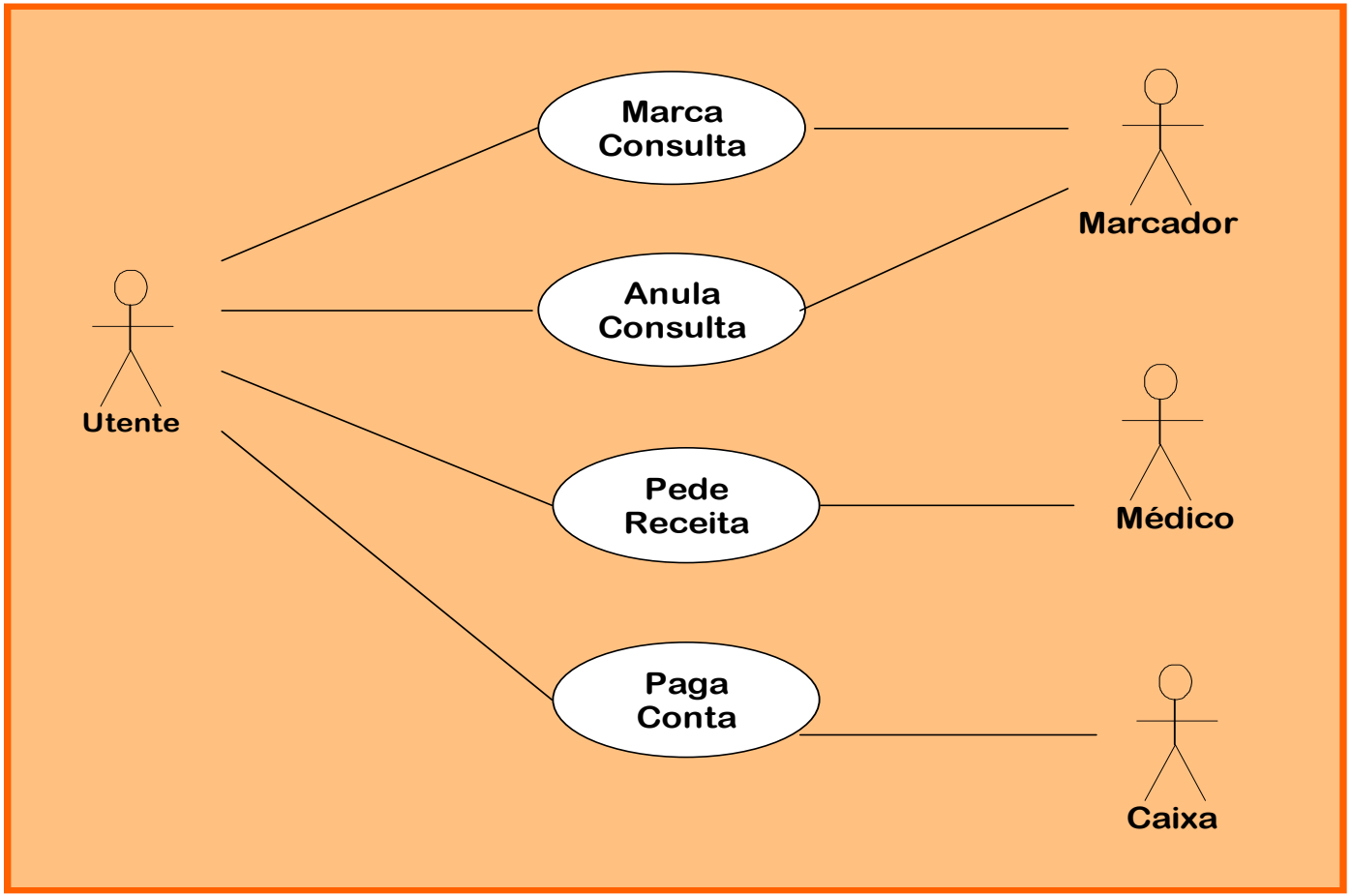


Use Case = **Uma funcionalidade do sistema (software ou não).**

USE CASES

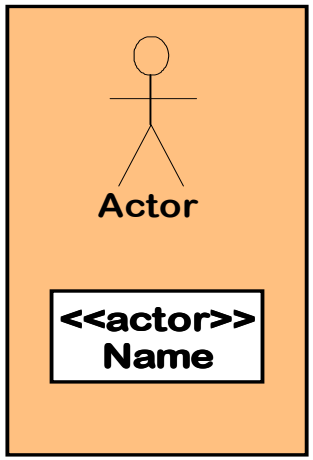

use case
Telephone Catalog
Check Status
Salesperson
Place Order
actor
Customer
Fill Orders
Shipping Clerk
subject
Establish Credit
Supervisor

USE CASES



Use Case = **Uma funcionalidade do sistema (software ou não).**

USE CASES



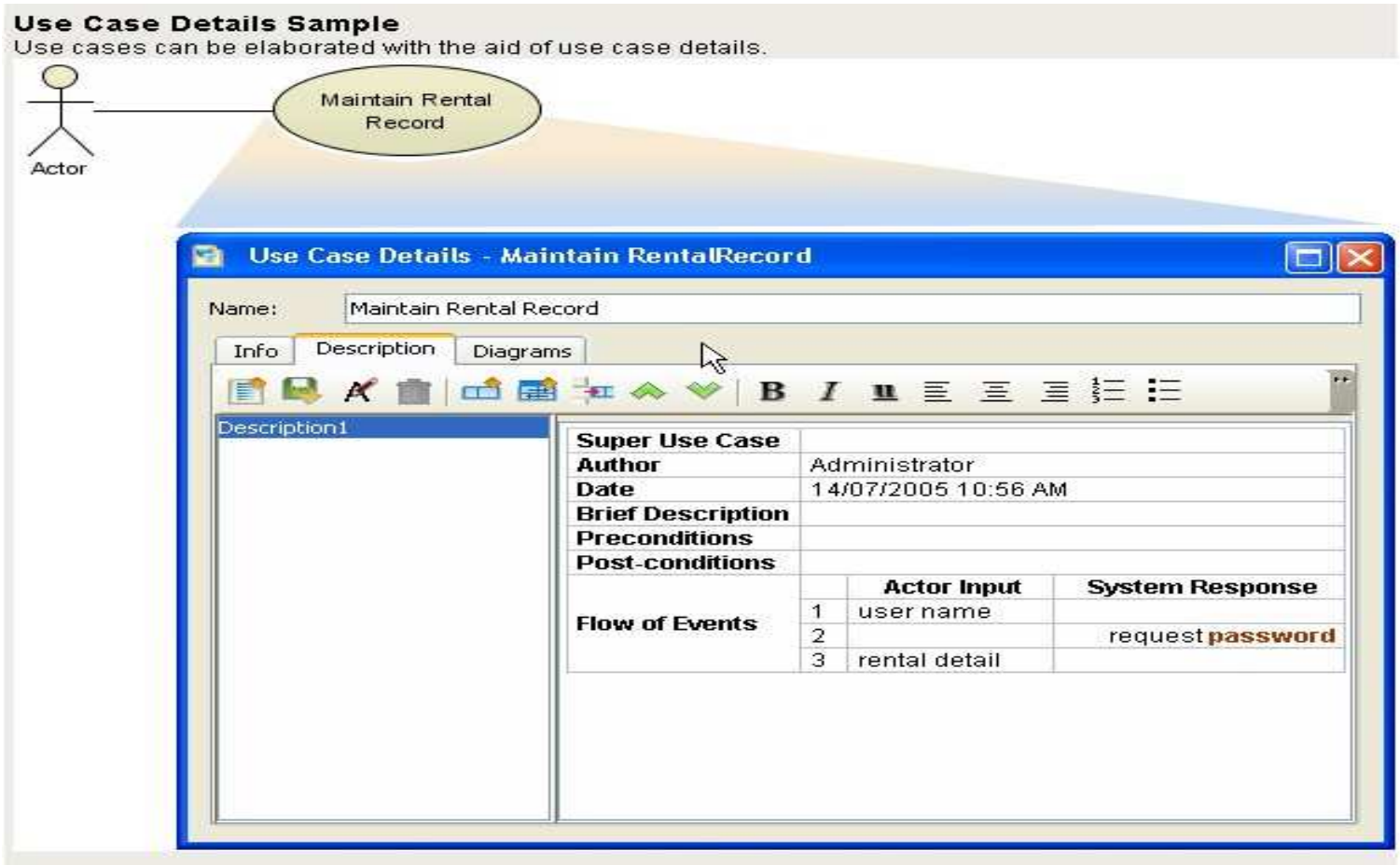
▣ Um actor representa um papel ("role") que "alguém" ou qualquer "coisa" externa ao sistema representa, ao ser responsável por iniciar os eventos necessários (interagir) para que uma determinada tarefa se cumpra. Uma pessoa pode corresponder a vários actores e vice-versa.

▣ Um use case é um resumo de todos os possíveis cenários para a realização de uma dada tarefa ou obtenção de um dado objectivo (goal). É comum um use case ter vários actores envolvidos.

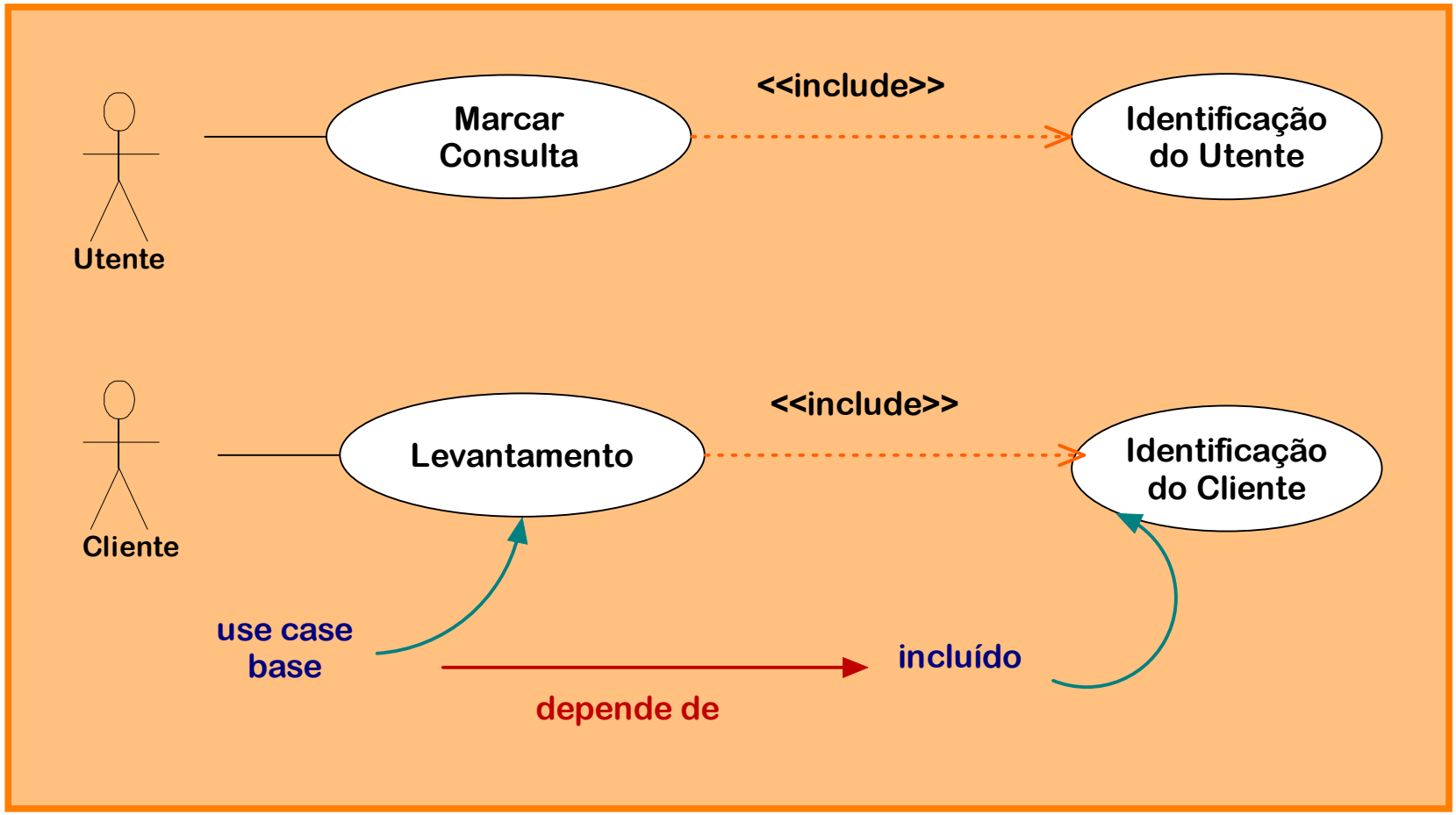
▣ O que falta? Modularidade, estruturação e reutilização !

▣ Que relações são típicas da abordagem OO ?

Não esquecer que cada use case diagramático irá ser depois especificado textualmente passo a passo (exº em Visual Paradigm).

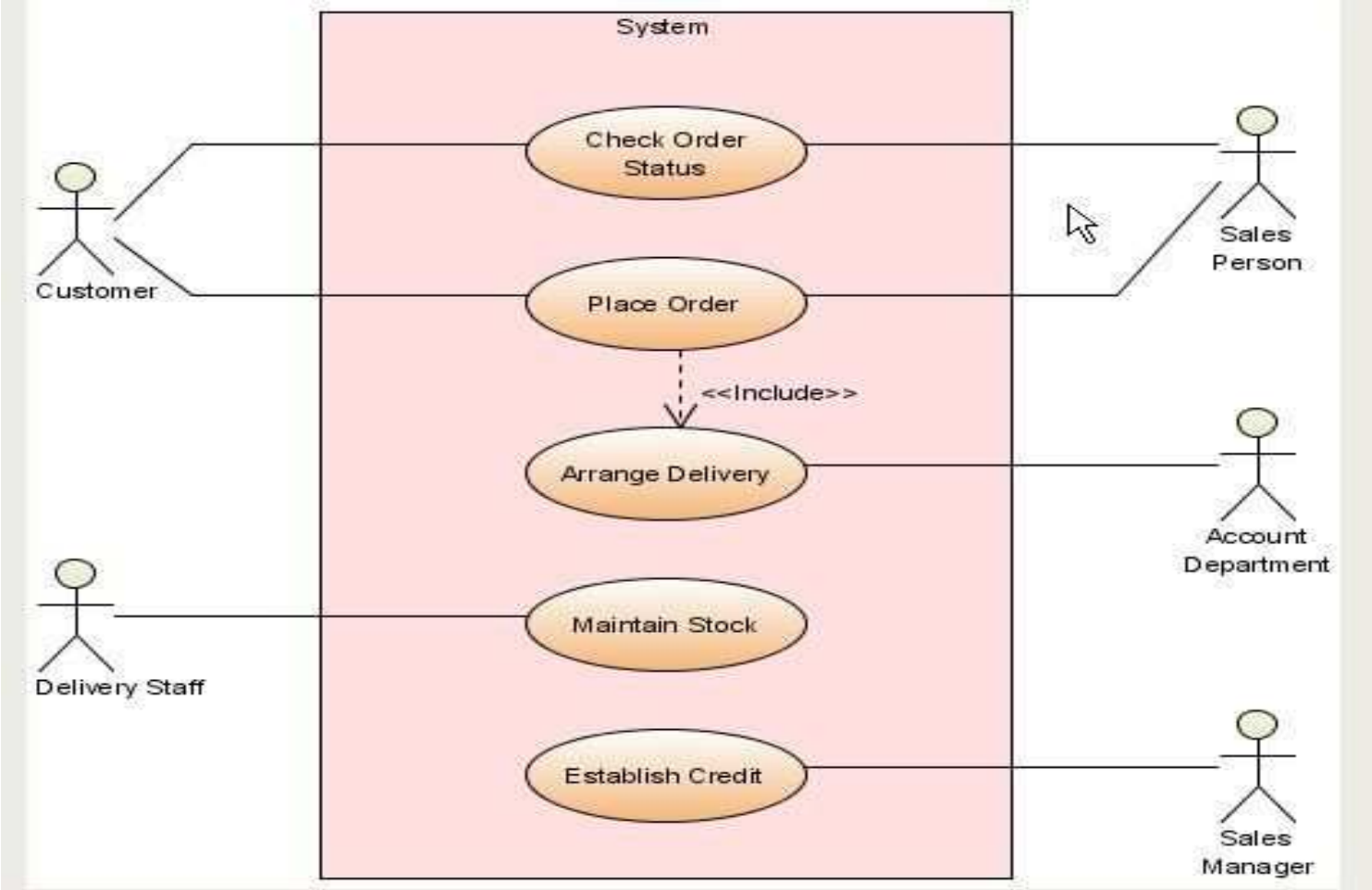


USE CASES: <<include>>



Semântica Operacional: Invocação de uma subrotina
<<include>> diz-se um estereótipo (uma marca especial)

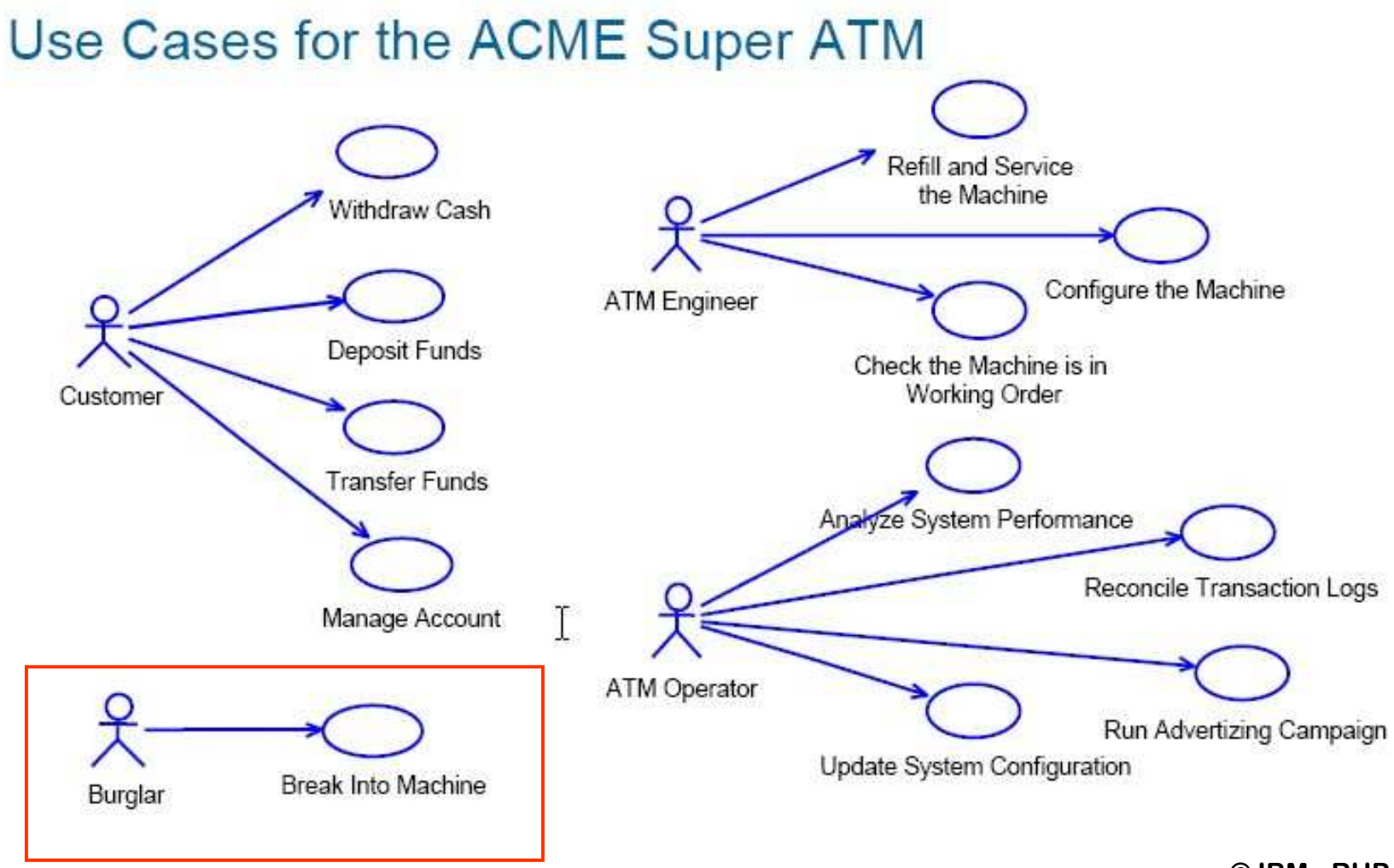
USE CASES: <<include>>



USE CASES: Importância

- São uma forma intuitiva e sistemática de capturar requisitos funcionais;
- São a base de todo o processo de desenvolvimento;
- Permitem identificar melhor as tarefas que são os objectivos dos utilizadores do sistema;
- Identificam o que o sistema deve fazer para cada tipo de utilizador;
- Especificam todas as possíveis utilizações do sistema;
- São instrumento de diálogo entre clientes e projectistas;
- Permitem desenvolver protótipos da Interface com o Utilizador.

USE CASES: Exemplo



© IBM - RUP

USE CASES: a continuar

Artigo para ler sobre Use Cases:

Use Cases: Yesterday, Today, and Tomorrow
Ivar Jacobson (o pai)

http://www.ibm.com/developerworks/rational/library/775.html