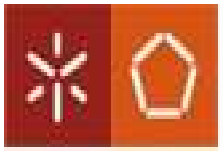


Universidade do Minho
Departamento de Informática

ARQUITECTURAS DE SOFTWARE

UC: ACS - MI/MEI

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;
- Estudo da linguagem OCL (“Object Constaint Language”);

FUNCIONAMENTO

- Teóricas e práticas laboratoriais por semana;
- 1 projecto obrigatório, em grupo, a entregar por fases;
- 3 relatórios teóricos, em grupo;
- Nota Final:

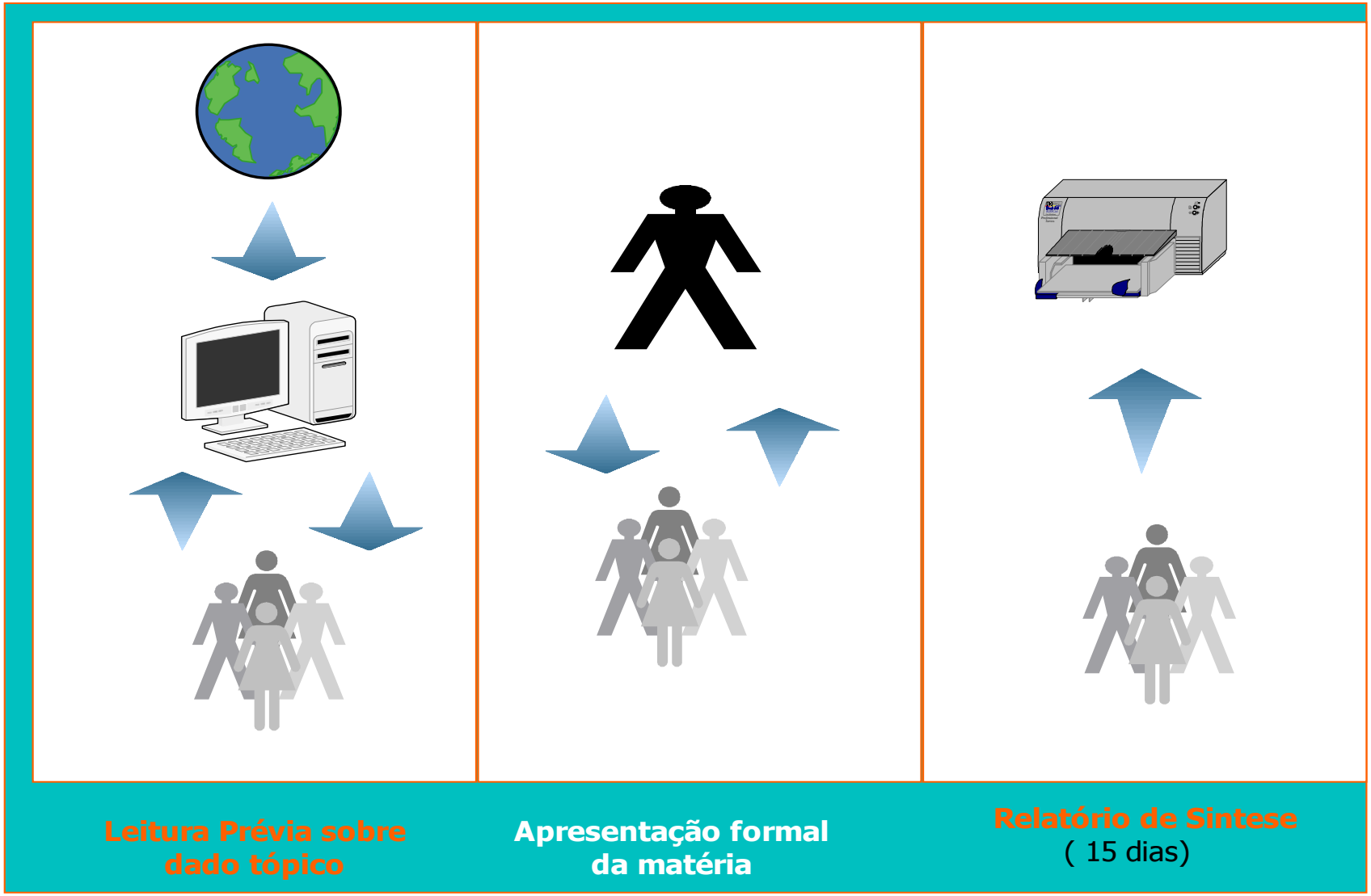
  Nota: 0.5 * Teórica + 0.5 * Trabalho

- Docentes:

  Prof. F. Mário Martins fmm@di.uminho.pt

Página da disciplina: sim.di.uminho.pt/disciplinas/as0809

PLANO - TEÓRICAS



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

MOTIVAÇÃO - IMPORTÂNCIA

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

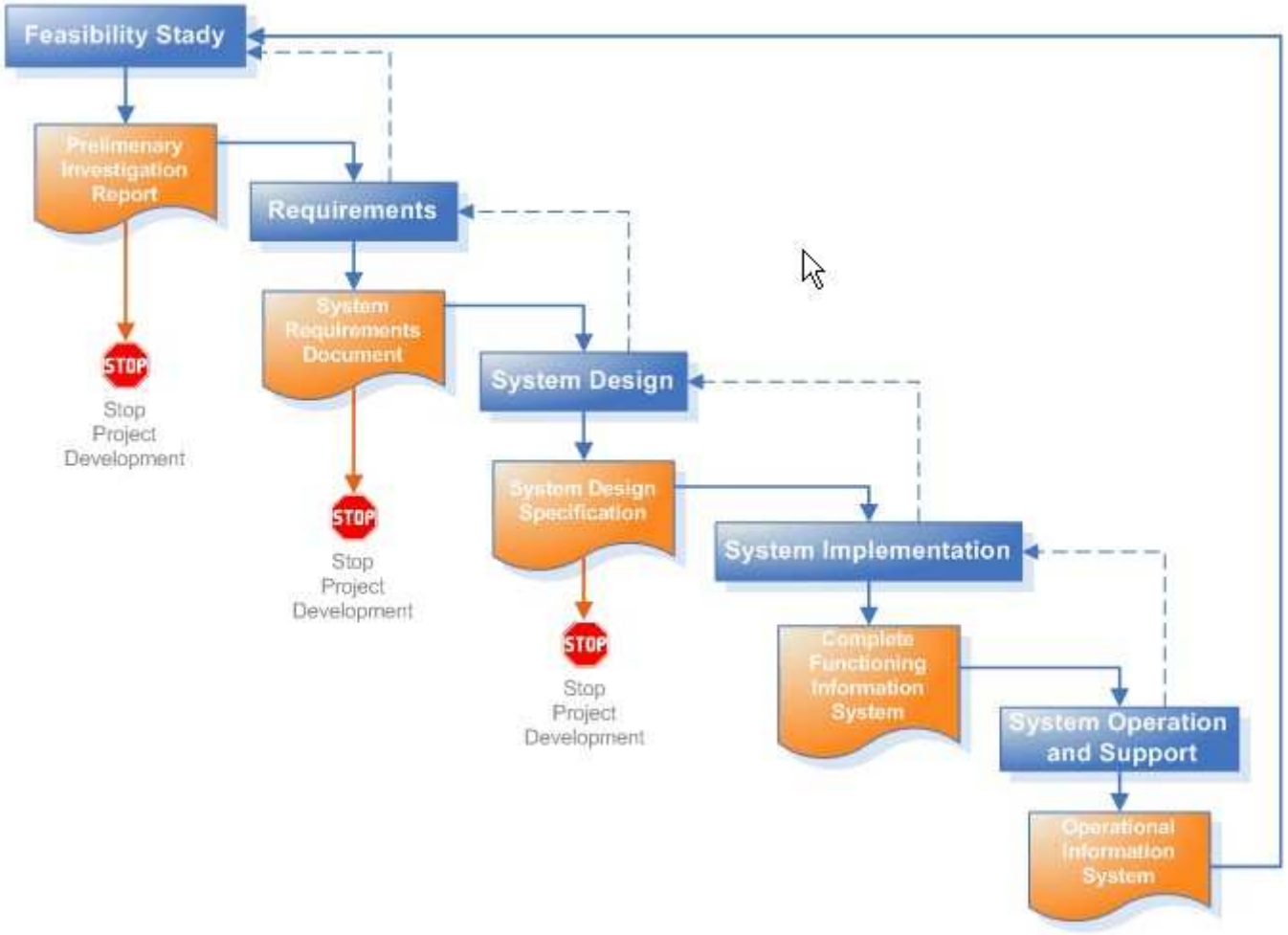
Slide 13

© 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.

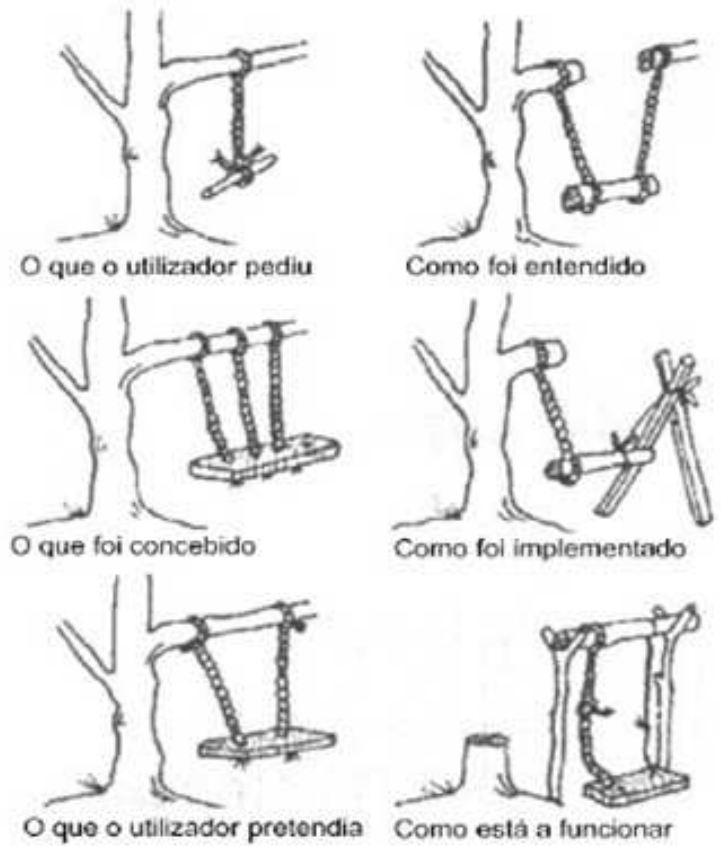
MOTIVAÇÃO - IMPORTÂNCIA



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

MOTIVAÇÃO - IMPORTÂNCIA

Desenvolver Sistemas Software não é trivial ...

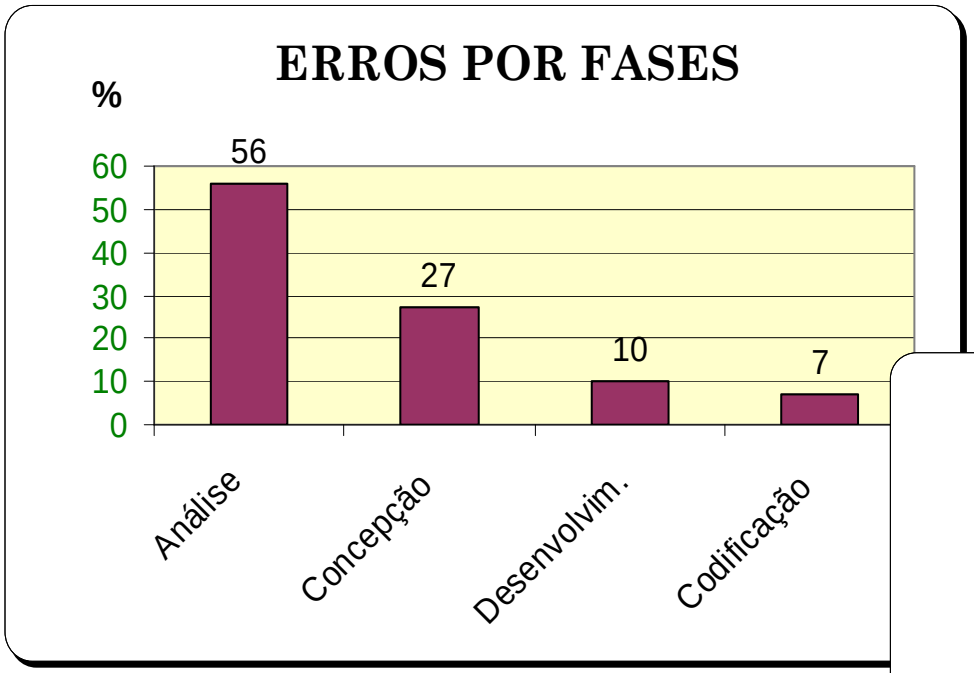


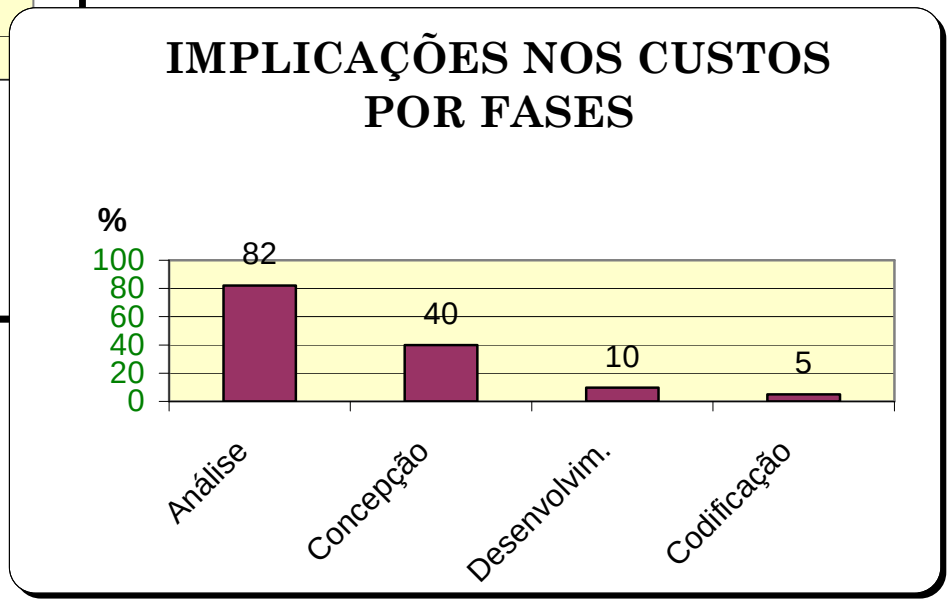
- Riscos associados aos requisitos.
- Riscos tecnológicos.
- Riscos de competência.
- Riscos políticos.

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

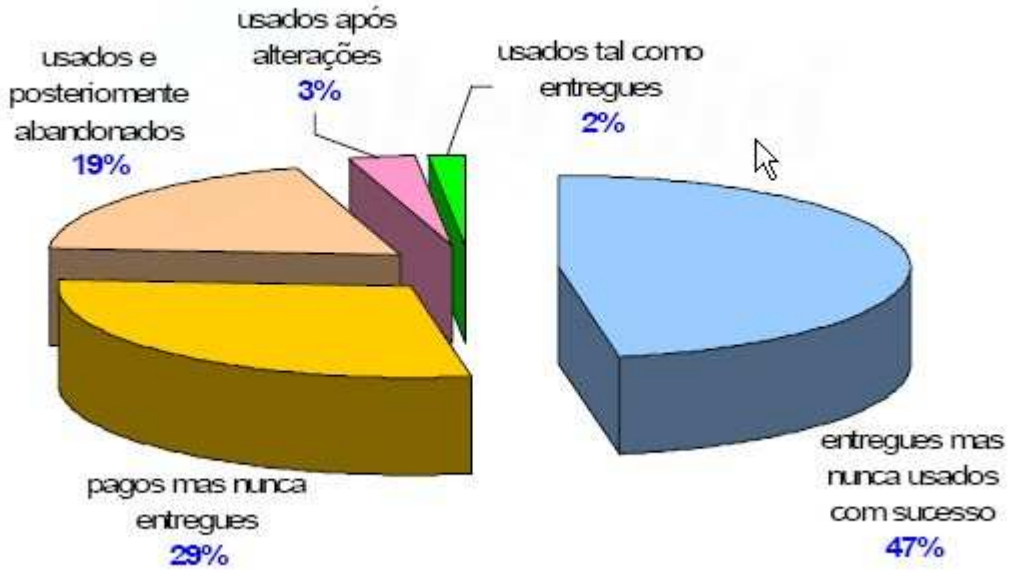
Sendo de salientar os erros de análise e custos implicados !!





MOTIVAÇÃO - IMPORTÂNCIA

E a história tem sido muito pouco satisfatória …

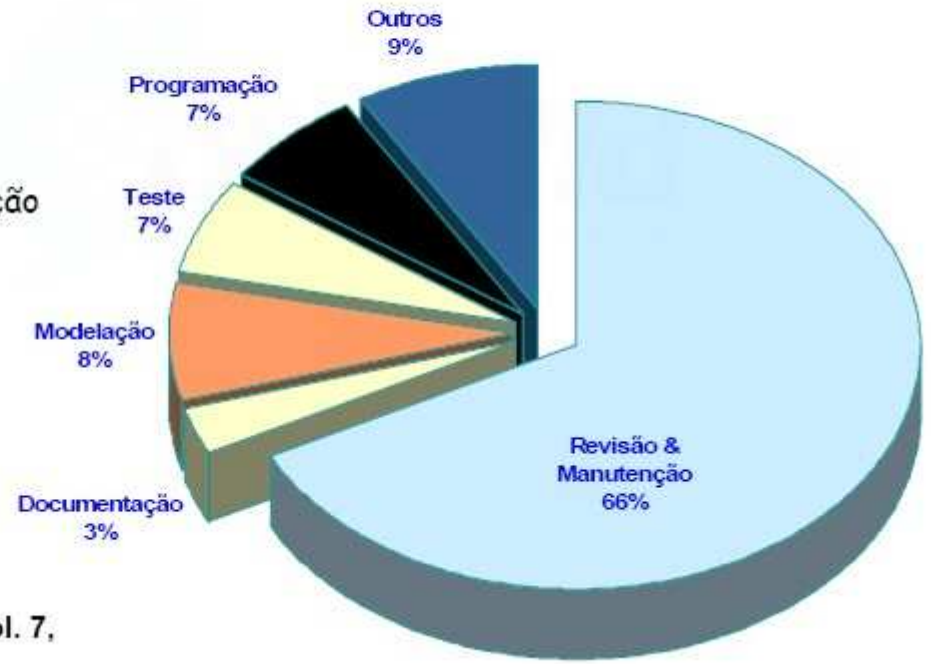


MOTIVAÇÃO - IMPORTÂNCIA

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

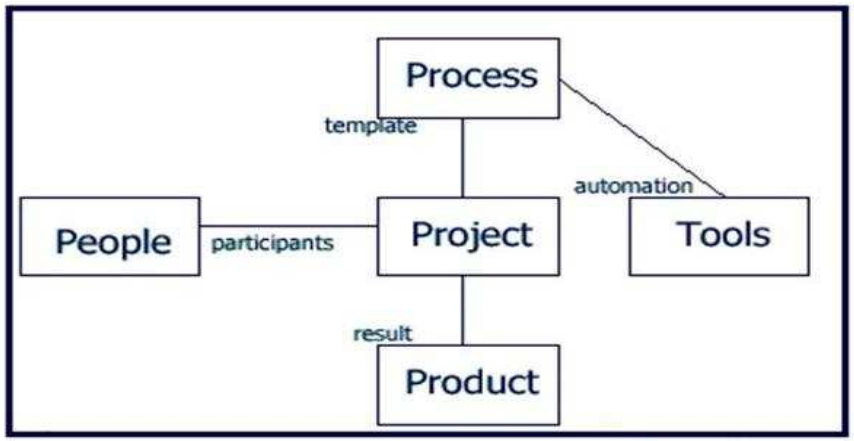
O desenvolvimento de software ainda tem muito de arte e muito pouco de verdadeira Engenharia.

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 efectivamente 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 projectado 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.

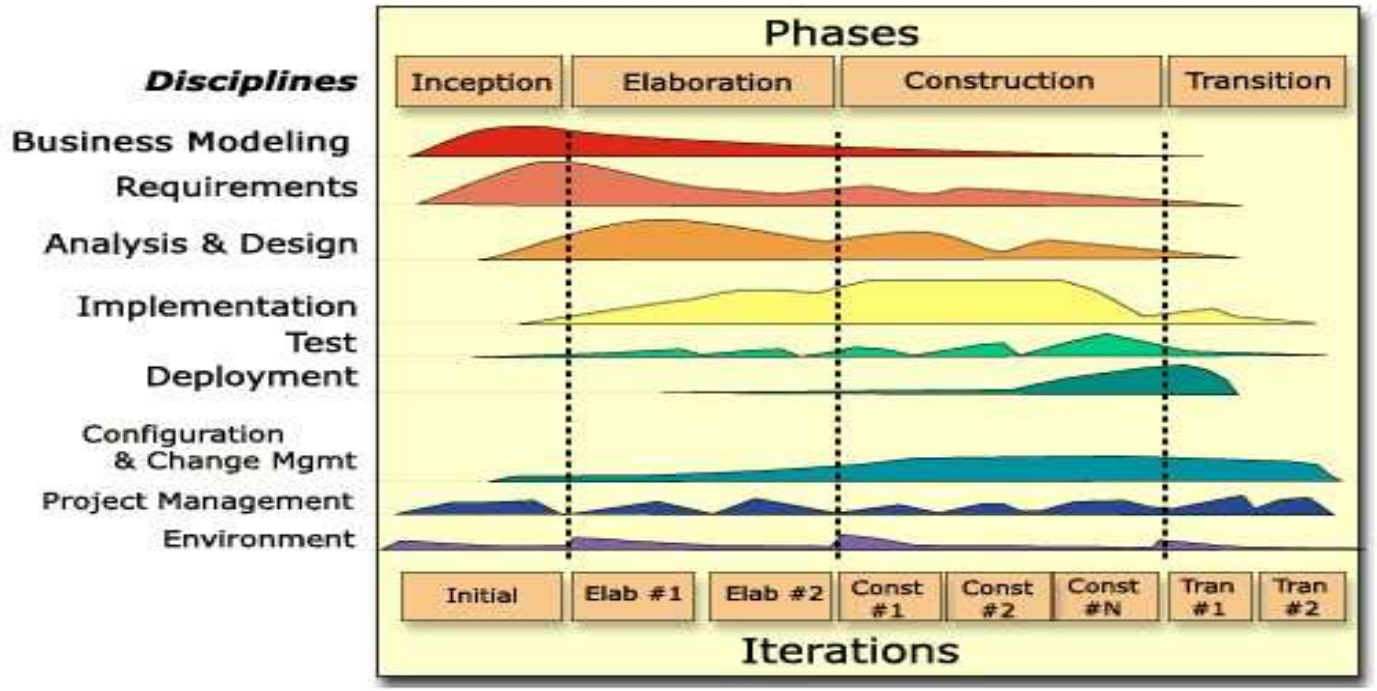
MOTIVAÇÃO - IMPORTÂNCIA

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)

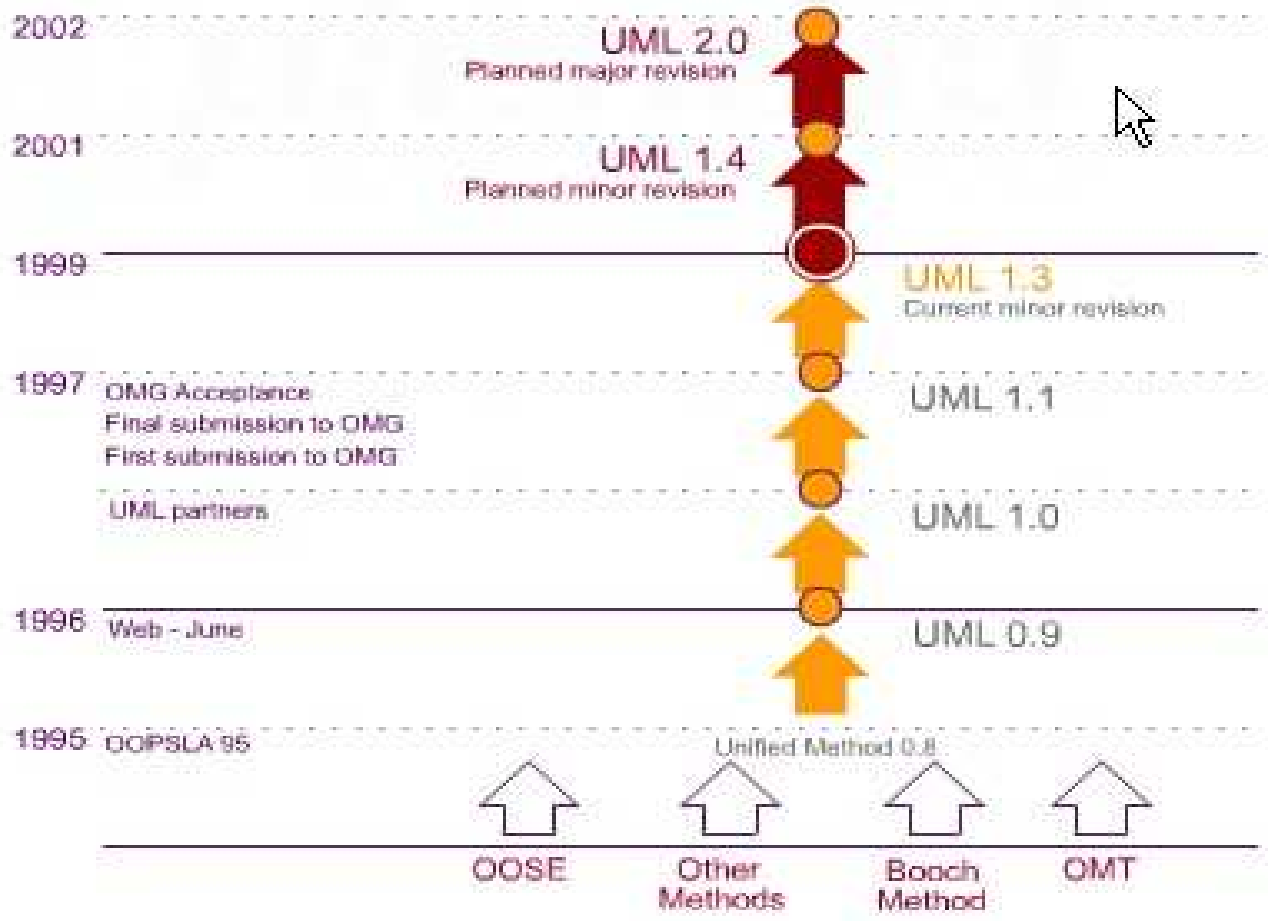
É um standard de facto

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

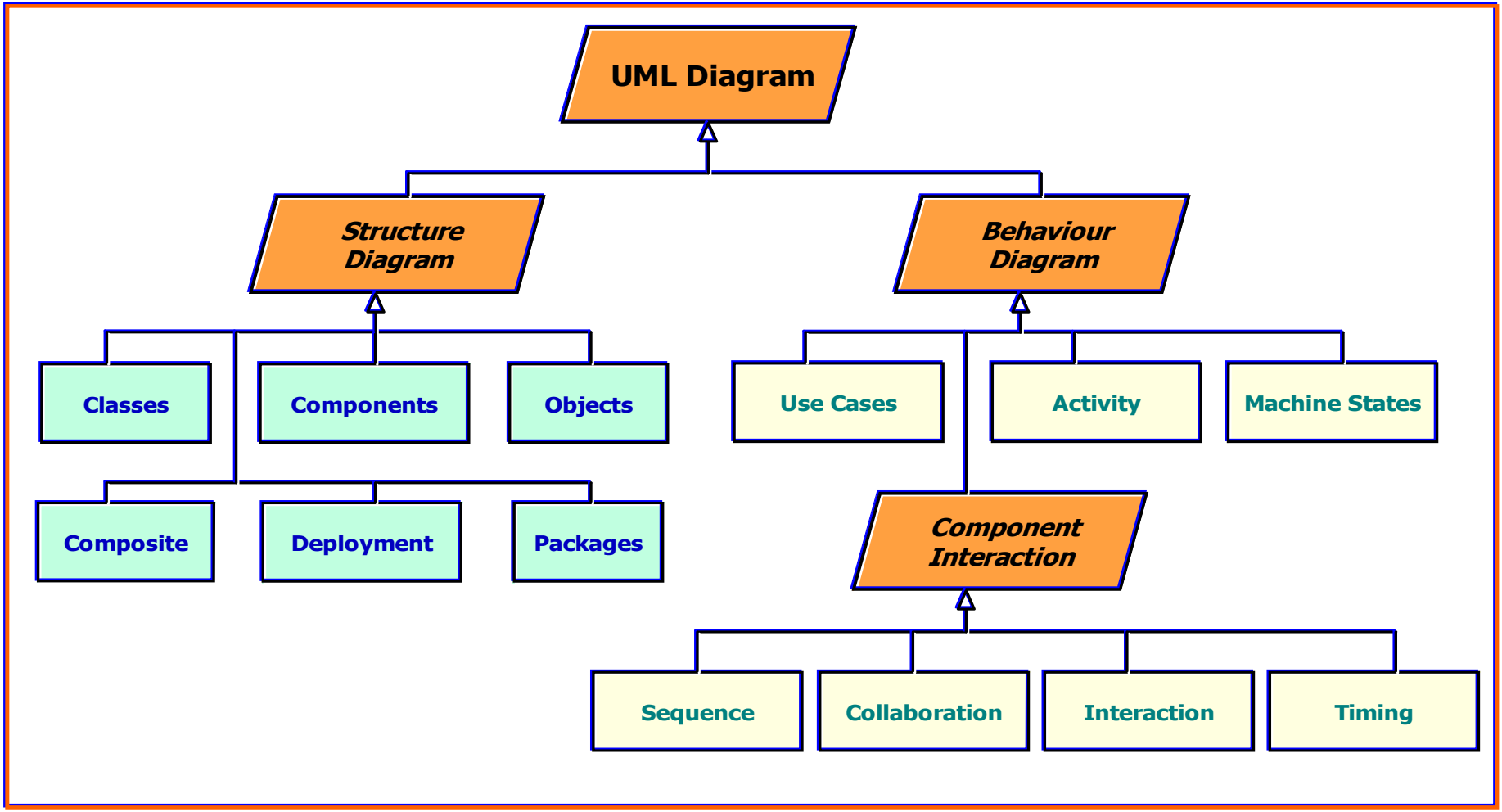
Diagramas de Comportamento

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

Diagramas de Interacção

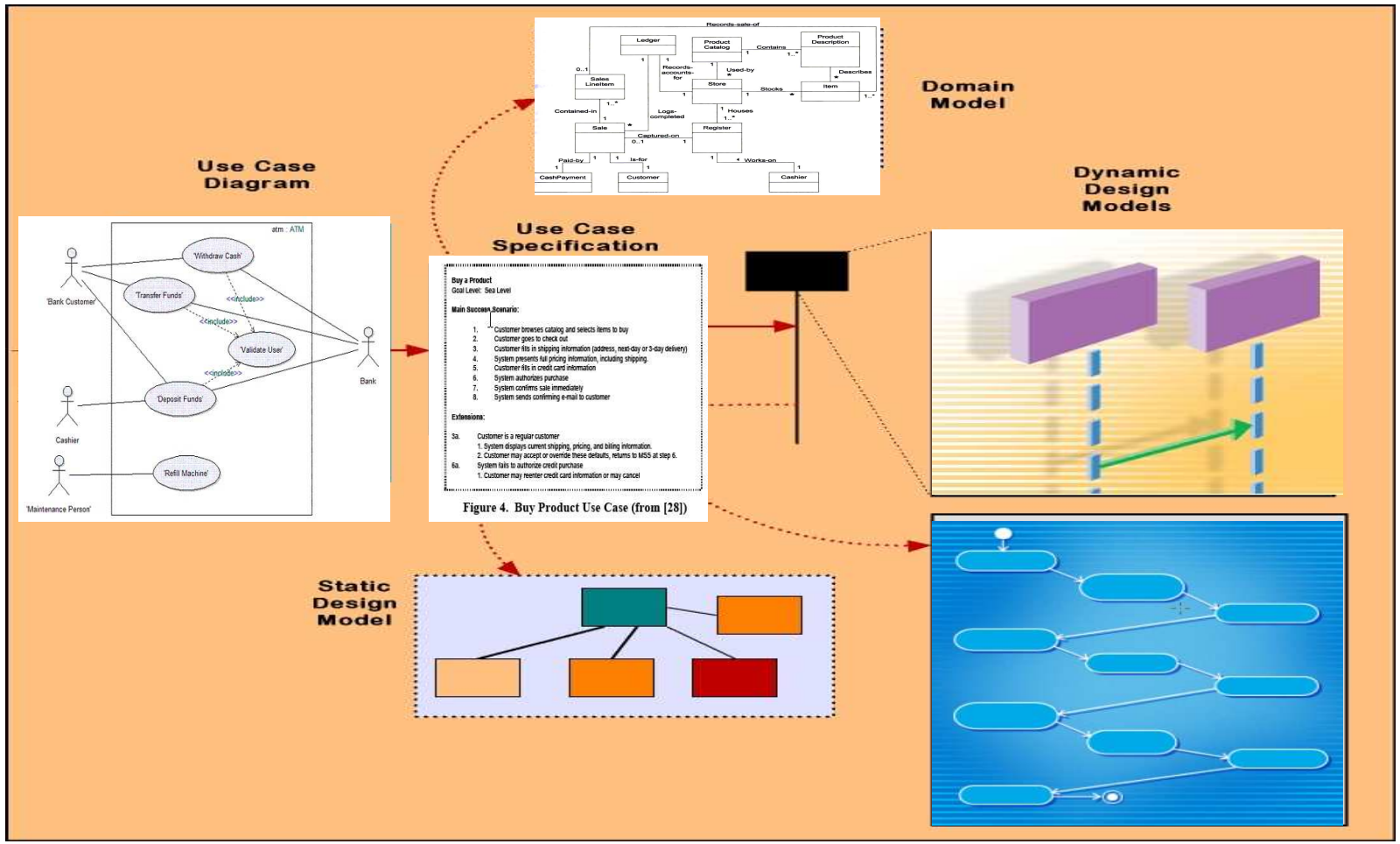
- Diagrama de Sequência
- Diagrama de Interacção
- Diagrama de Colaboração
- Diagrama de Temporização

UML: DIAGRAMAS e MODELOS



VISÕES FUNDAMENTAIS: ESTRUTURAL e COMPORTAMENTAL

E DEPOIS METODOLOGIA …



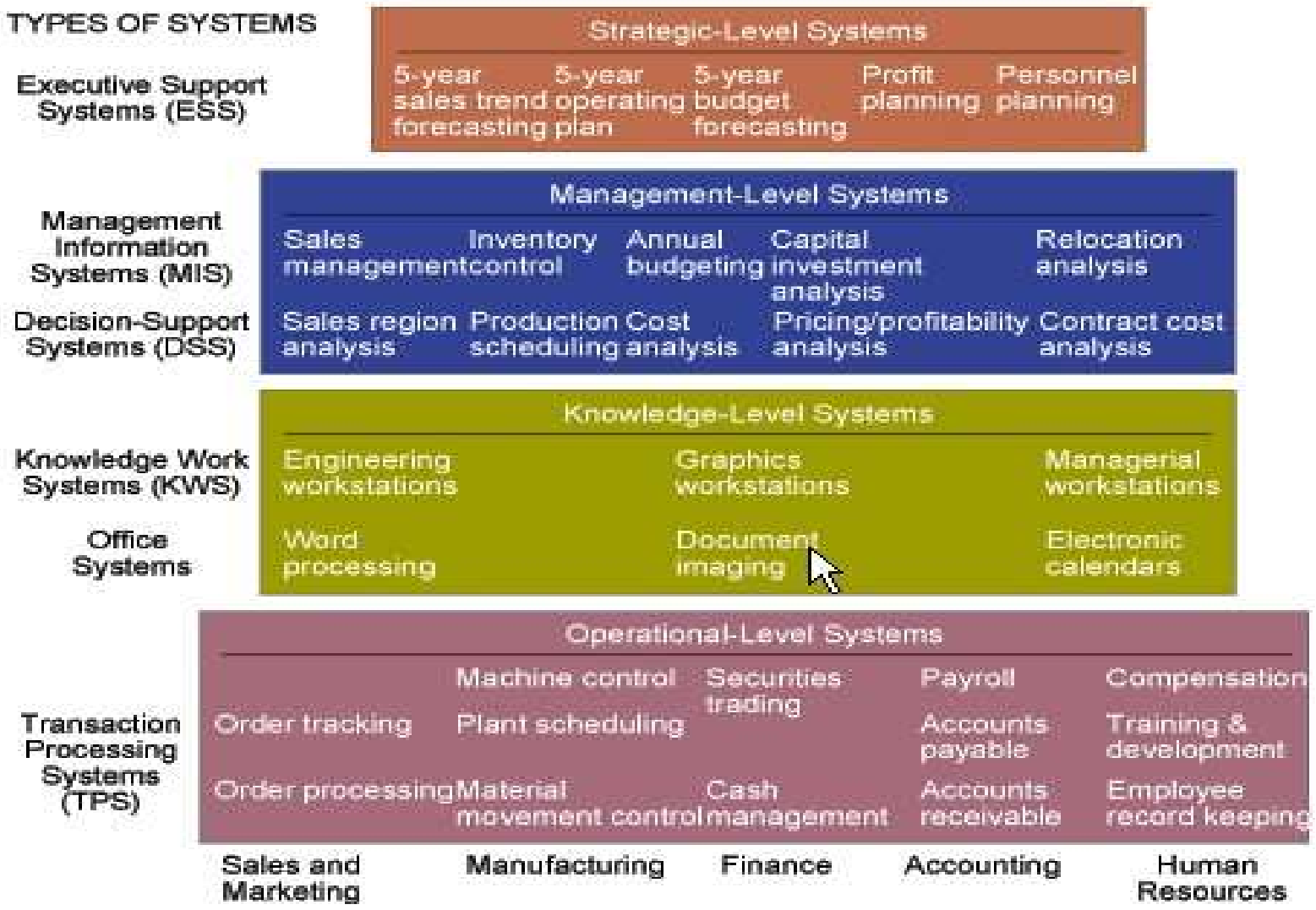
Domain Model + Use Case Model são depois refinados sistematicamente nos outros modelos, estruturais e comportamentais, idealmente diferenciando objectos que são de camadas distintas (dados, computacional, negócio e IU).

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 ?

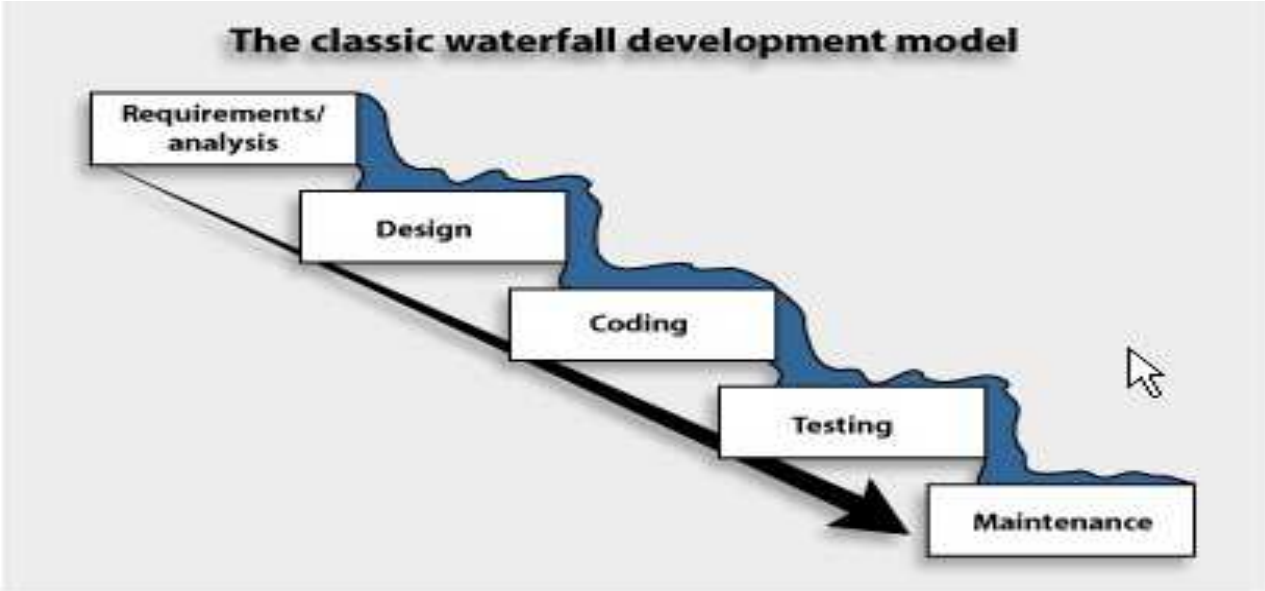
1.- Adoptar um Processo de desenvolvimento de Sistemas Software que nos garanta tais metas, em especial depois de conhecermos a história dos processos desenvolvidos; O processo adoptado deverá dar, justificadamente, muitas mais garantias de sucesso no desenvolvimento dos actuais e futuros SI (ou Sistemas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

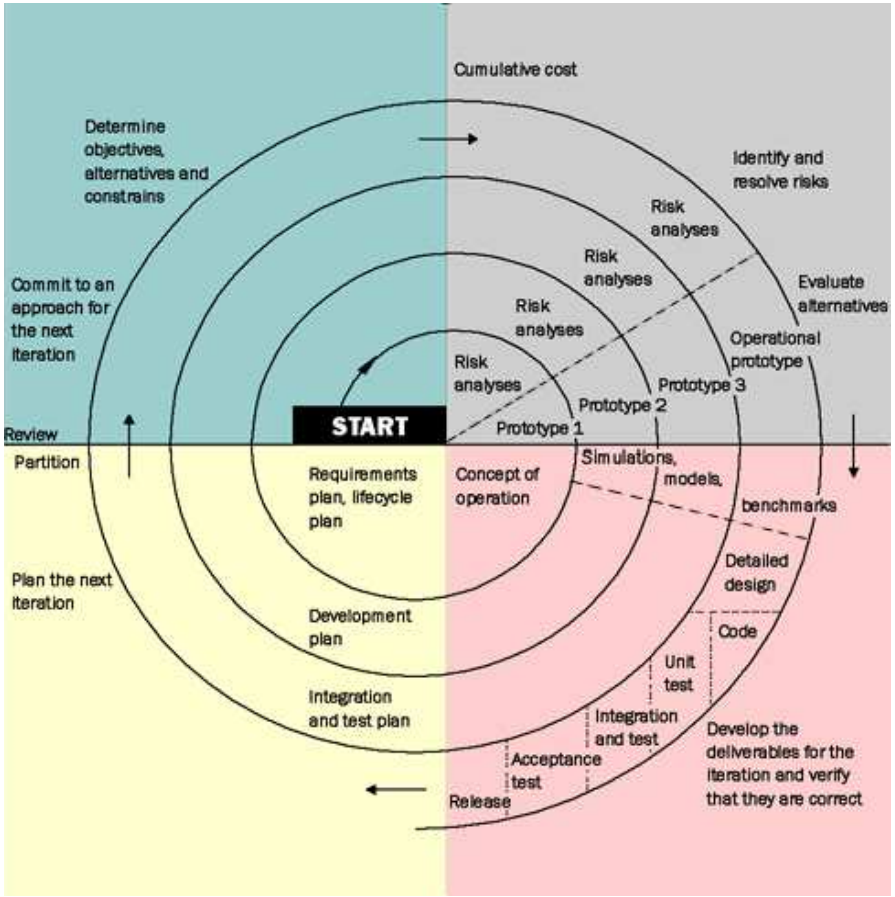
3.- Adoptar Ferramentas que representam o suporte automático ou semi-automático aos processos e aos métodos.

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 !!

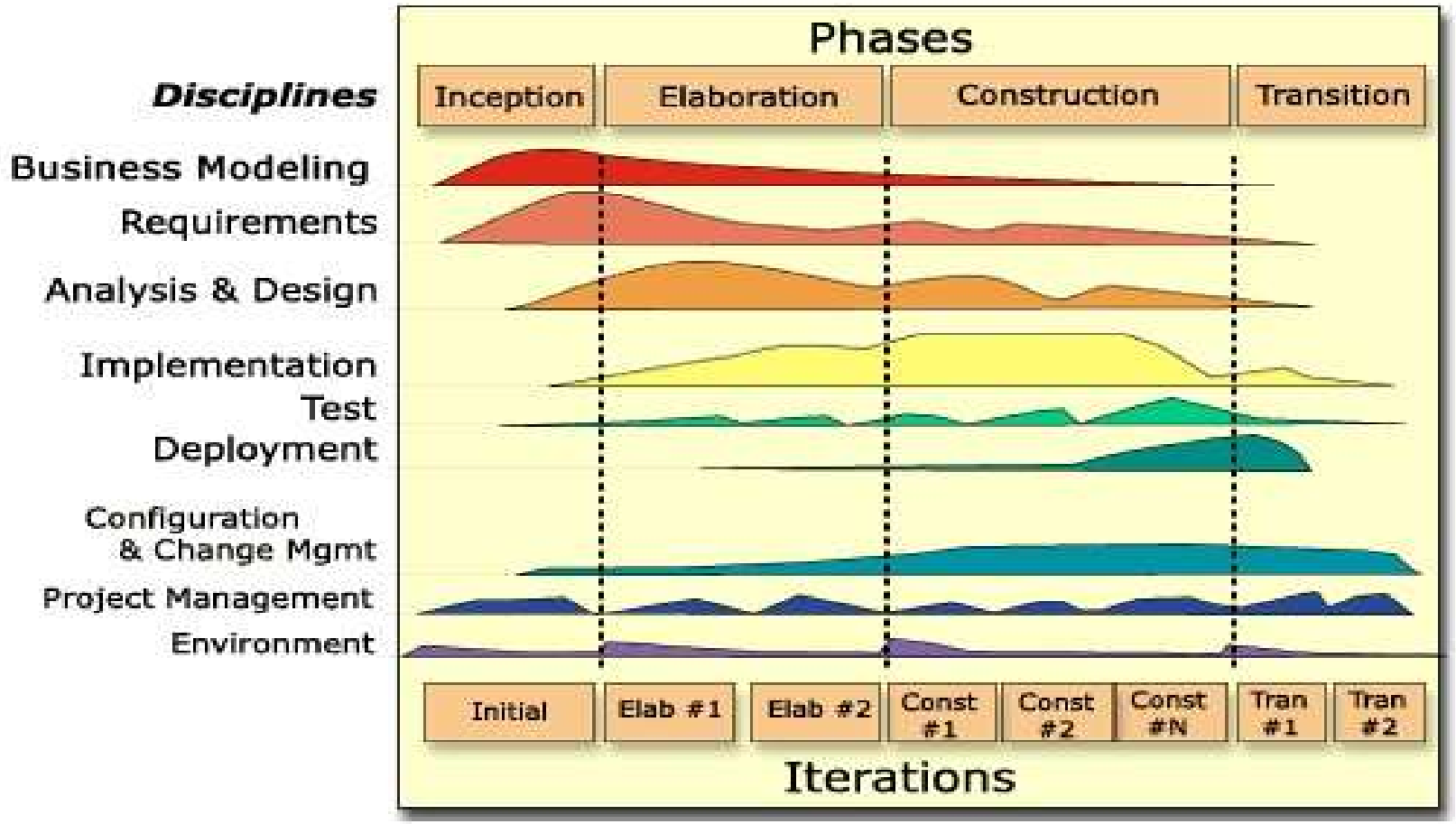


Modelo em Espiral

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:

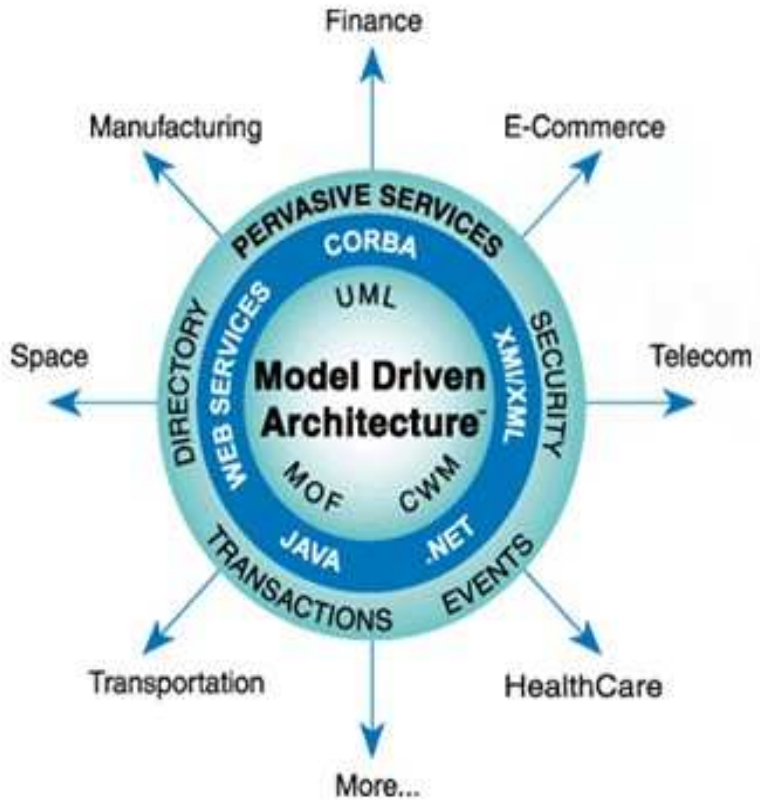
CONCEPÇÃO/DESIGN/ESPECIFICAÇÃO: Fase na qual, após todas as análises realizadas na fase anterior, se determina, "escreve", de forma mais ou menos rigorosa, o que o Sistema Software "deve ser capaz de fazer", informação que é fundamental para os programadores, assim sejam eles capazes de compreender as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

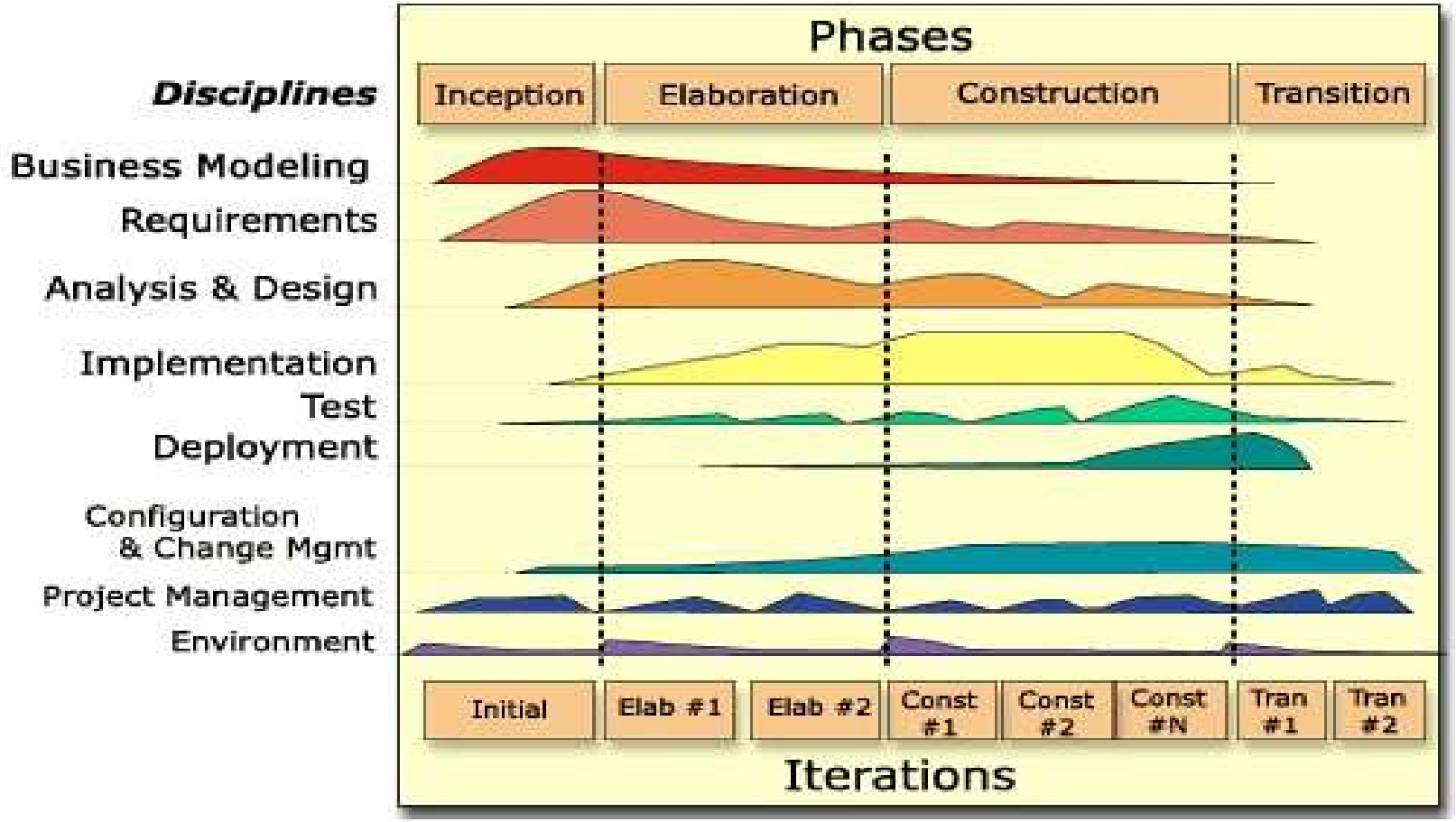
OMG Model Driven Architecture



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.

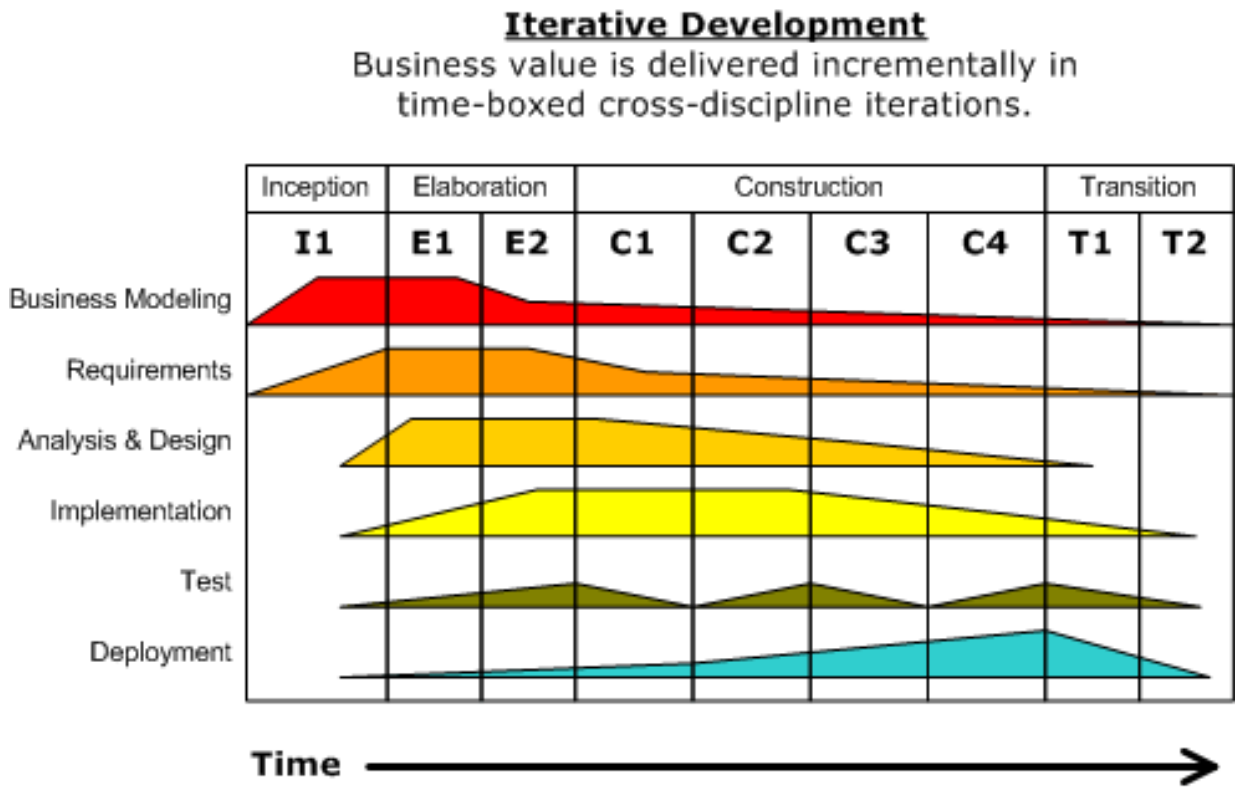
O QUE DIZ O RUP …



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

O QUE DIZ O RUP ...

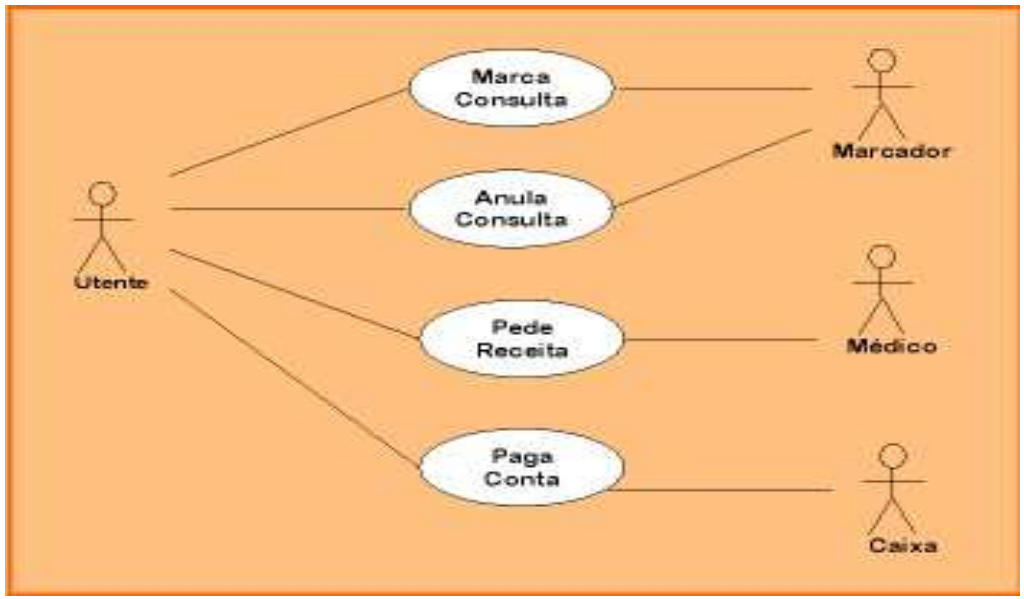
- **Processo de Desenvolvimento de Software que é iterativo e incremental**
  - As várias fases são divididas em séries de mini-fases que correspondem a sucessivas versões mais completas dos sistemas.



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

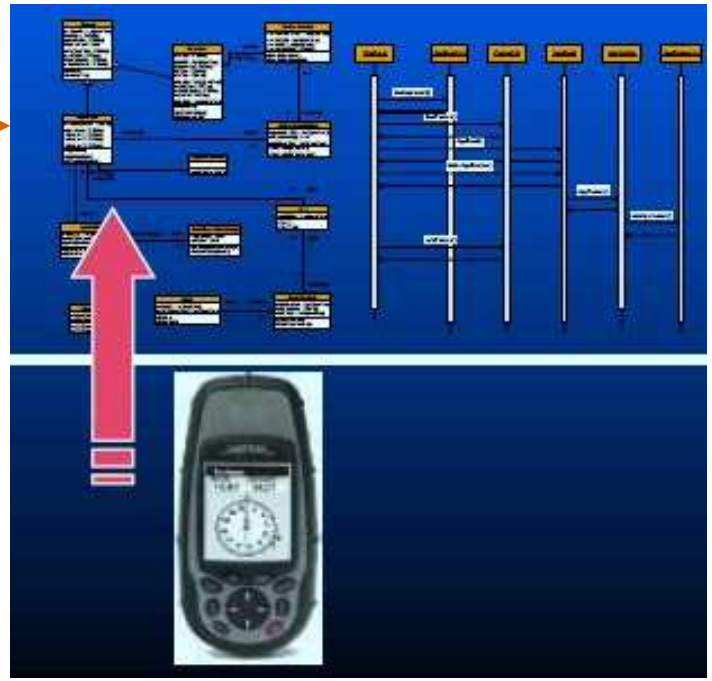
O QUE DIZ O RUP …

- **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



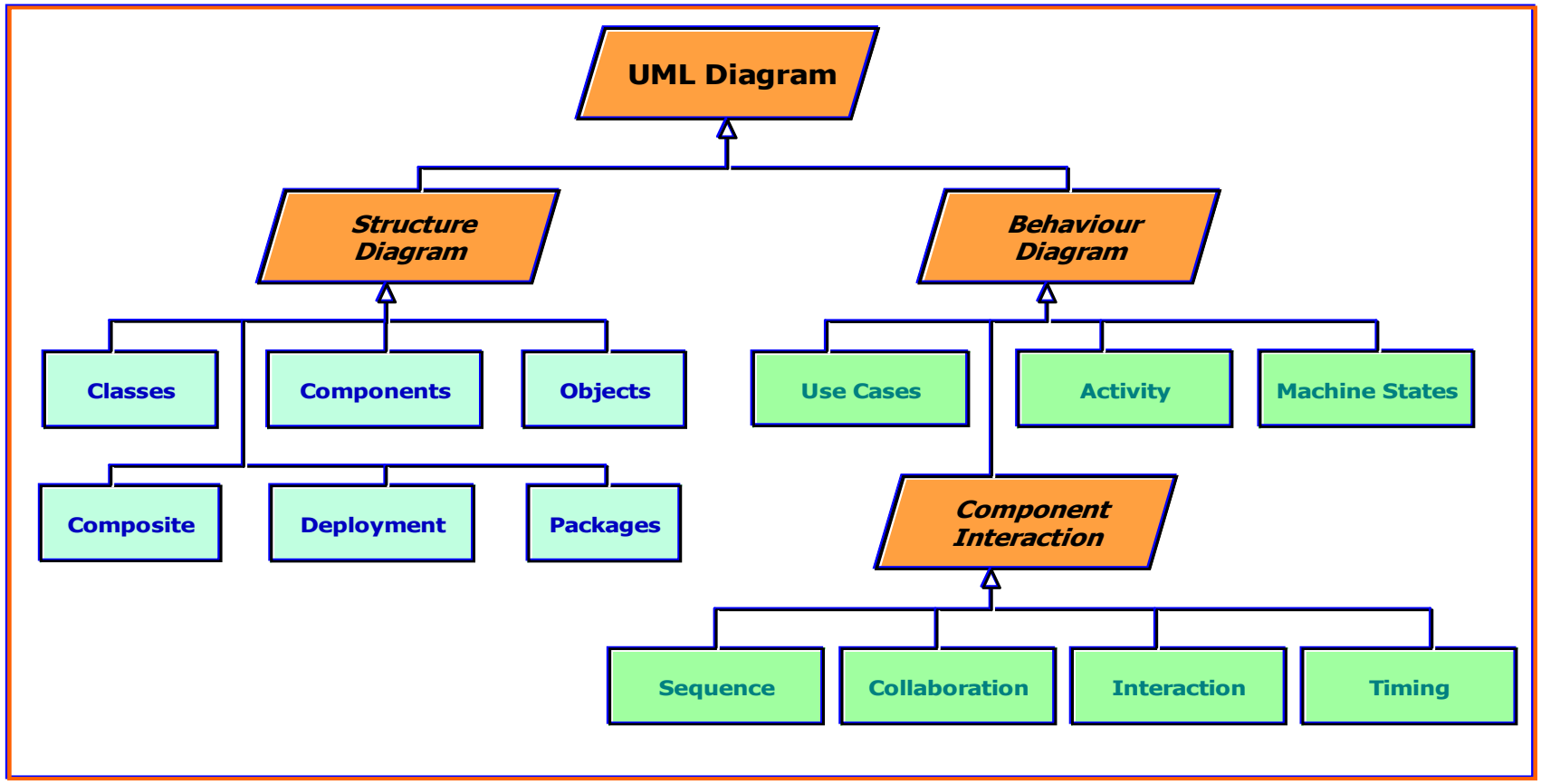
Nota: 1 aspecto => 1..* modelos

Mm: Espaço de modelação

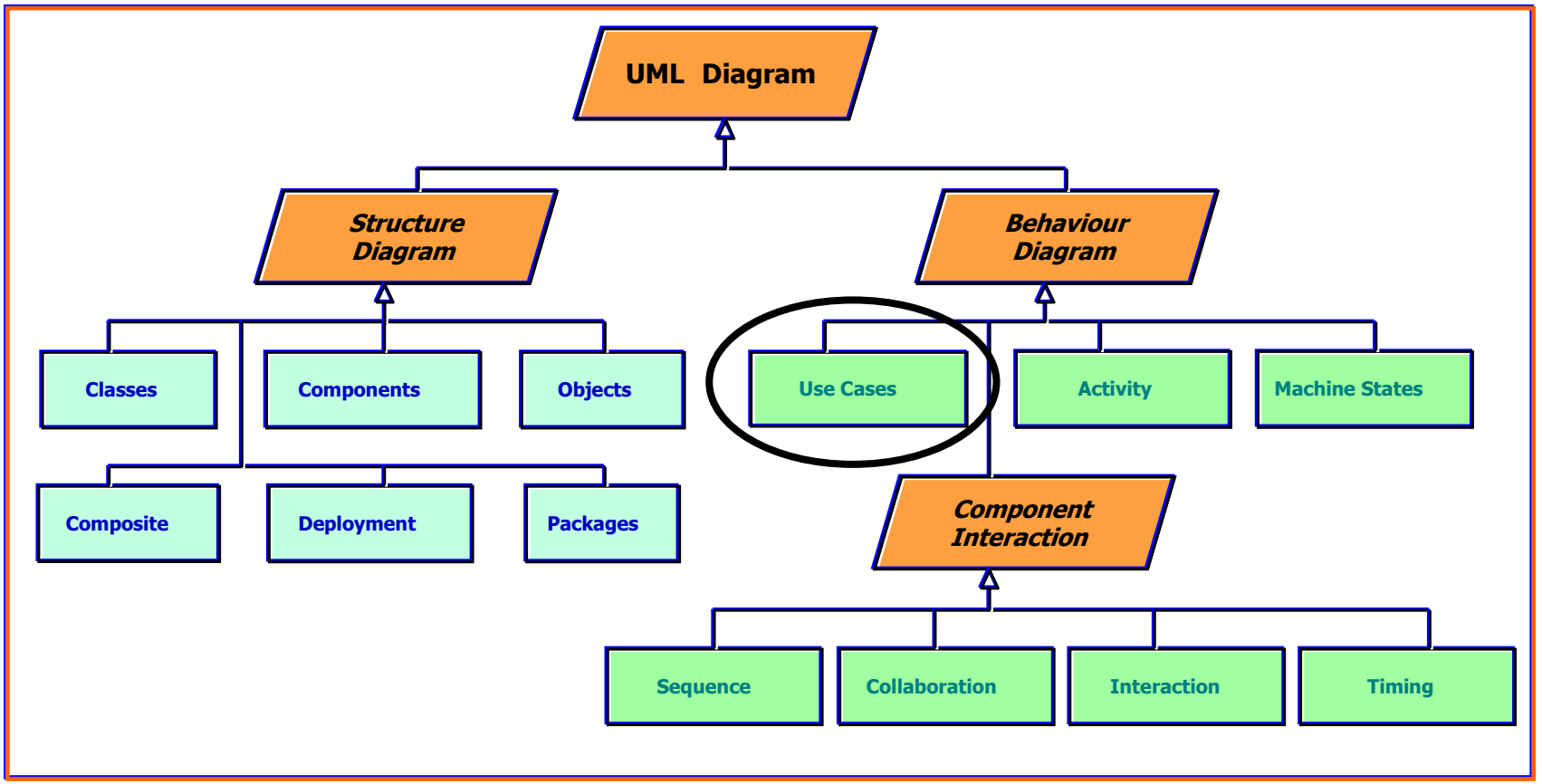
Mr: Mundo real

Modelos são conjuntos de diagramas + texto;

Diagramas são vistas de um modelo;

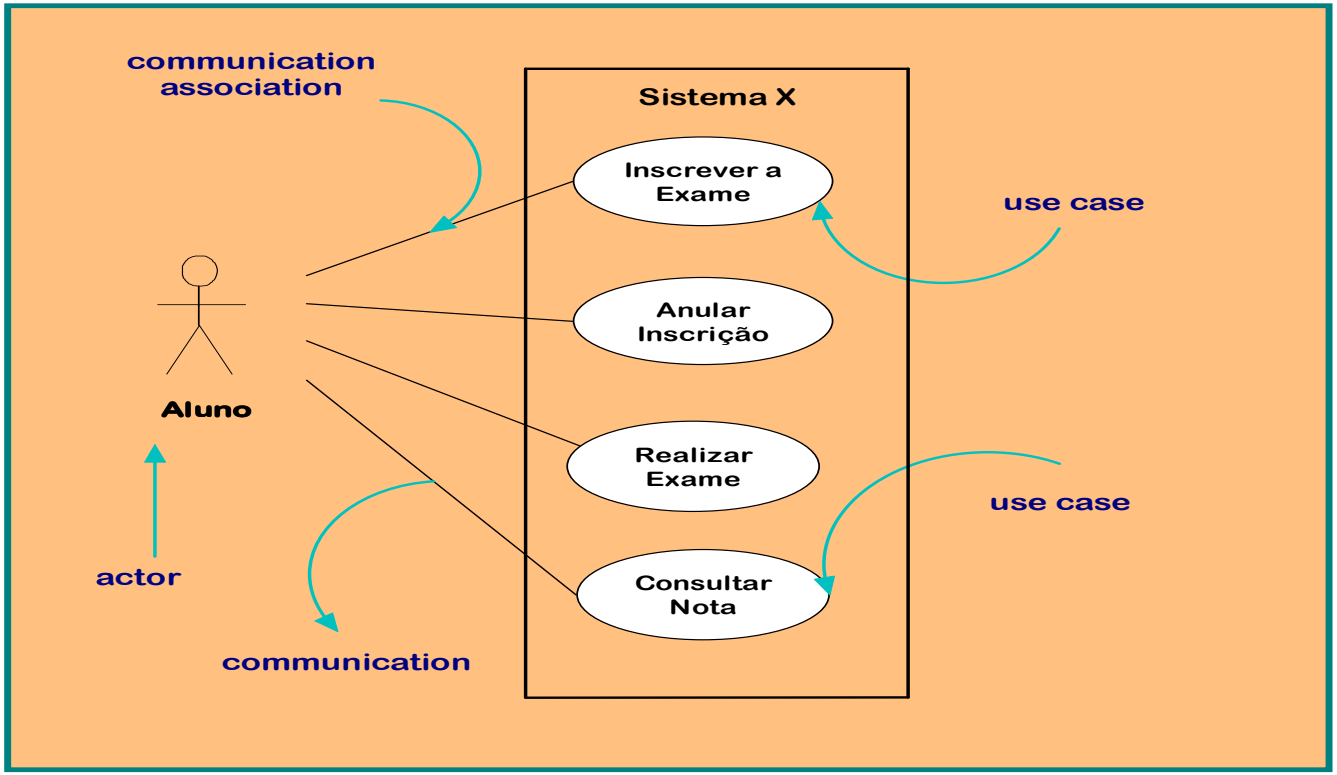


USE CASES

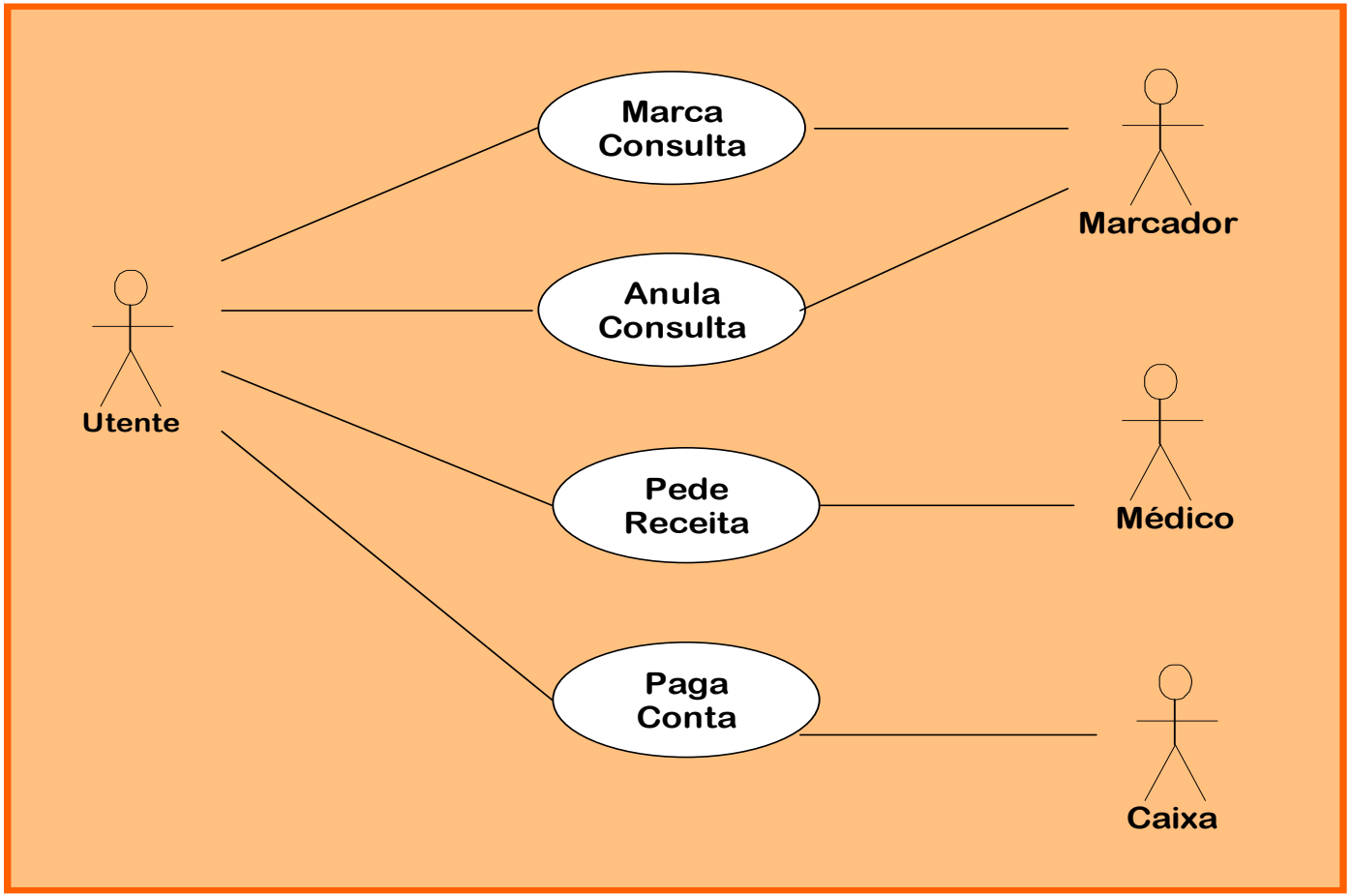

UML Diagram
Structure Diagram
Behaviour Diagram
Classes
Components
Objects
Use Cases
Activity
Machine States
Composite
Deployment
Packages
Component Interaction
Sequence
Collaboration
Interaction
Timing

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).

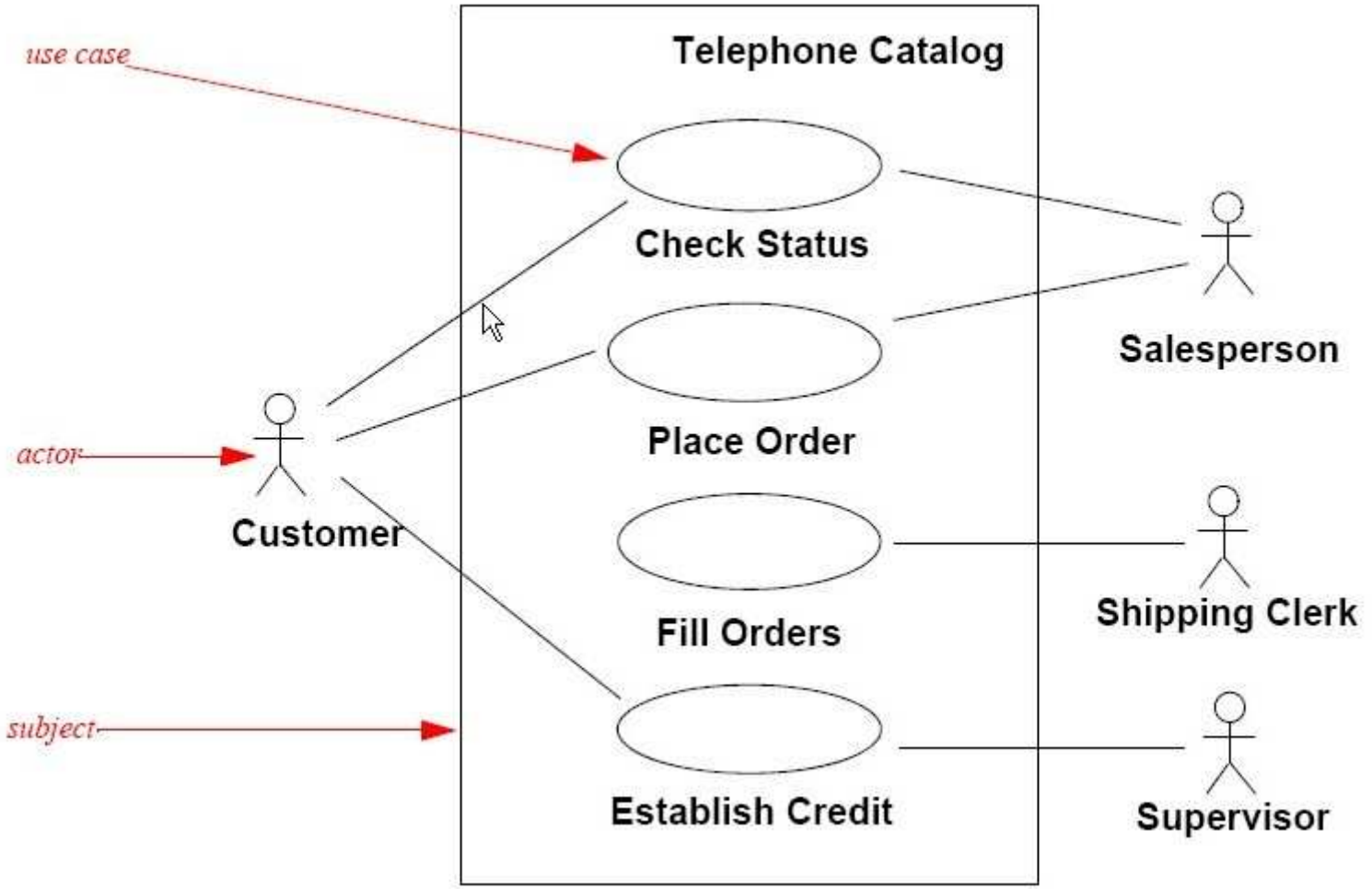


USE CASES

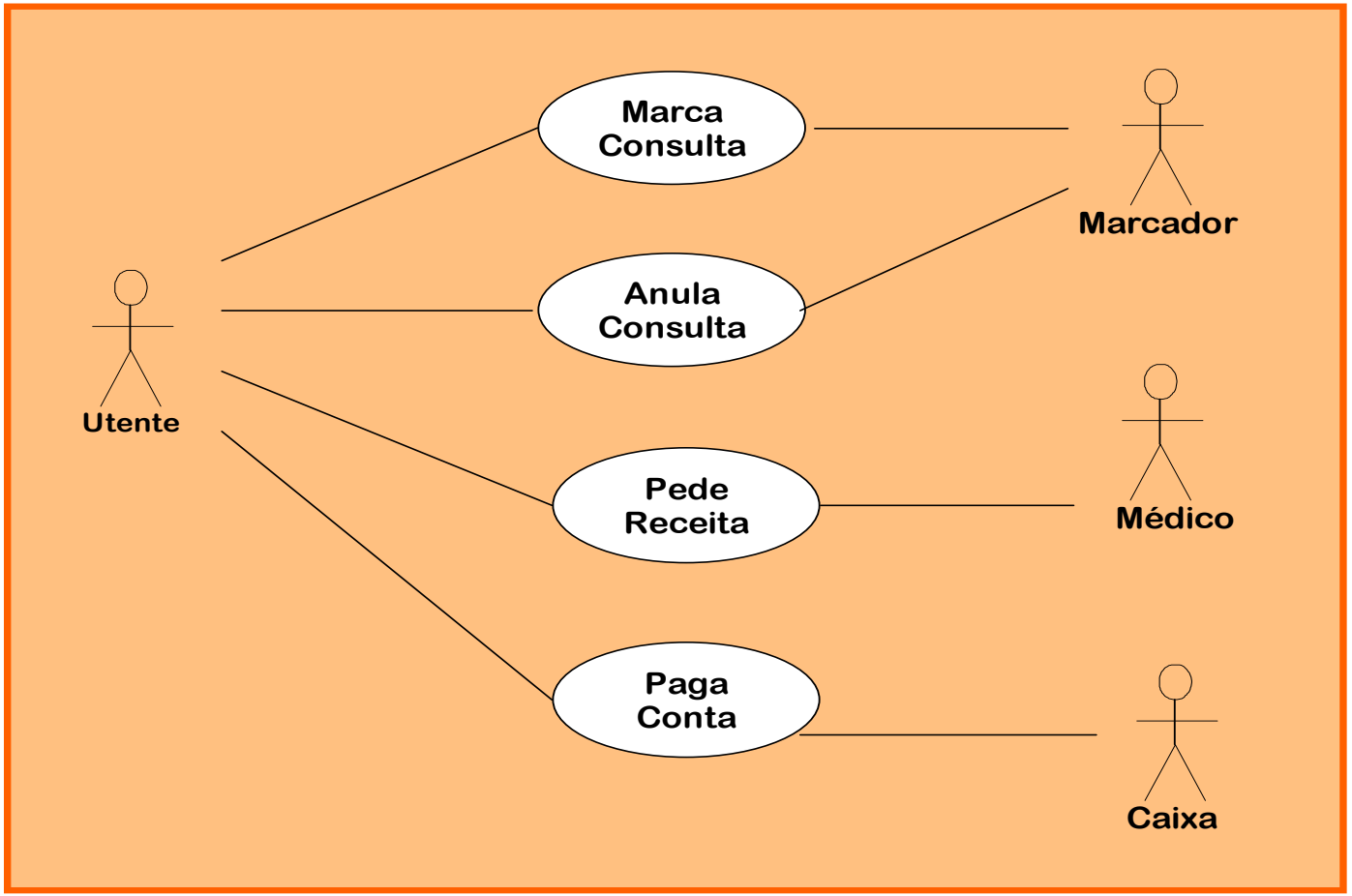


Use Case = **Uma funcionalidade do sistema (software ou não).**

USE CASES

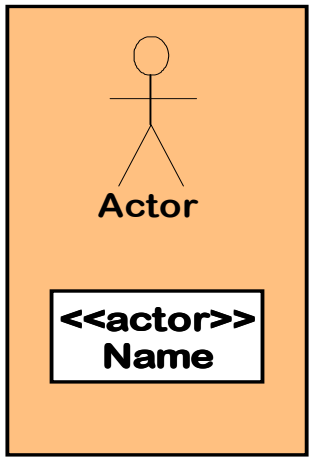

use case
Telephone Catalog
Check Status
Salesperson
Place Order
actor
Customer
Fill Orders
Shipping Clerk
subject
Establish Credit
Supervisor

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.

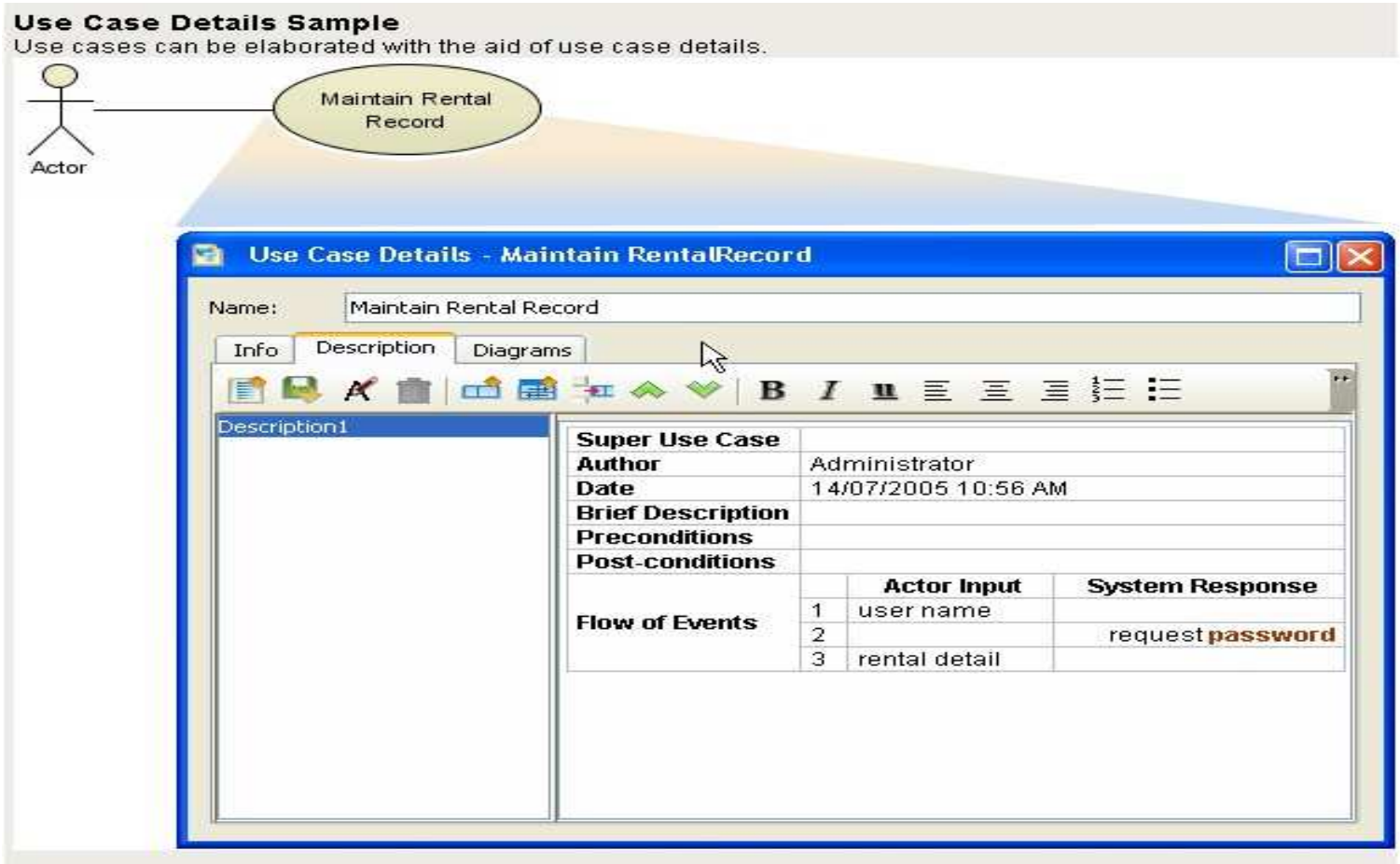
▣ Um use case é um resumo de todos os possíveis cenários para a realização de uma dada tarefa ou obtenção de um dado objectivo (goal). É comum um use case ter vários actores envolvidos.

▣ O que falta? Modularidade, estruturação e reutilização !

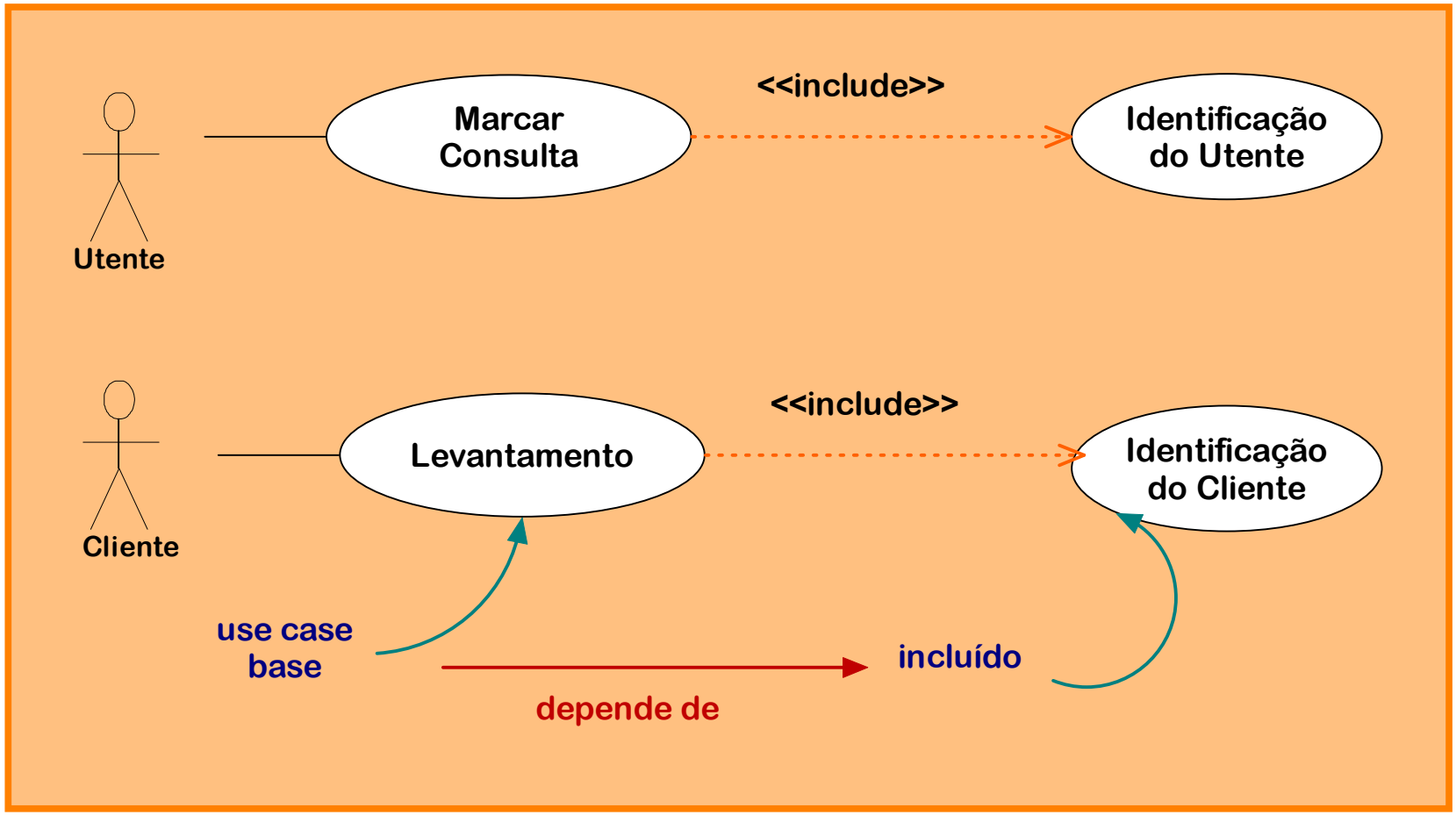
▣ Que relações são típicas da abordagem OO ?

USE CASES

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

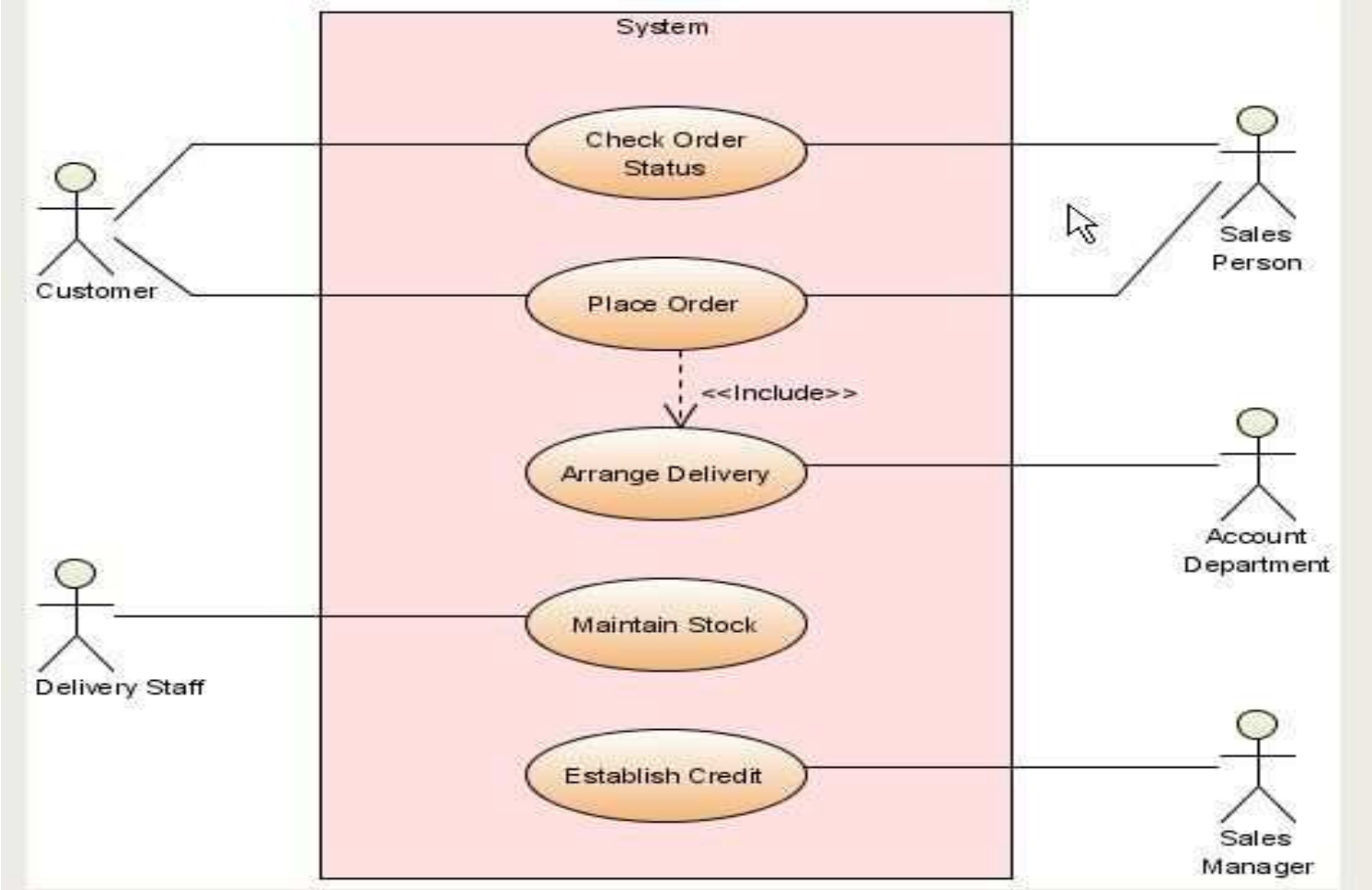


USE CASES: <<include>>



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

USE CASES: <<include>>

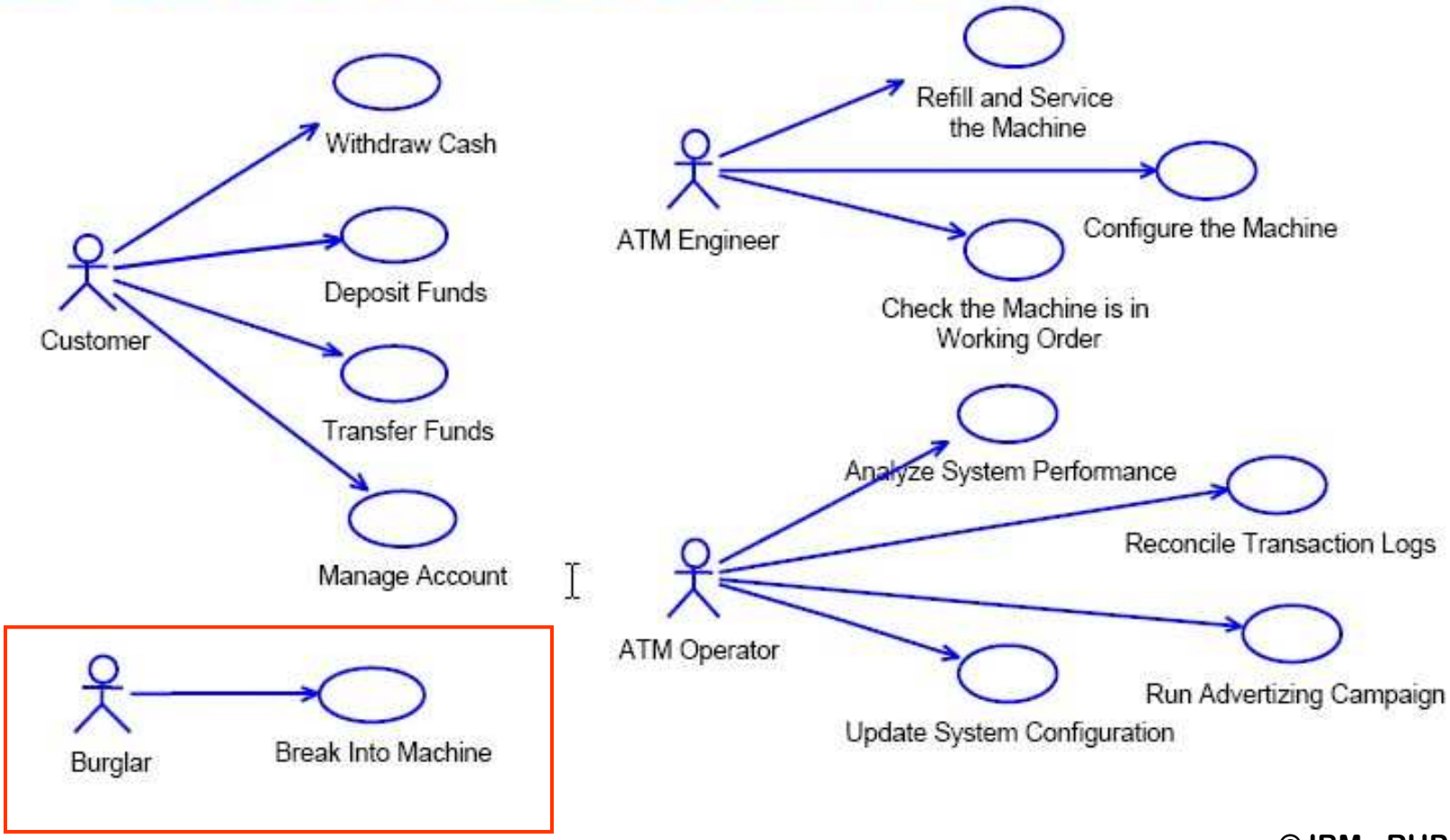


USE CASES: Importância

- 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

Use Cases for the ACME Super ATM



© IBM - RUP

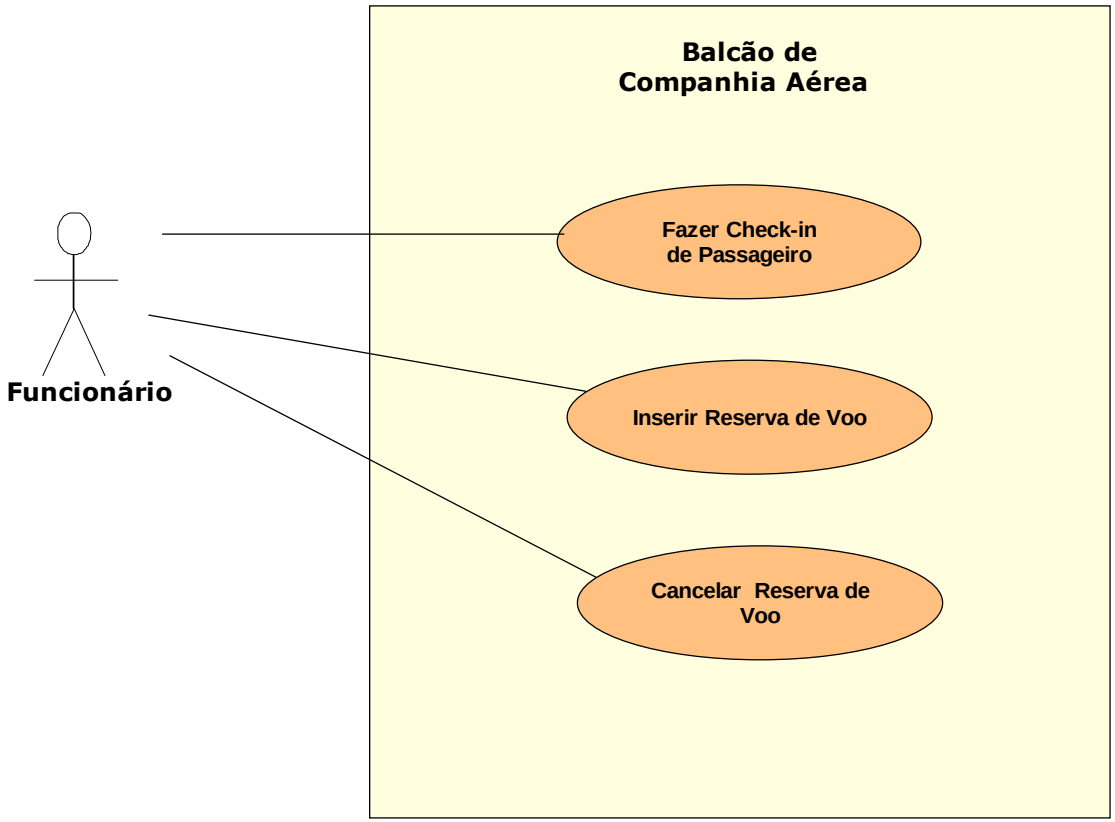
USE CASES: a continuar

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

USE CASES: continuação

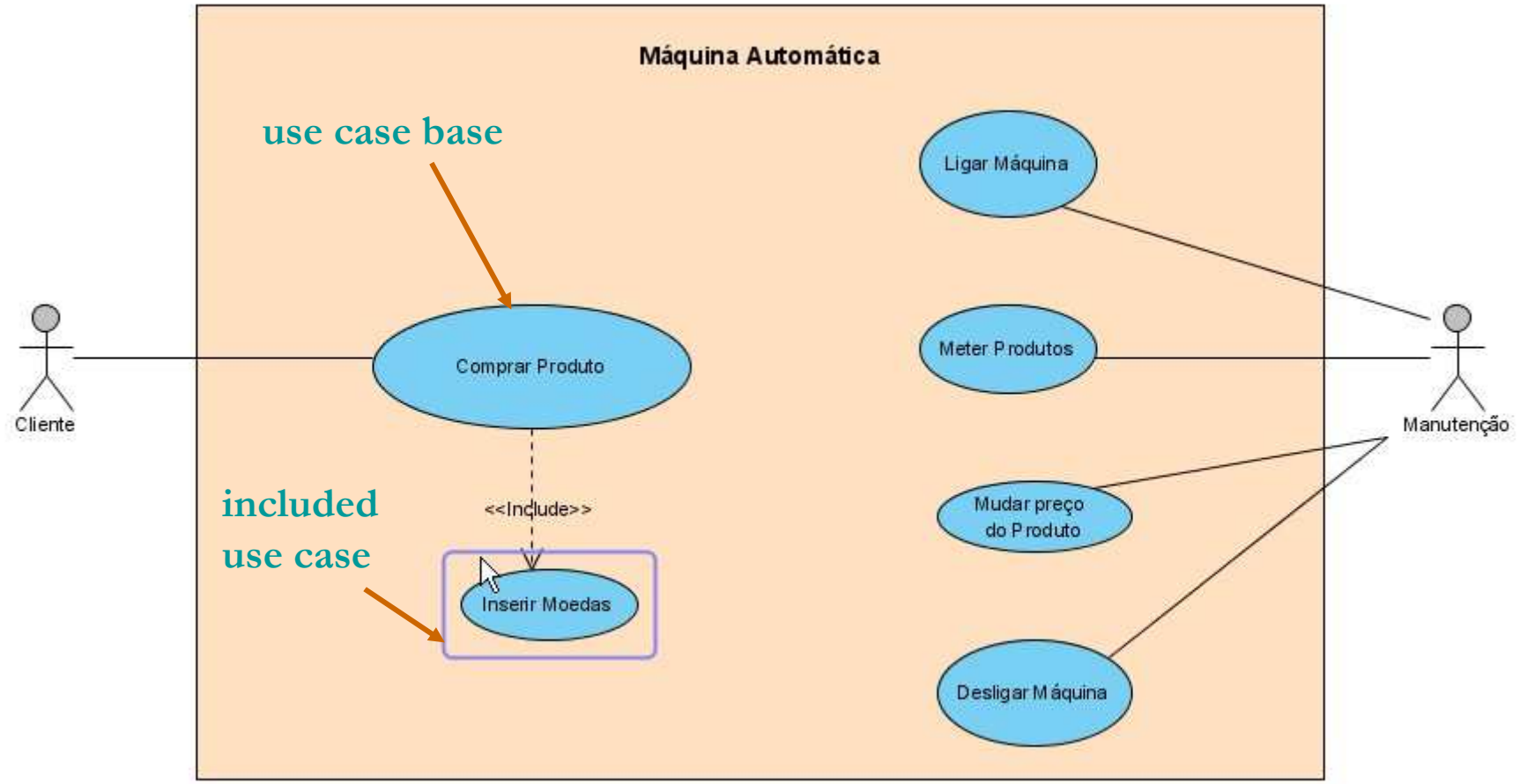


Os primeiros diagramas de Use Case (DUC) de um Sistema, descrevem apenas a funcionalidade total "esperada" do sistema, sem detalhar possíveis estruturações de use cases, nem ordem das acções, etc.

USE CASES: Importância

- São uma forma intuitiva e sistemática de capturar requisitos funcionais (o que o sistema deve oferecer aos seus utilizadores);
- São a base (ou o centro) 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 instrumentos de diálogo entre clientes e projectistas;
- Permitem até desenvolver protótipos da Interface com o Utilizador.

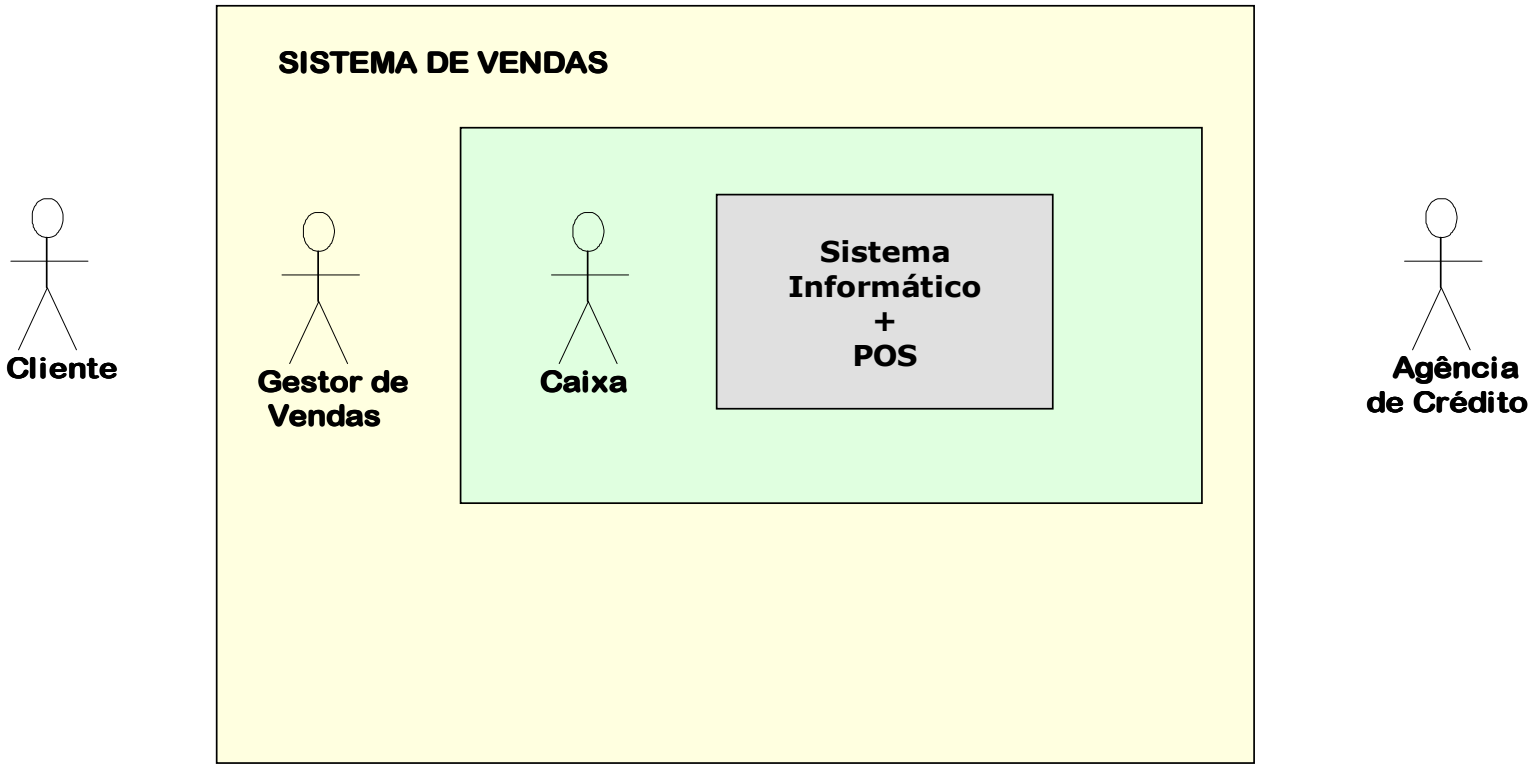
USE CASES: cont.



Só depois, gradualmente, e em função do que se vai compreendendo e decidindo, é que se introduz mais detalhe, relacionando UCs entre si, reutilizando UCs, etc.

USE CASES: continuação

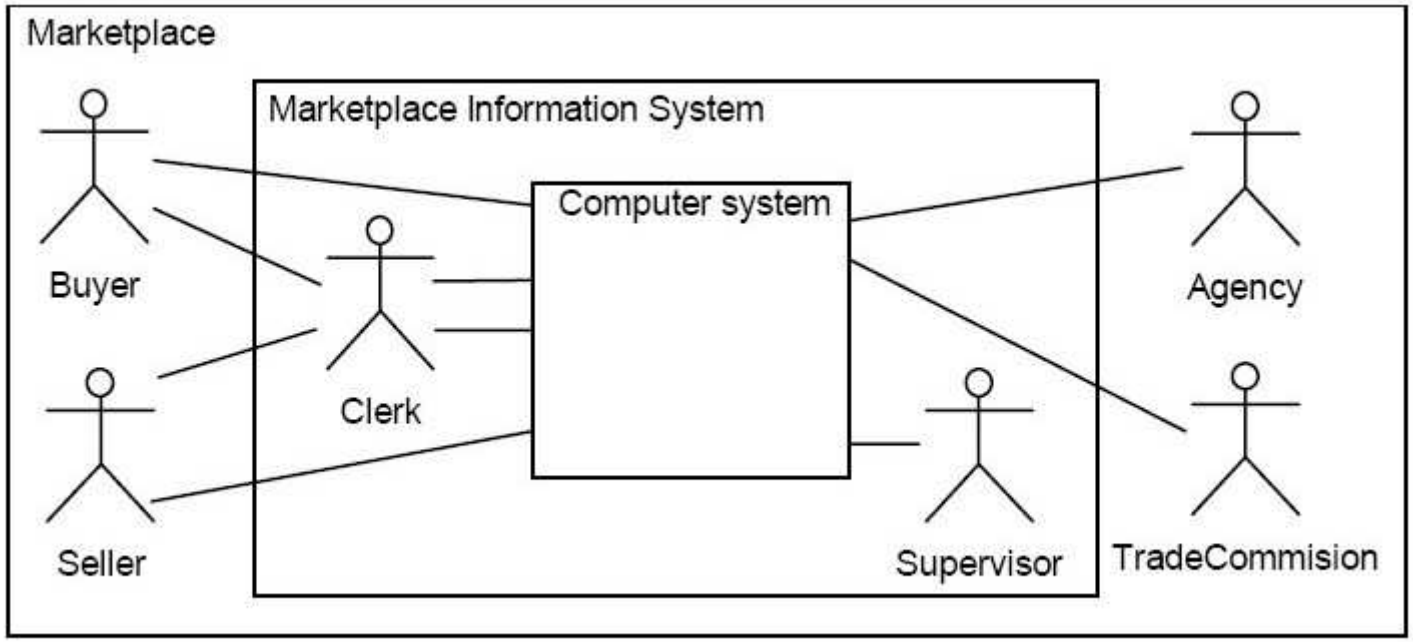
CONTEXTO



É importante ter sempre uma ideia de qual o contexto global ("scope") de funcionamento do Sistema Software que vamos desenhar e construir, e de "quem" está envolvido e porquê.

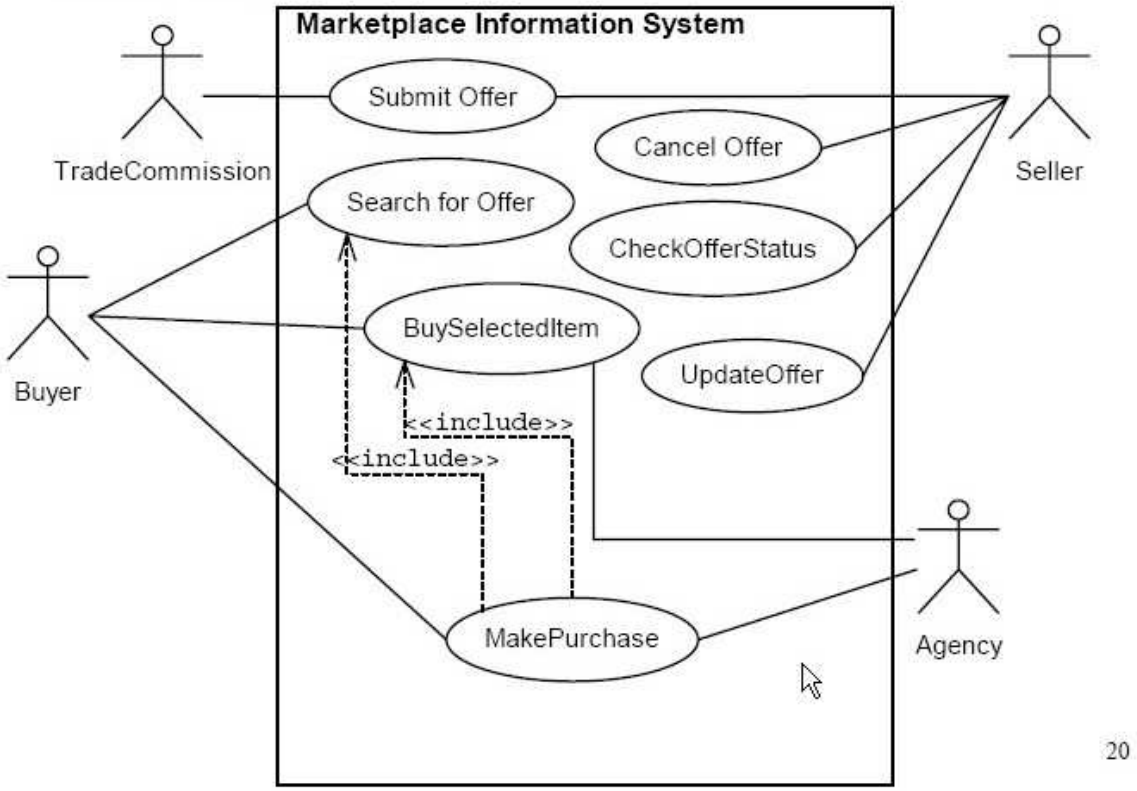
USE CASES: continuação

DIAGRAMA DE USE CASES DE CONTEXTO



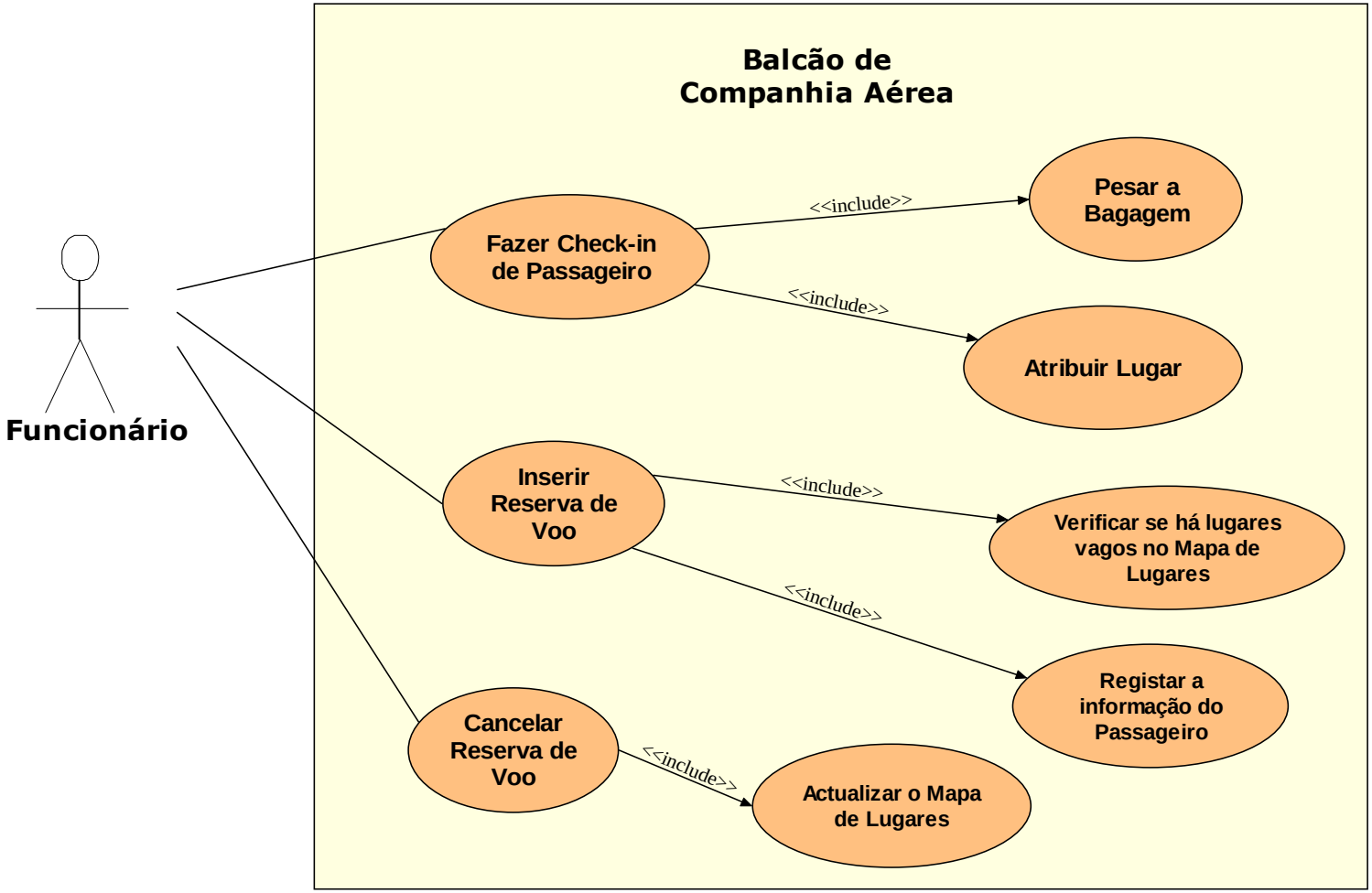
Por vezes, desenham-se DUC que envolvem várias partes da organização, para que se compreenda melhor o contexto do desenvolvimento do Sistema Sofware: Bolsa, Sistema geral de Informação da Bolsa, Sistema Sofware da Bolsa e Agências, etc.

DIAGRAMA DE USE CASES DE CONTEXTO



Por vezes define-se o DUC de todo o Sistema de Informação necessário para a organização funcionar, e apenas depois se decide o que irá ser informatizado (ou seja vai fazer parte do Sistema Software) !

USE CASES



Precisamos de agora estudar mais formas de relacionamento entre UCs e entre Actores.

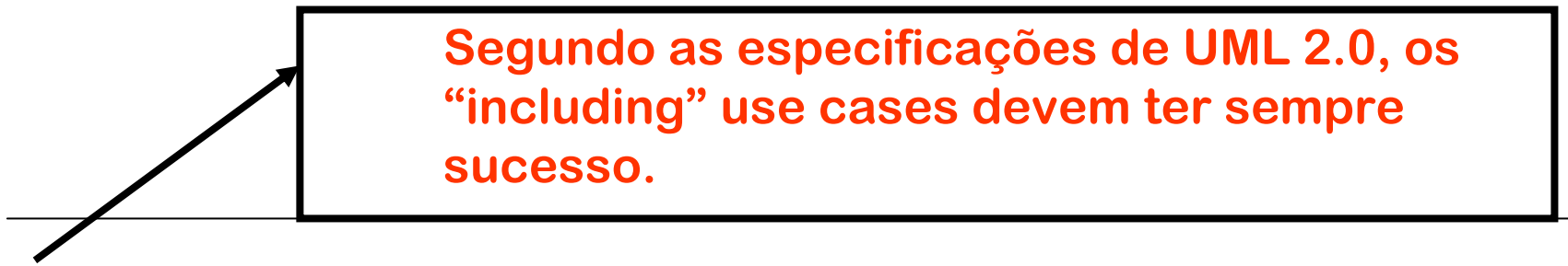
USE CASES: Relacionamentos

Estereótipos pré-definidos para UCD (extensões aos conceitos base, por vezes para tornar mais clara a sua semântica; existem pré-definidos; podem ser criados pelo modelador; são por vezes um enriquecimento semântico para o programador; palavra entre << .. >>):

<<include>> e <<extend>> são 2 relacionamentos entre Ucs.

<<include>>

Os "including" use cases não são opcionais, pois são sempre necessários para a correcta execução do "use case" base !



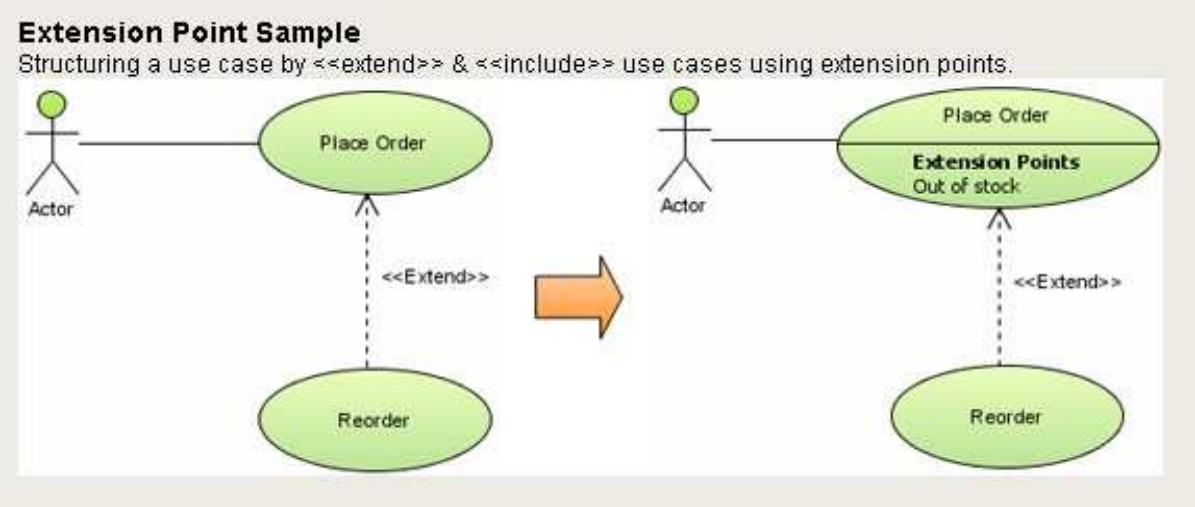
Vamos flexibilizar esta norma, tratando as excepções no UC base !!

USE CASES: Relacionamentos

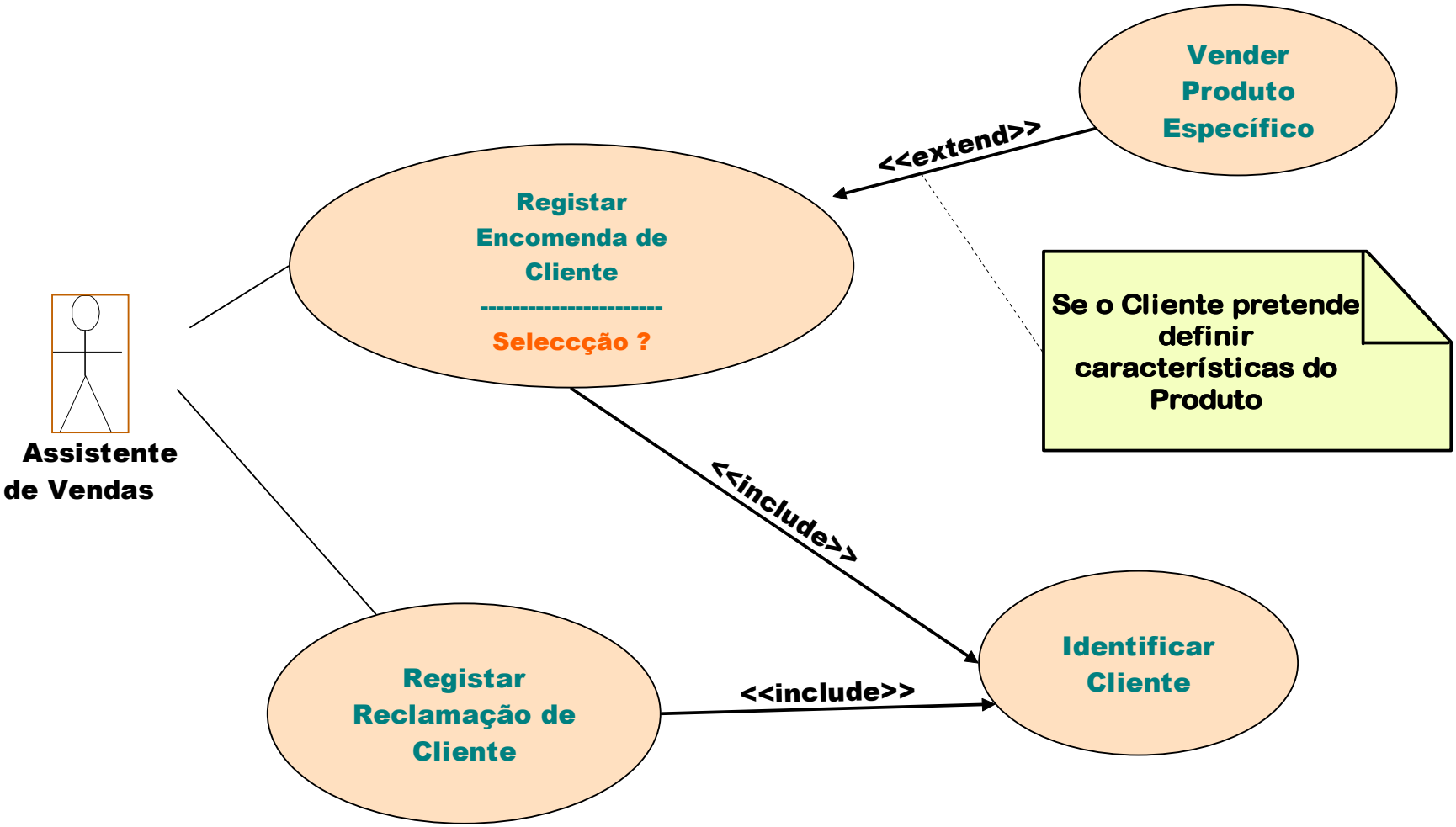
<<extend>>

Os "extending" use cases servem para, em dadas condições, ou seja, condicionalmente, acrescentarem funcionalidade ao "use case" de base (em função das **condições de extensão**).

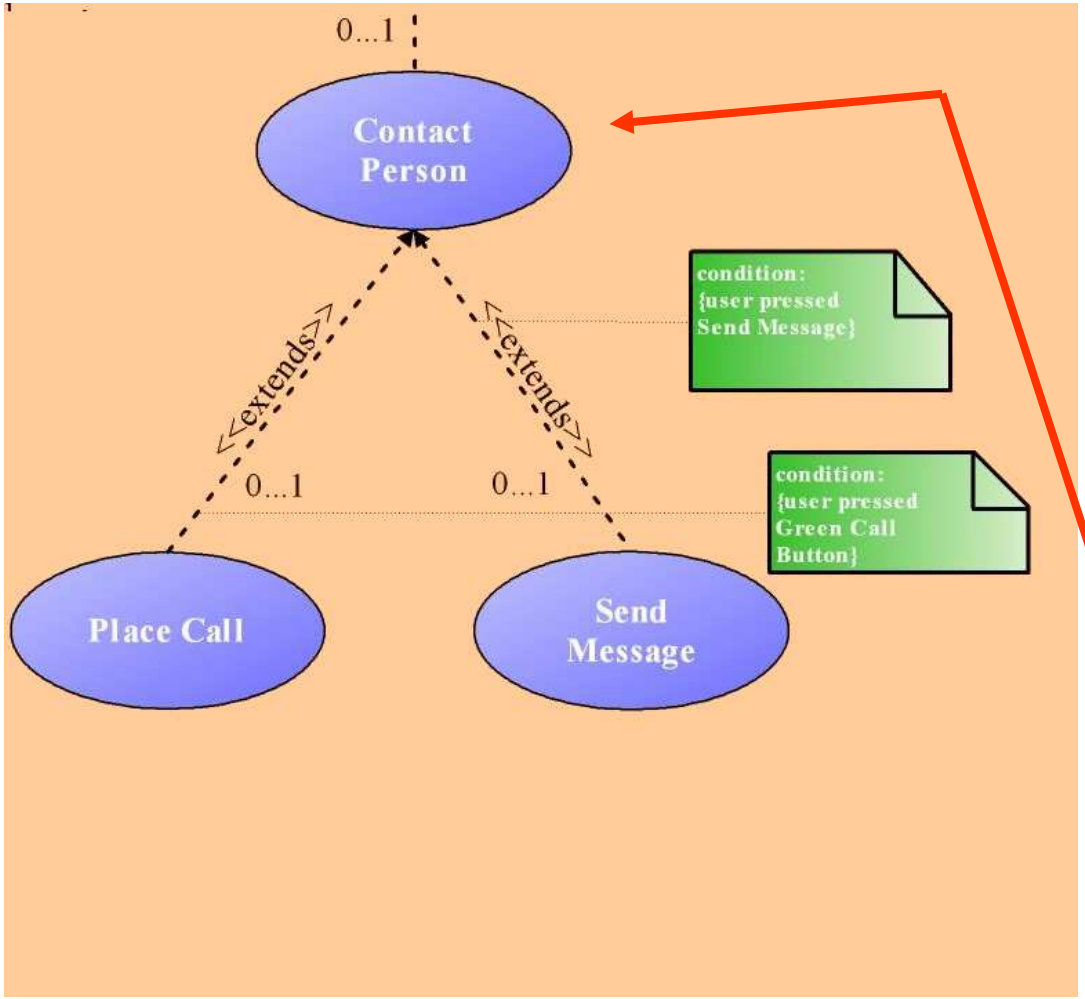
Mas, o use case de base deve possuir um comportamento que, mesmo que nenhuma extensão seja adicionada, seja um comportamento significativo e relevante, regra que nem sempre é seguida !!



USE CASES: Relacionamentos



USE CASES: <<extend>>

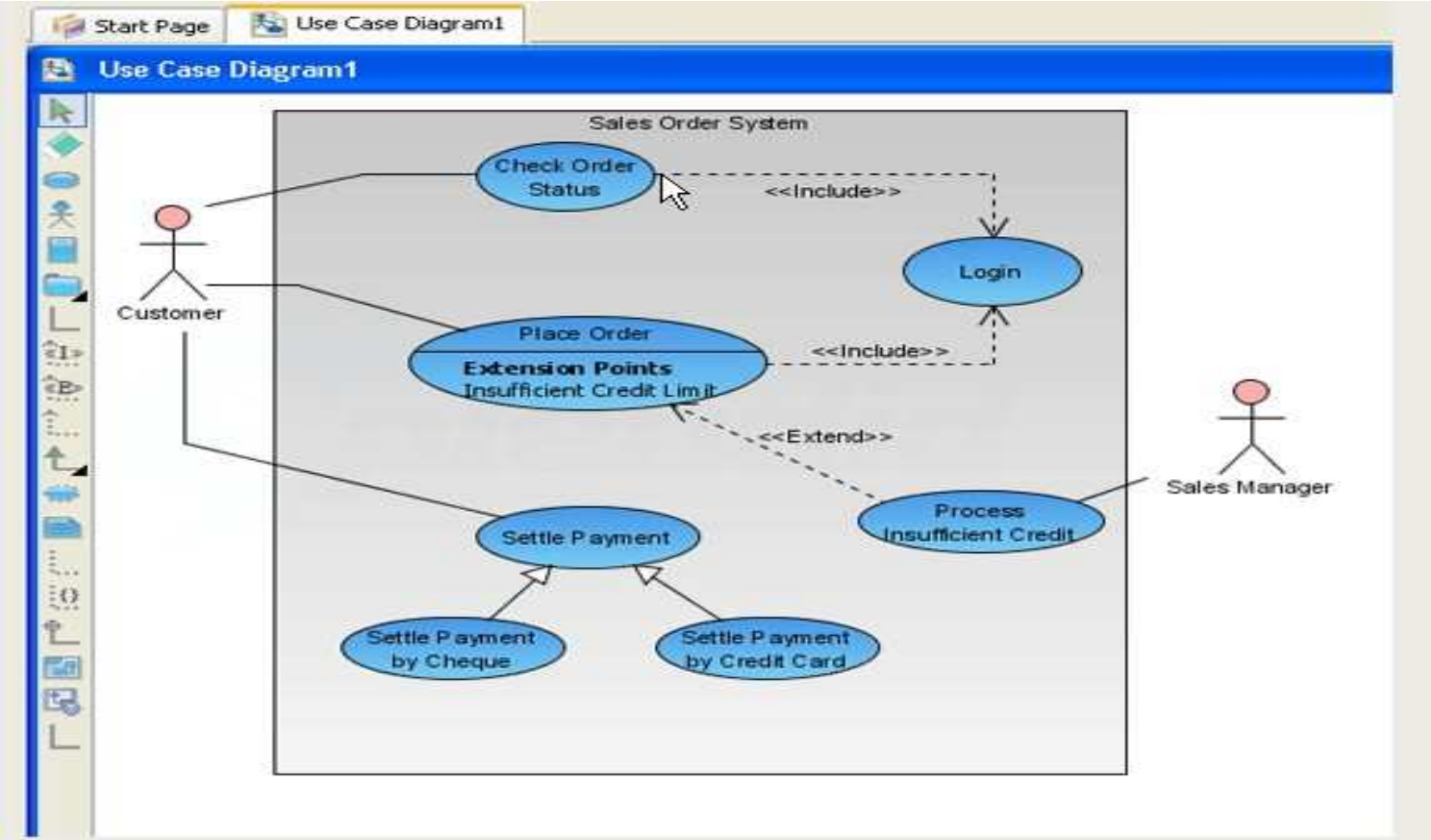


Se as condições forem disjuntas os use cases de extensão são alternativas, podendo acontecer que nenhum seja activado, cf. 0..1

Notar que o use case de extensão é que aponta para o use case de base !!

O use case base que é aumentado, deverá ter uma funcionalidade própria, significativa e independente das "potenciais" extensões

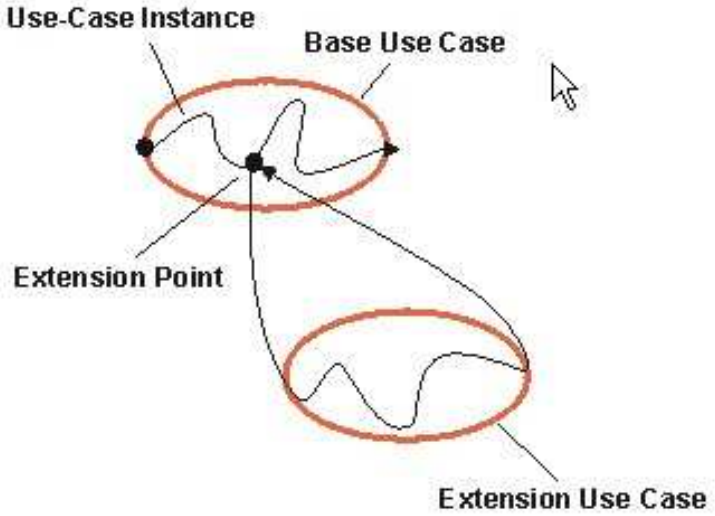
USE CASES: Extension Points

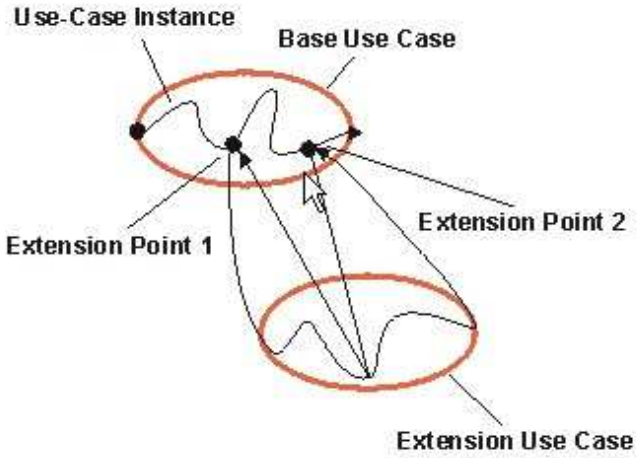


Um "extended" Use Case, pode ter o seu comportamento aumentado ou não, dependendo das condições definidas nos designados "extension points".

USE CASES: Extension Points

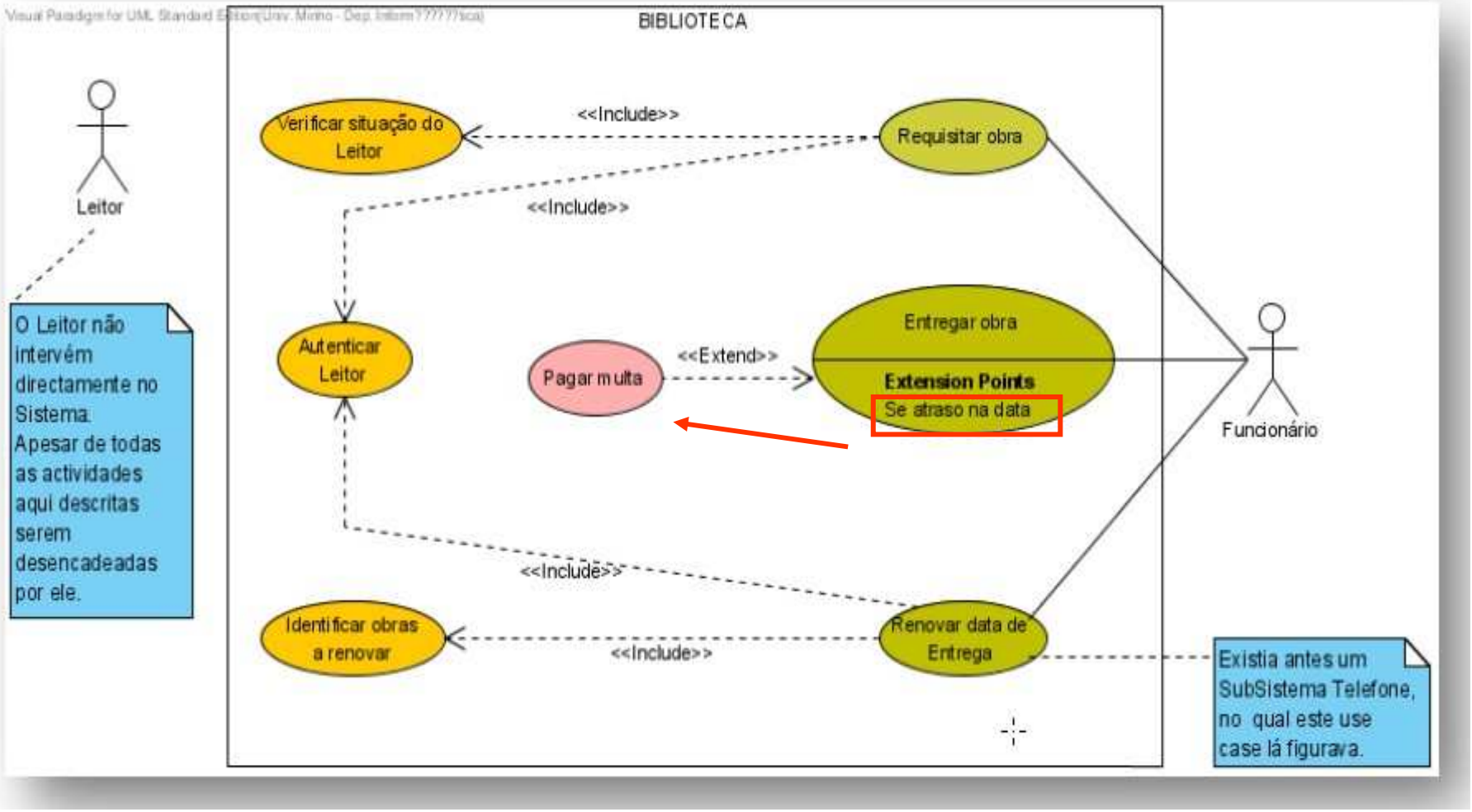
Semântica operacional correcta de <<extend>>



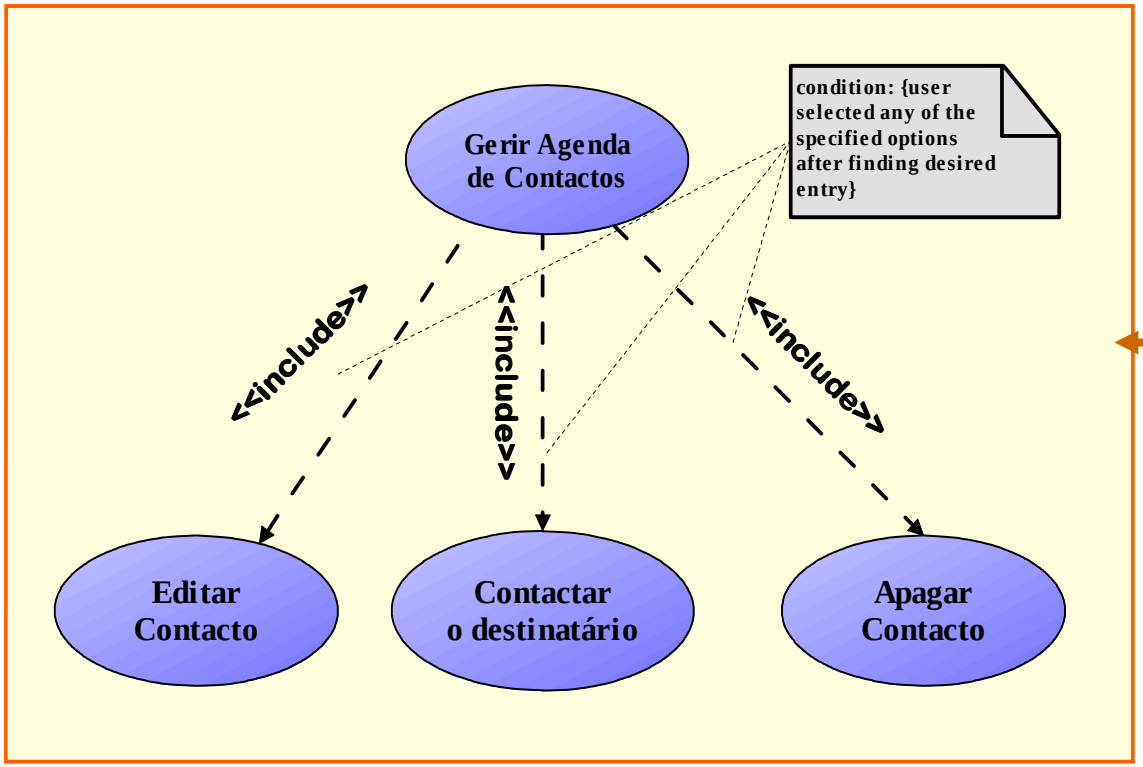


Se as condições de teste de extensão especificadas nos “extension points” forem verdadeiras, então serão realizadas as operações definidas nos respectivos use cases de extensão.

USE CASES: <<extend>>



Utilizações erradas de <<include>>

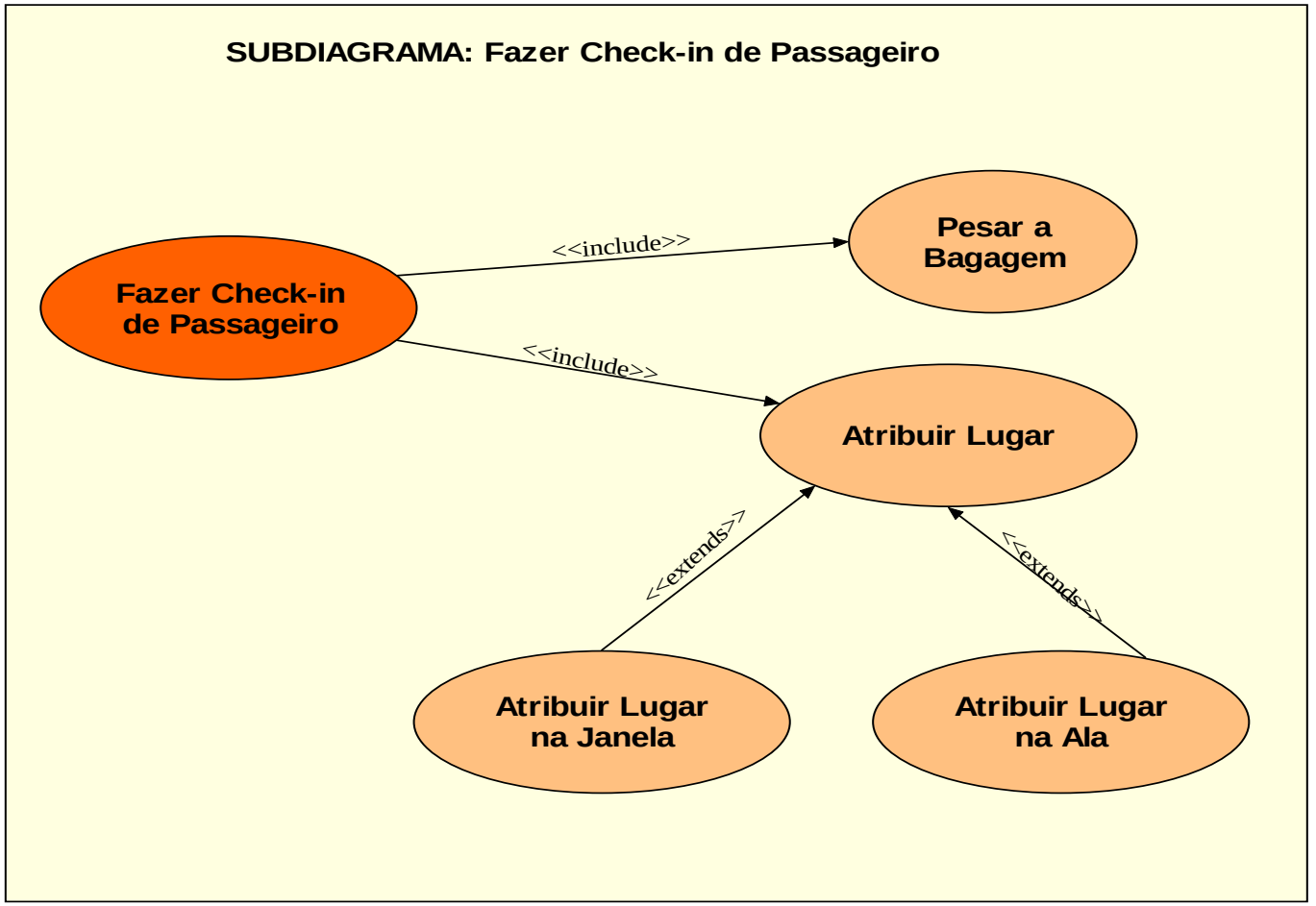


Pretendido:

O "use case" base pode ser aumentado por um qualquer dos sub use-cases, mas não os inclui a todos, podendo mesmo não incluir nenhum.

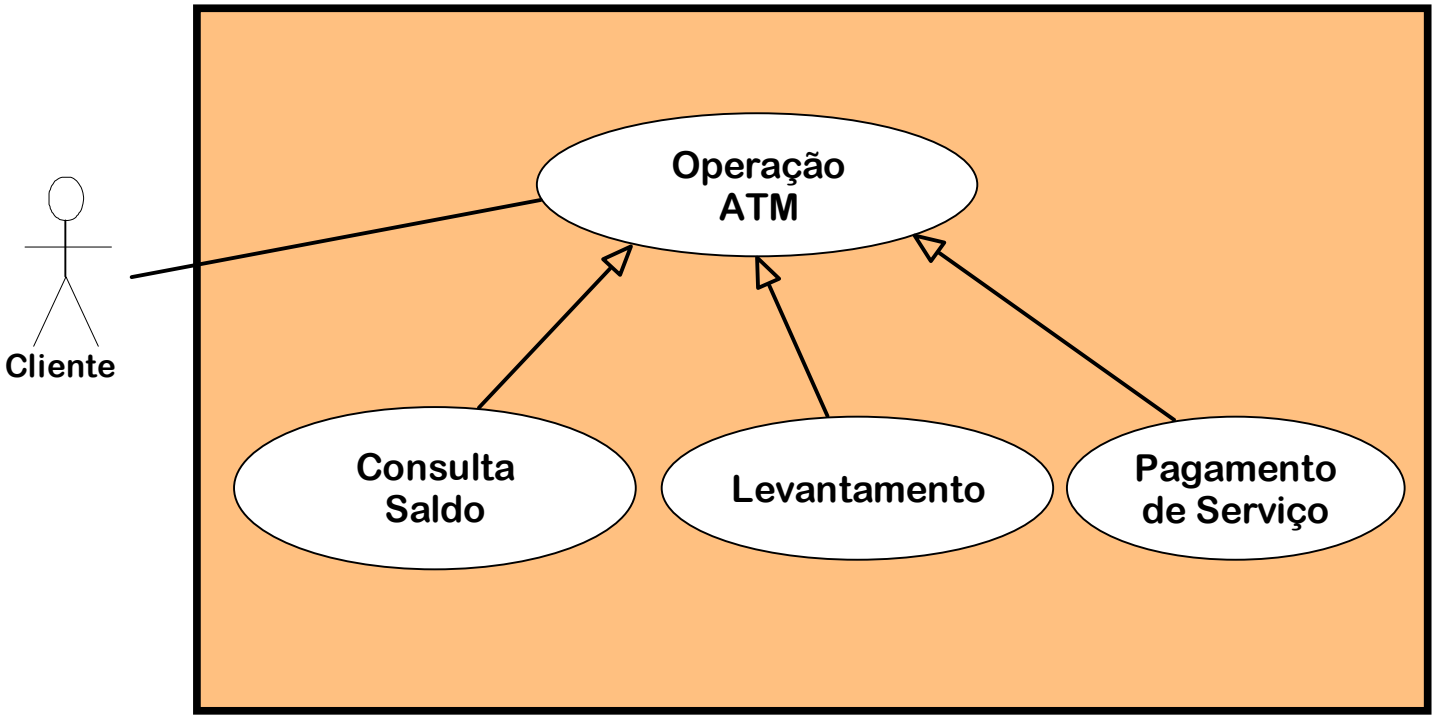
▣ O que está especificado não é isto, mas sim que o "use case base" deve incluir TODOS os sub. A "nota" e as setas estão erradas. Atenção !!

USE CASES: Subdiagramas



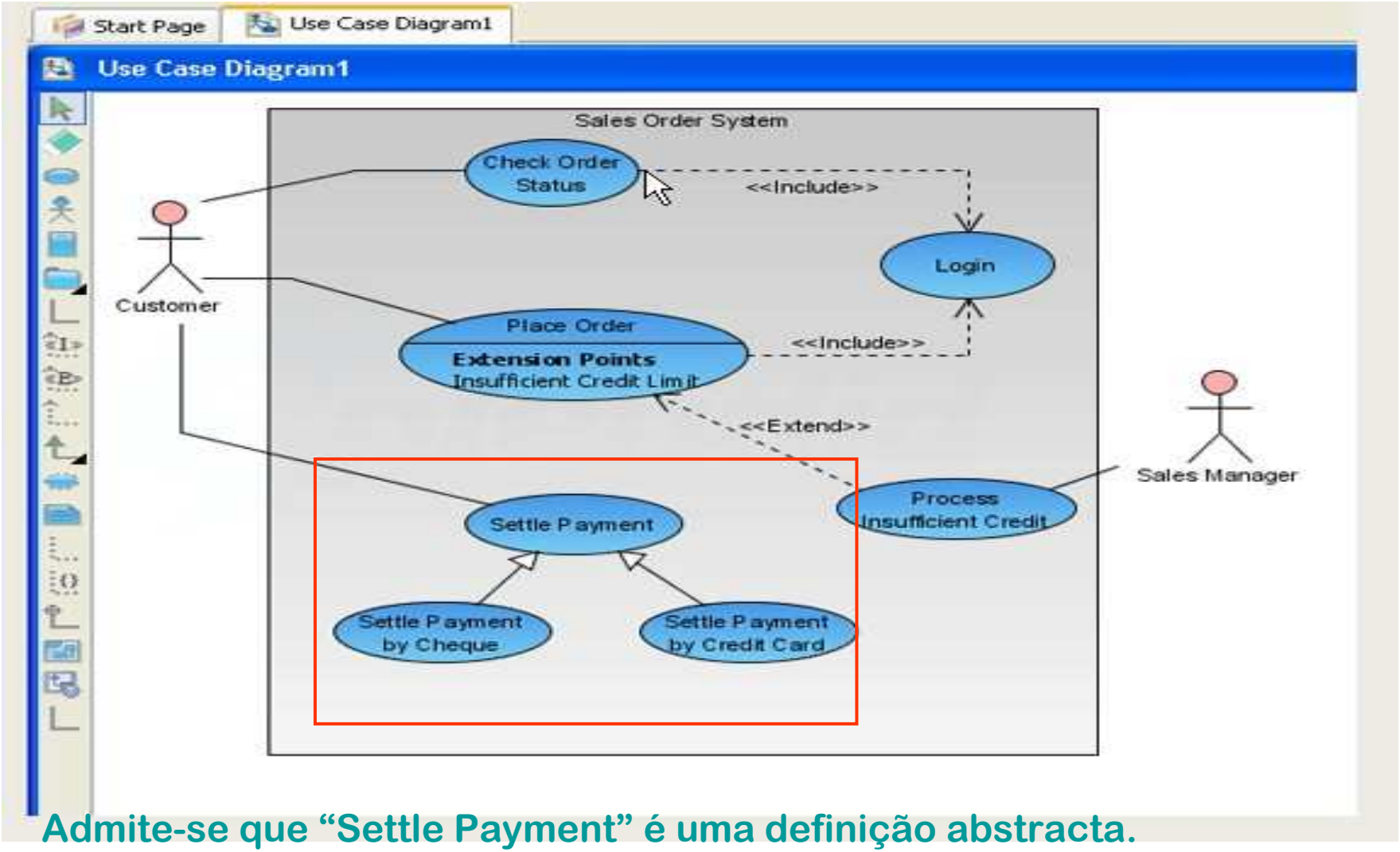
Podemos "esconder" os detalhes usando subdiagramas (modularidade e abstracção).

USE CASES: Generalização



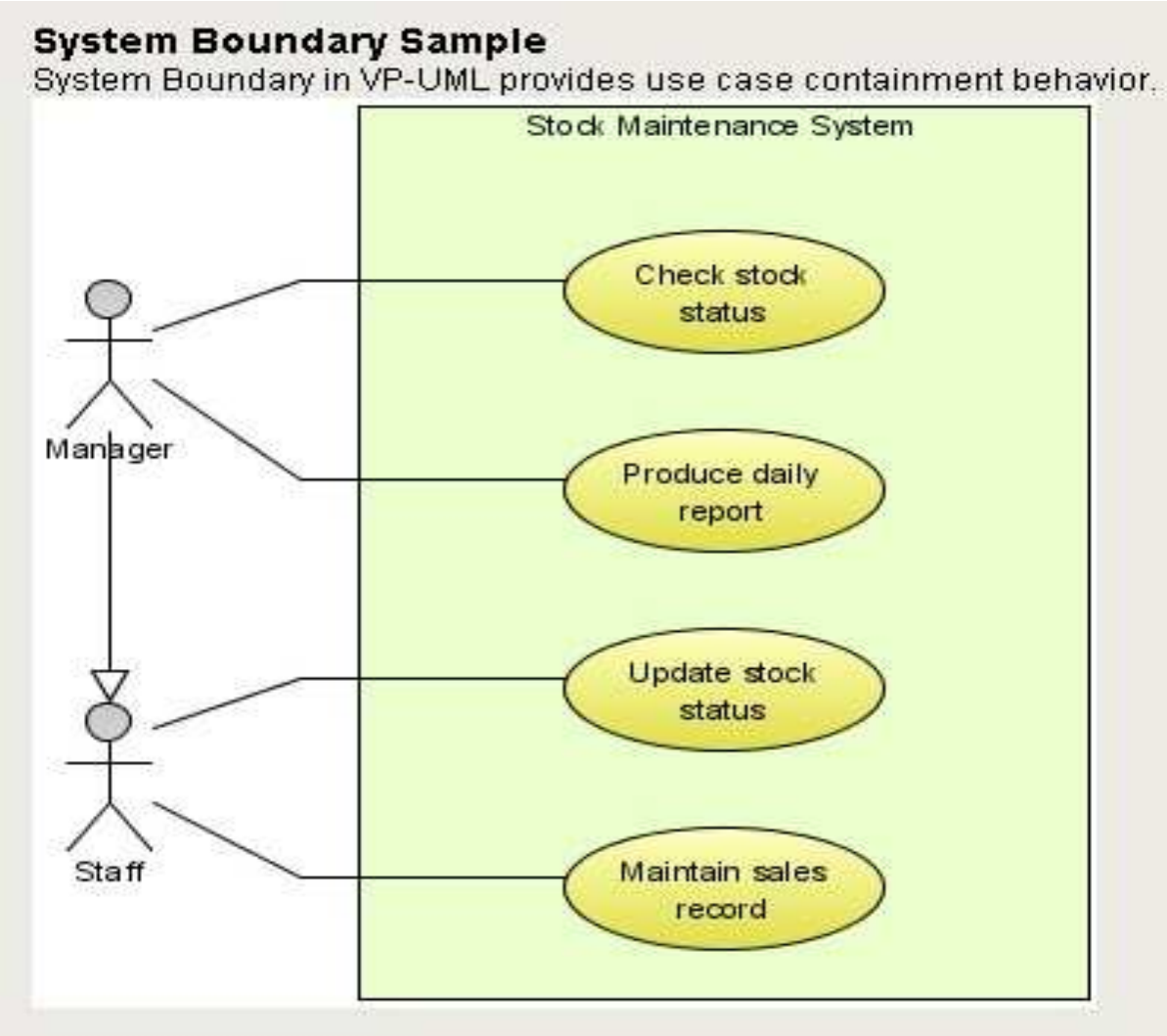
▣ **Generalização** permite expressar relacionamento **is-a**, subclassificação, herança e polimorfismo. O use case base pode assumir uma qualquer das formas (comportamentos) expressas nos sub-use cases (tal como indica o princípio da substituição em OO).

USE CASES: Generalização



Admite-se que "Settle Payment" é uma definição abstracta.
Mas o polimorfismo de Use Cases é DESACONSELHADO !!

ACTORES: Generalização



Generalização de Actores

Corresponde a uma clarificação de papéis e responsabilidades perante o sistema ou subsistema.

Assim, quem assume certo "perfil" tem acesso a e tem responsabilidade sobre certas actividades

ACTORES: Generalização

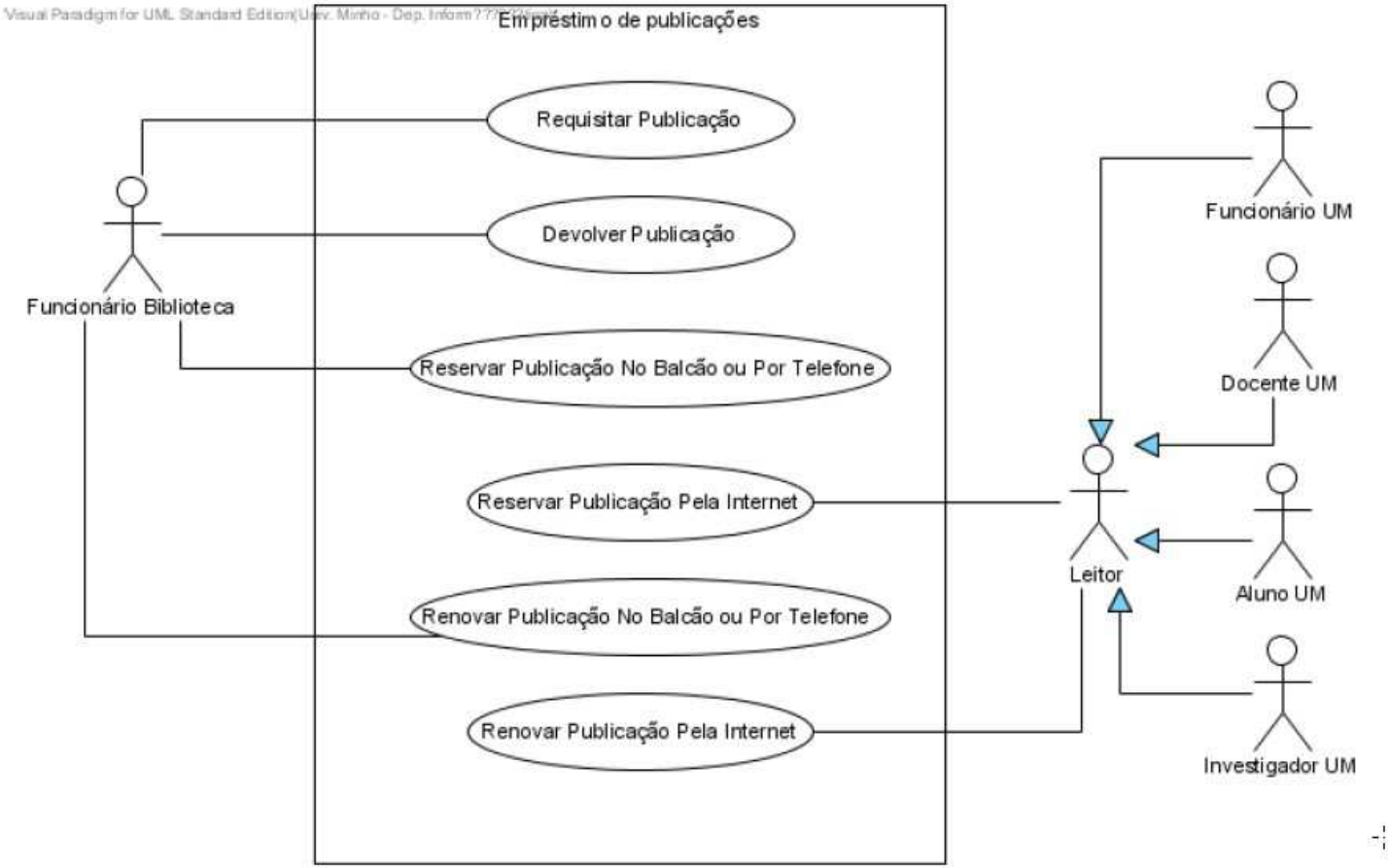
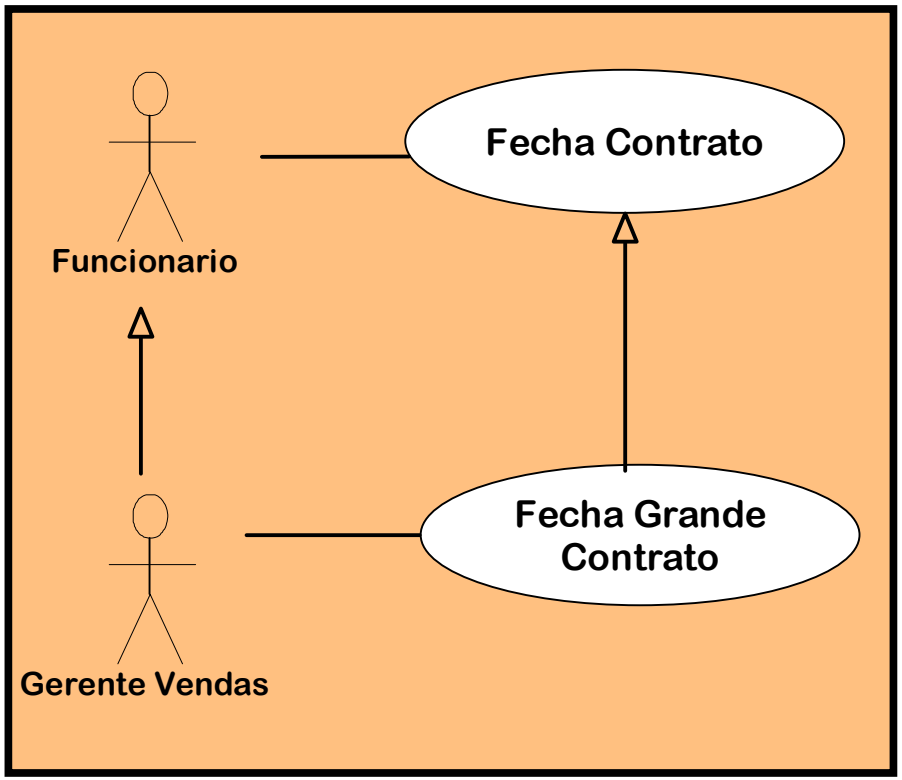


Figura 1: Diagrama de Use Cases do Sistema de Empréstimo de publicações dos SDUM

Diversos TIPOS de LEITOR, mas com os mesmos objectivos.

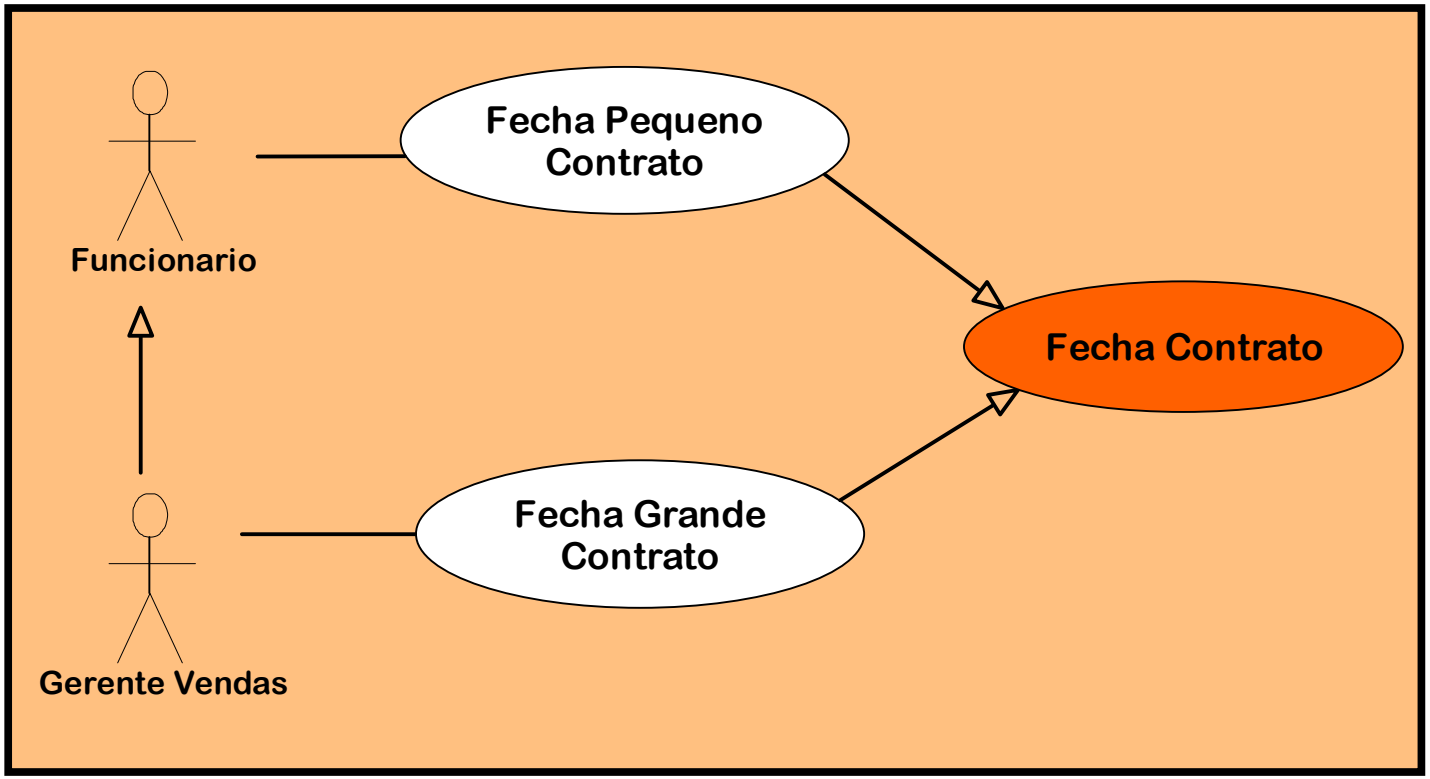
Problemas com a generalização de Actores e UCs



Herança => Polimorfismo, pelo que "Fecha Contrato" é substituível por "Fecha Grande Contrato". Assim, o simples Funcioário pode fechar grandes contratos, tal como o Gerente.

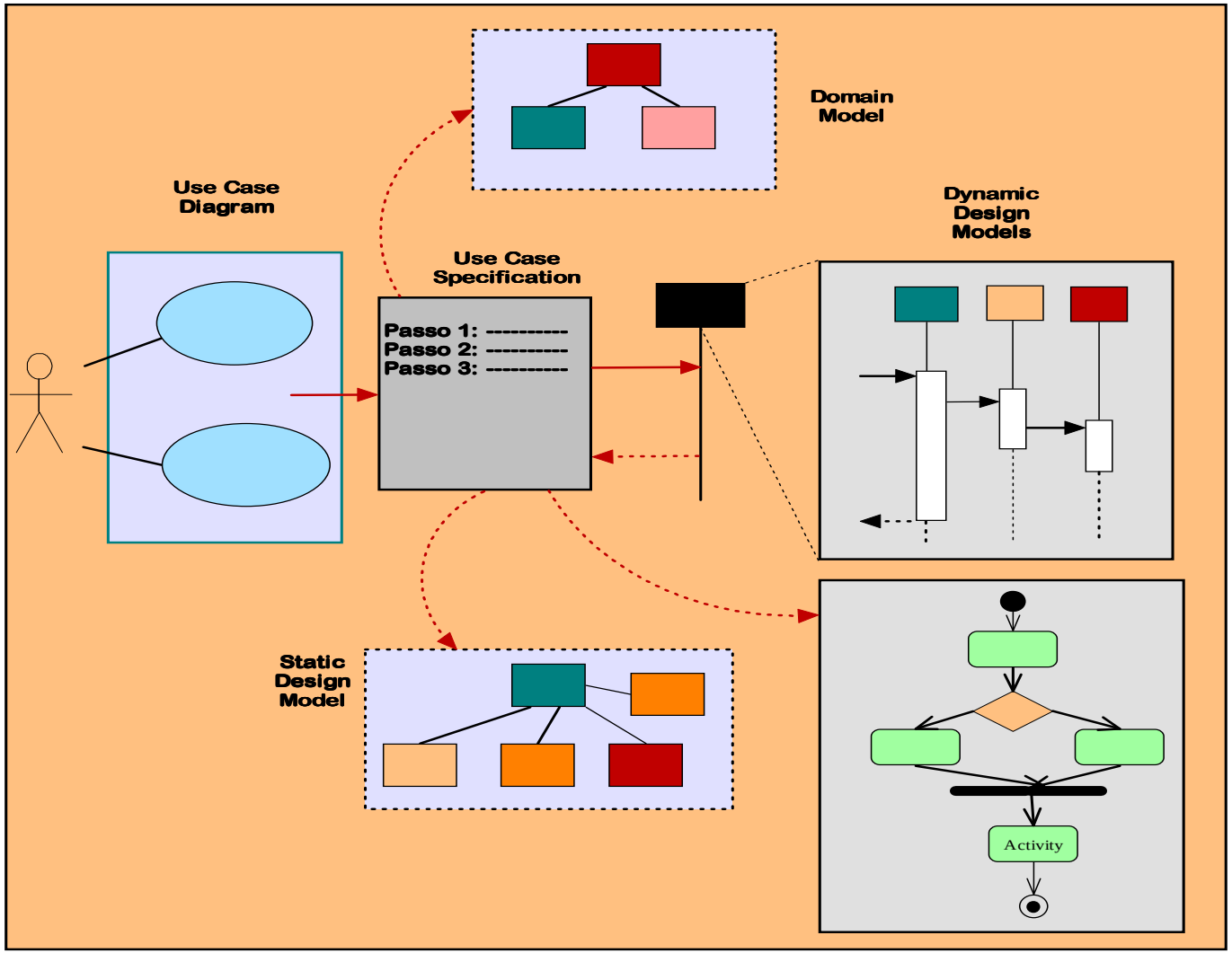
Não era certamente isto que se pretendia especificar !

Solução (Generalização das tarefas com "profiling" adequado):



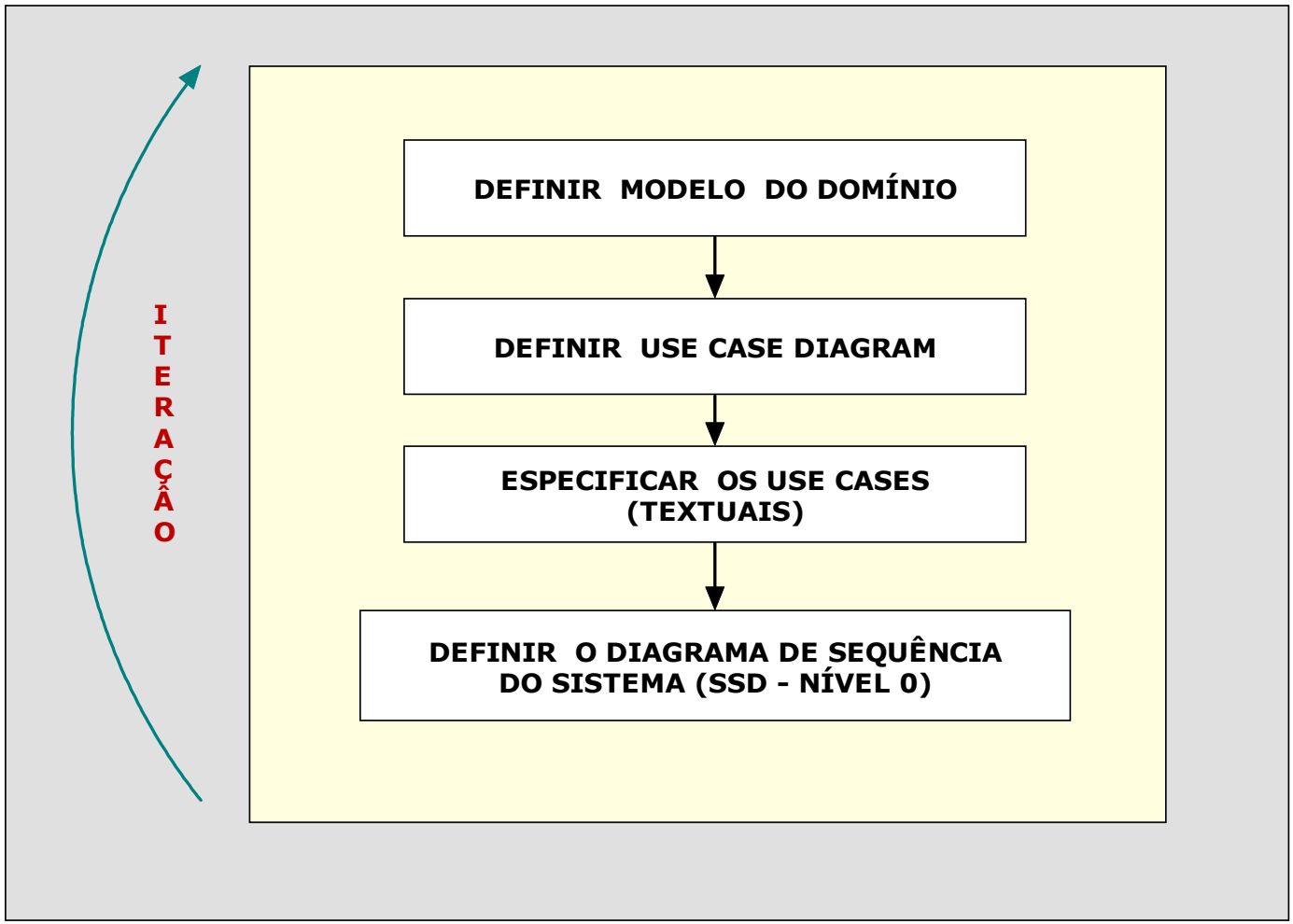
Situação pouco comum em que se justifica generalização de UC.

METODOLOGIA



Use Cases são vistos como os elementos centrais do processo de análise (cf. Jacobson e RUP)

Requisitos funcionais



Modelos do Domínio

O que é um Domínio ?

O termo **Domínio** denota, em Engenharia Informática, mas não só, um conjunto de sistemas ou áreas funcionais, dos vários sectores organizacionais de actividade, ou da sociedade em geral, onde a existência de uma terminologia (**sintaxe ou termo**) deve estar inequivocamente associada a certos conceitos, o que em muito simplifica a compreensão e, em consequência, a correcta comunicação entre quem tem que "definir contractos de informatização" e quem tem que "realizar tais contractos" sno âmbito (domínio) de tais sistemas, áreas, etc.

Alguns Domínios em que a Engª Informática tem produzido "produtos":

Quase todos ou todos. Quem indica excepções ?

Exemplos de Domínios típicos do âmbito da Engª Informática:

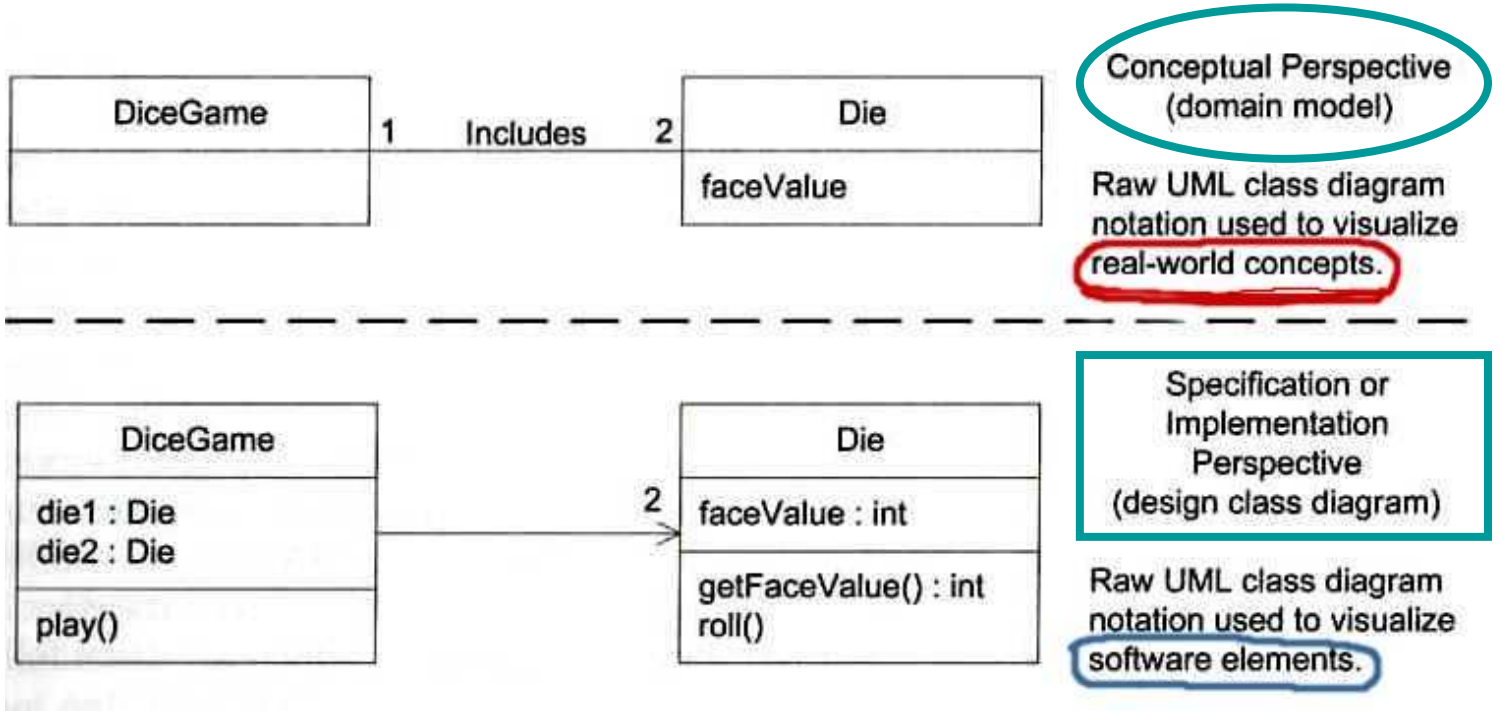
Bancário; Aviónica; Construção Civil; Administração Pública; Telecomunicações; Medicina e Clínica Médica; Biotecnologia; Segurança, enfim, todas as outras Engenharias e, actualmente, todas as outras áreas, cf. Arqueologia, Museus, Documentação, Administrativo, Gestão, etc.

- **Como modelar um Domínio numa perspectiva OO ?**

1) **Organizando e definindo todo o vocabulário "fundamental" do domínio do problema, em especial o que sobressai da análise de requisitos com os interlocutores numa tabela semântica, ou seja, um dicionário terminológico aceite e assinado por ambas as partes;**

2) **Organizando e relacionando termos que estão definidos num glossário ou num dicionário de dados firmado, se a complexidade do problema o justificar, definindo as classes que representam as entidades d**o estado interno persistente e partilhado do sistema, como sendo as entidades (e eventos) do negócio, e suas respectivas ligações semânticas a outras entidades, bem como as classes **que modelam a estrutura de documentos (que são dados estruturados) trocados entre o sistema e o seu ambiente;**

Modelos do Domínio

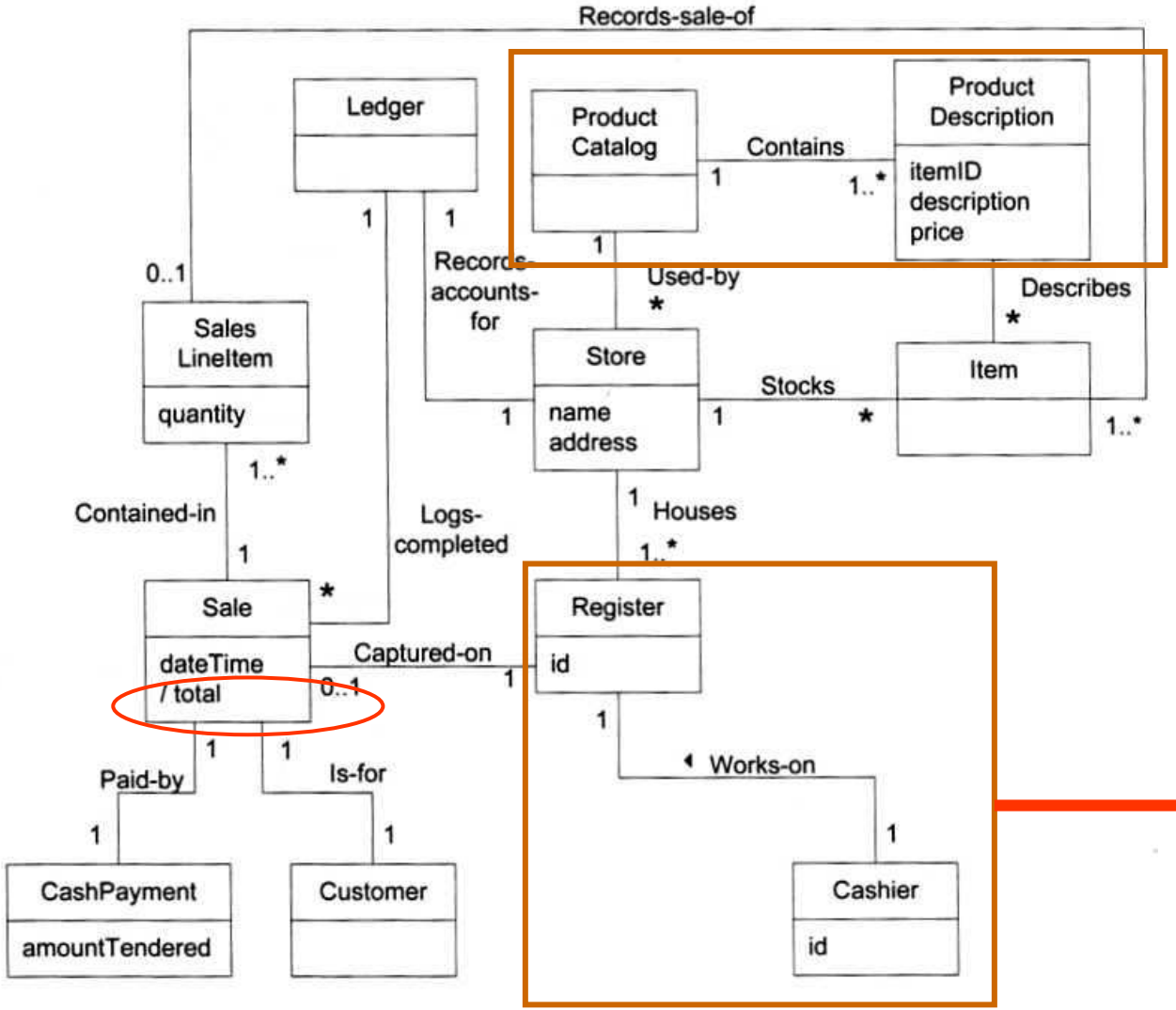


Jogo de Dados

- Classes, atributos e relacionamentos servem em UML para modelar o domínio do sistema, tal como, com mais detalhe, servem para modelar os elementos sofware.
- Porém, **Modelos de Domínio** são Conceptuais e **Diagramas de Classe** são já especificações/implementações com vista à codificação!

Atenção !

Modelos do Domínio: POS



Associações entre classes podem possuir:

Nome

Contains, Paid-by, etc.

Multiplicidade:

1 – um, 0 - zero

* - mais de 1

1..* - 1 ou mais

Navegabilidade, ou seja, orientação:

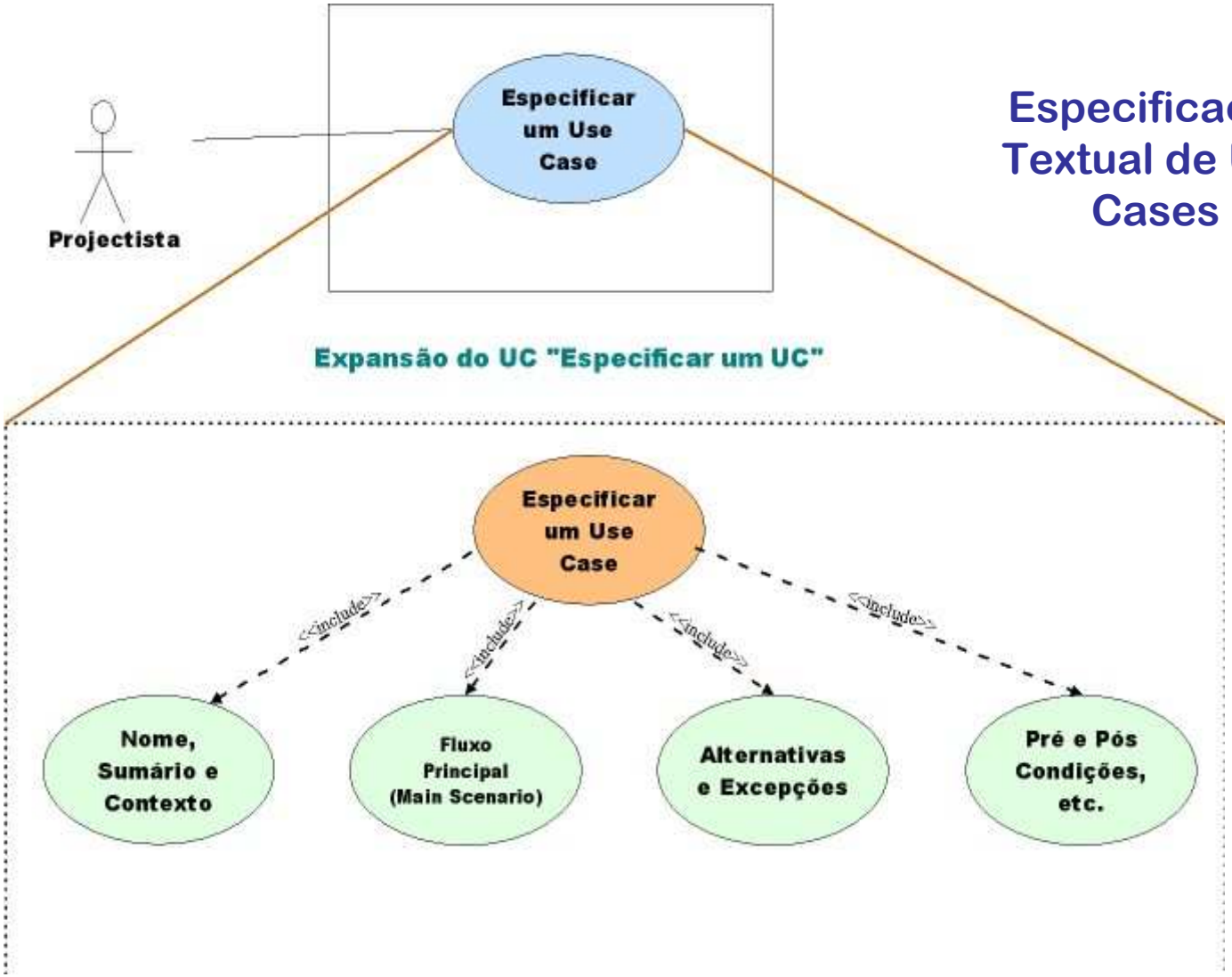
"Cada funcionário de caixa trabalha, a cada momento, numa máquina com identificação única".

UMA ENORME, E POR ISSO INTERESSANTE REFLEXÃO SOBRE A DIFICULDADE EM ENCONTRAR DISPONÍVEIS, ONDE QUER QUE SEJA, NAS EMPRESAS DE SW, NA INTERNET, ETC., MODELOS DE DOMÍNIOS ANALISÁVEIS OU, PELO MENOS, DADOS COMO EXEMPLOS QUE POSSAM SER APRESENTADOS NUMA AULA DE ENGª INFORMÁTICA, É SINTOMÁTICA DE QUE A MAIORIA DAS ORGANIZAÇÕES NÃO TÊM NEM MODELOS DE DOMÍNIO NEM DE NEGÓCIO.

PORQUE SERÁ ??

PORTANTO, TEMOS QUE SER NÓS, ENGENHEIROS DE SW, A INFERI-LOS, ANALISÁ-LOS E DEPOIS, SE POSSÍVEL, A MELHORÁ-LOS E A IMPLEMENTÁ-LOS.

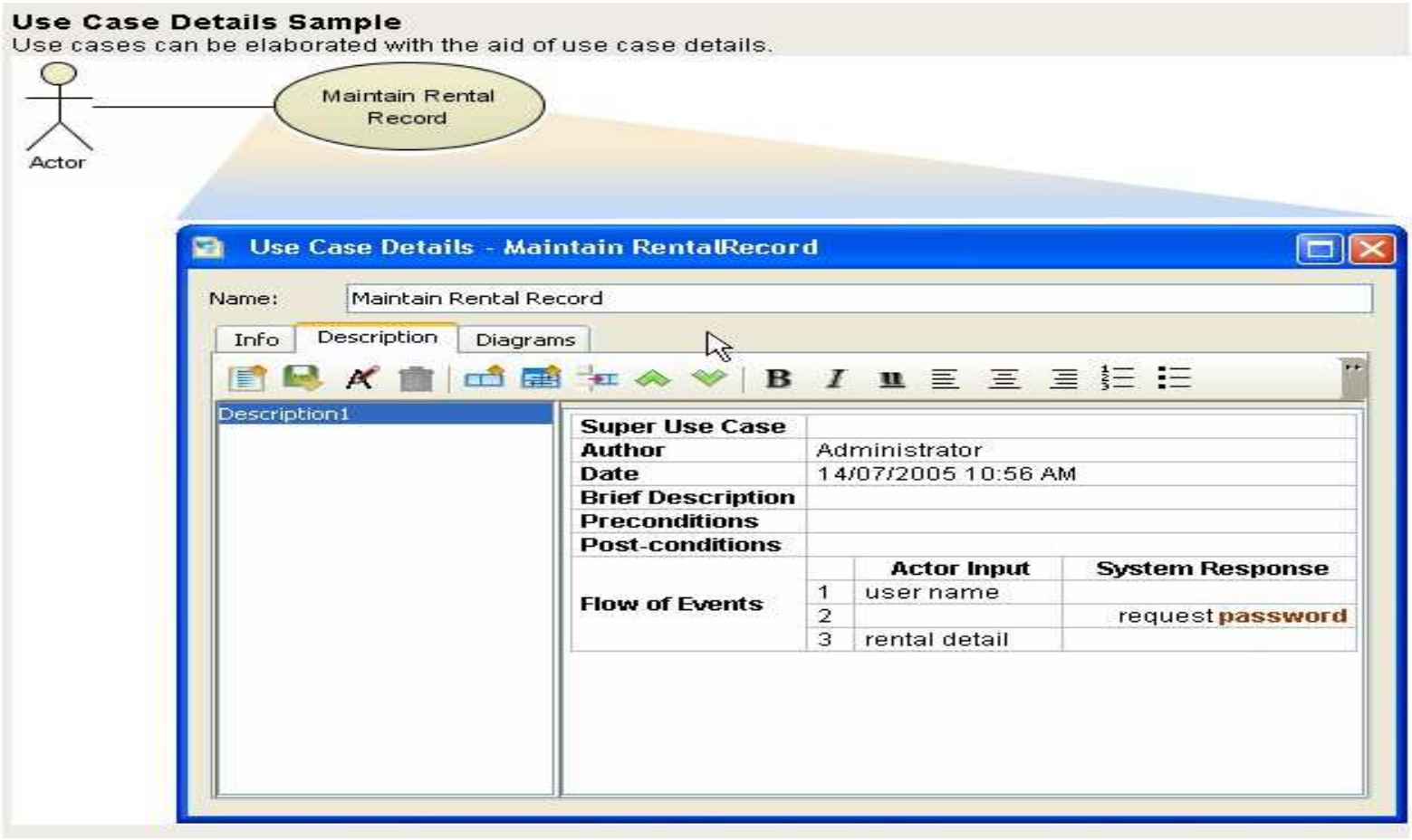
Como especificar UC



Especificação Textual de Use Cases

Especificação Textual de UC

O formato é livre, cf. UML 2.0.
Nós usaremos o template do Visual Paradigm, mas com algumas modificações.



A (single) Use Case: textual form

Use Case: #1 Seller submits an offer
Scope: Marketplace
SuD: Marketplace Information System
Level: Primary Task
Primary Actor: Seller
Supporting Actor: Trade Commission
Main success scenario specification:

1. Seller submits information describing an item
2. System validates the description.
3. Seller adjusts/enters price and enters contact and billing information.
4. System validates the seller's contact information.
5. System verifies the seller's history to permit the seller to operate
6. System validates the whole offer with the Trade Commission
7. System lists the offer in published offers.
8. System responds with an uniquely identified authorization number.

Extensions: Excepções

2a Item not valid
  2a1 Use case aborted
5a Seller's history inappropriate
  5a1 Use case aborted
6a Trade commission rejects the offer
  6a1 Use case aborted

Sub-variations: Alternatives

2b Price assessment available
  2b1 System provides the seller with a price assessment.

Vladimír Mencl
From Textual Use Cases to Behavior Specifications, Berlin, Apr 19, 2004

Apenas 1 exemplo ...

Especificação Textual de UC

USE CASE #		*Identificador: Frase contendo o verbo*
Objectivo no contexto		*Objectivo do use case por palavras*
Âmbito e Nível		*Que sistema é tomado como "black box"?*
	<nível>	<Geral; Tarefa; Subfunção>
PRÉ-CONDIÇÃO		*Estado do sistema antes do início do use case*
ACTORES	<prim>	*Nome do papel do actor primário (o que inicia o UC)*
	<sec>	*Outros sistemas envolvidos*
NORMAL (Sucesso)	**PASSO**	**ACÇÃO**
	1	
	2	*passos do cenário desde o início até se atingir o objectivo,*
	3	*podendo conter <<include>> de use cases, pontos de*
	4	extensão e
	5	
	....	
EXCEPÇÕES	**PASSO**	**ACÇÃO DE EXCEPÇÃO**
	x.a	*Condição causadora da ramificação*
		Acção ou nome do sub-usecase
ALTERNATIVAS	**PASSO**	**ACÇÃO**
	x.a	*Condição causadora da ramificação*
		Acção ou nome do sub-usecase
PÓS-CONDIÇÃO	pc-suc	*Estado do sistema após execução com sucesso do use case*
	pc-erro	*Pode conduzir a estados de erro ? Quais ?*
Use Cases Superiores		
Sub-use cases		
OUTRA INFORMAÇÃO		
Frequência (dia/mês)		
Canais dos Actores		*Telefone, e-mail, carta, ficheiro, ..*
Data Limite de Conclusão		
Data Actual		
Versão actual		
Autores		

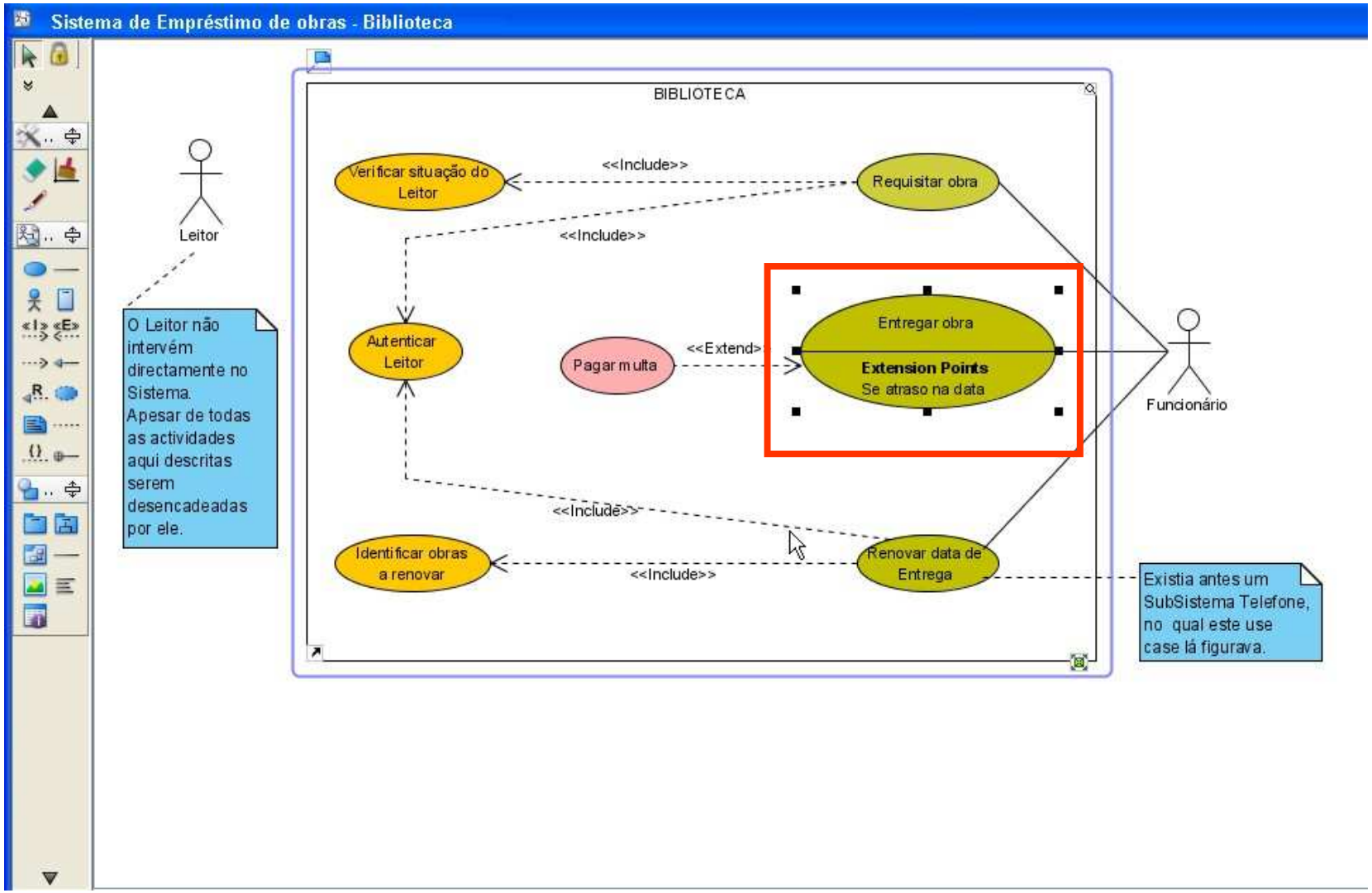
O NOSSO MODELO

Especificação do Fluxo Principal

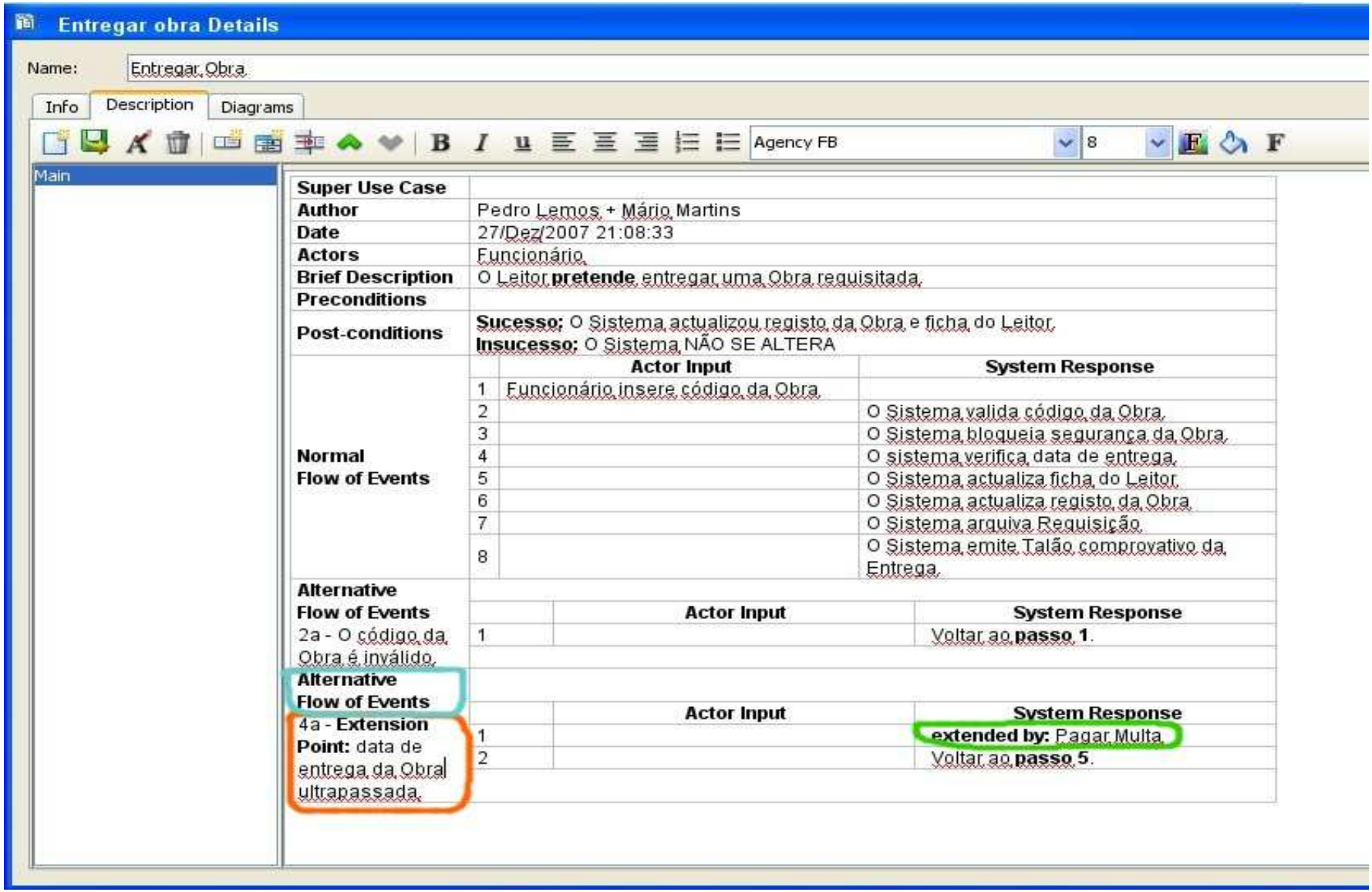
Especificação de fluxos de mau comportamento ou erro, eventualmente recuperáveis.

Especificação de Fluxos de alternativos de sucesso.

Especificação Textual de UC



Especificação Textual de UC



Entregar obra Details

Name: Entregar Obra

Info | Description | Diagrams

Main

Super Use Case			
Author	Pedro Lemos + Mário Martins		
Date	27/Dez/2007 21:08:33		
Actors	Funcionário		
Brief Description	O Leitor **pretende** entregar uma Obra requisitada.		
Preconditions			
Post-conditions	**Sucesso:** O Sistema actualizou registo da Obra e ficha do Leitor. **Insucesso:** O Sistema NÃO SE ALTERA		
Normal Flow of Events		**Actor Input**	**System Response**
	1	Funcionário insere código da Obra.	
	2		O Sistema valida código da Obra.
	3		O Sistema bloqueia segurança da Obra.
	4		O sistema verifica data de entrega.
	5		O Sistema actualiza ficha do Leitor.
	6		O Sistema actualiza registo da Obra.
	7		O Sistema arquiva Requisição.
	8		O Sistema emite Talão comprovativo da Entrega.
Alternative Flow of Events			
2a - O código da Obra é inválido.		**Actor Input**	**System Response**
	1		Voltar ao **passo 1**.
Alternative Flow of Events			
4a - **Extension Point:** data de entrega da Obra ultrapassada.		**Actor Input**	**System Response**
	1		**extended by:** Pagar Multa.
	2		Voltar ao **passo 5**.

Especificação Textual de UC

Próxima aula:

- Normalização da especificação textual de Use Cases;
- Especificação usando UC de uma Máquina ATM (ou parte dela).

- Pensem durante a próxima semana no que podem actualmente "fazer" (obter "goals") , enquanto utilizadores, numa actual máquina ATM !
- Mas não se esqueçam da importância do Modelo do Domínio;