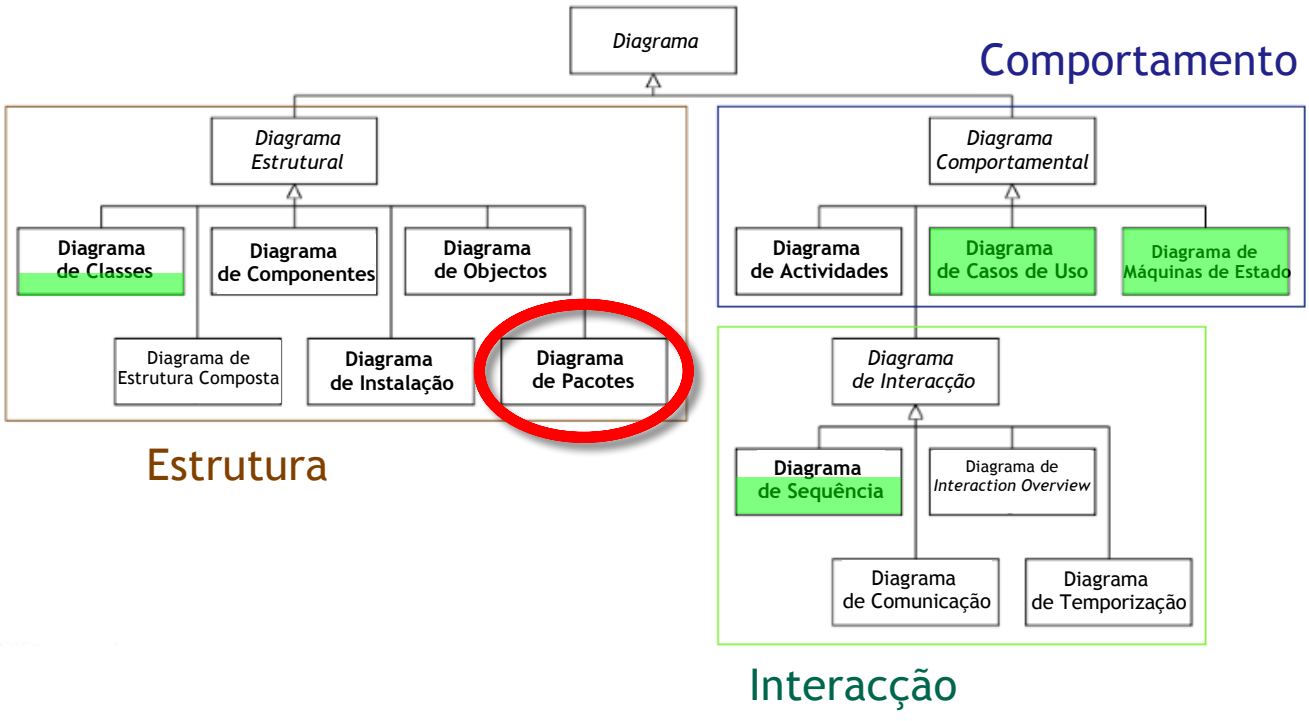
Desenvolvimento de Sistemas Software

Aula Teórica 14

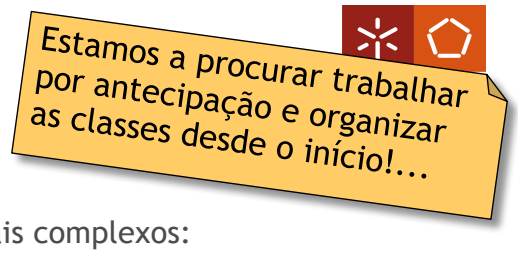
Modelação Estrutural / Diagramas de Package

Desenvolvimento de Sistemas Software
José Creissac Campos / António Nestor Ribeiro

Diagramas da UML 2.x



Diagramas de Package



- À medida que os sistemas software se tornam mais complexos:
  - Torna-se difícil efectuar a gestão de um número crescente de classes
  - A identificação de uma classe e o seu papel no sistema dependem do contexto em que se encontram
  - É determinante conseguir identificar as dependências entre as diversas classes de um sistema.

- Em UML os agrupamentos de classe designam-se por *packages* (pacotes) e correspondem à abstracção de conceitos existentes nas linguagens de programação:
  - Em Java esses agrupamentos são os *packages*
  - Em C++ designam-se por *namespaces*

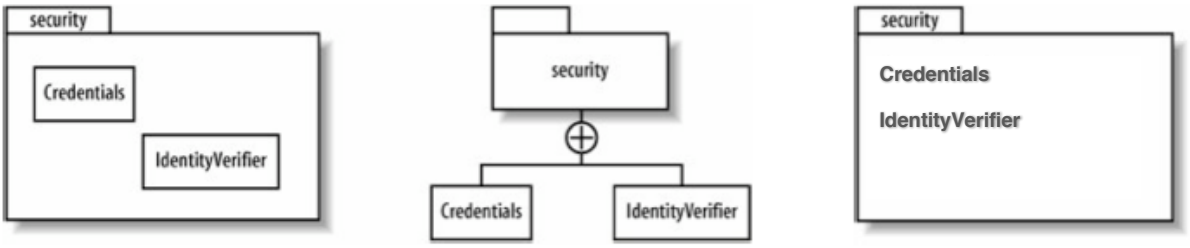
- A identificação das dependências entre os vários packages permite que a equipa de projecto possa descrever informação importante para a evolução do sistema

Diagramas de Package (cont.)

- Um diagrama de package representa os packages e as relações entre packages
- Os diagramas de packages em UML representam mais do que relações entre classes:
  - Packages de classes (packages lógicos) - em diagramas de classes
  - Packages de componentes – em diagramas de componentes
  - Packages de nós – em diagramas de distribuição
  - Packages de casos de uso – em diagramas de *use cases*
- Um package assim o dono de um conjunto de entidades, que vão desde outros packages, a classes, interfaces, componentes, use cases, etc.

Diagramas de Package (cont.)

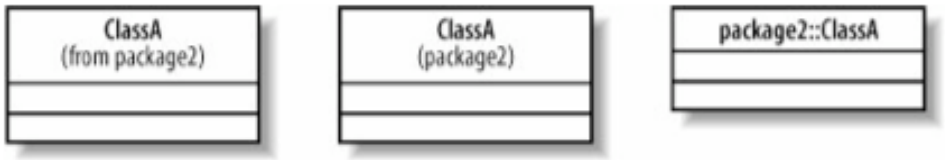
- Essa forma de agregação pode ser representada de diversas formas:




```
package security;
public/private/protected(?) class Credentials {
      ...
}
```


Diagramas de Package (cont.)

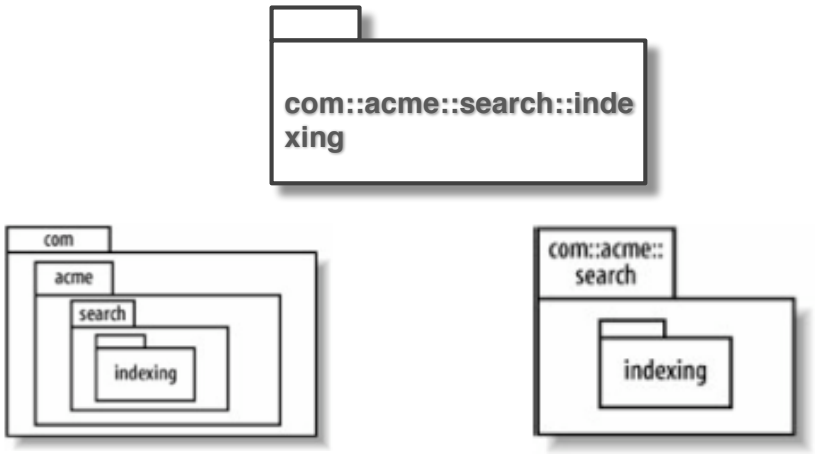
- Por uma questão de identificação do contexto de uma classe (o seu *namespace*) é usual que as ferramentas identifiquem no diagrama de classes, qual é o agrupamento lógico a que uma classe pertence.



- A notação **nomePackage::nomeClasse**, identifica (qualifica) inequivocamente uma classe.
  - Tal como em Java com a utilização do nome completo (ex: java.lang.String)
  - Permite que existam classes com nome idêntico nas diversas camadas que constituem uma aplicação

Diagramas de Package (cont.)

- Existem várias formas de representar a agregação de packages
- Essas regras definem também a qualificação dos nomes das classes

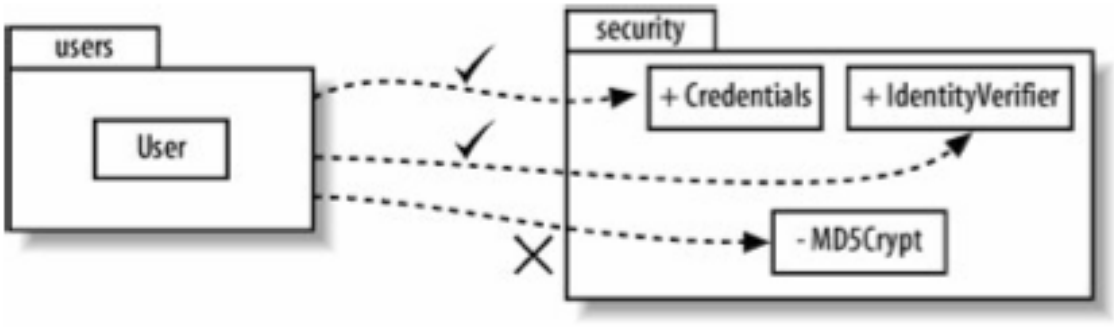


Os três diagramas representam a mesma informação

Desenvolvimento de Sistemas Software
José Creissac Campos / António Nestor Ribeiro

Diagramas de Package (cont.)

- A definição da visibilidade dos elementos de um package utiliza a mesma notação e semântica vista nos diagramas de classe:
  - "+" - público
  - "-" - privado
  - "#" - protected (só acessível/visível por filhos do package em causa)



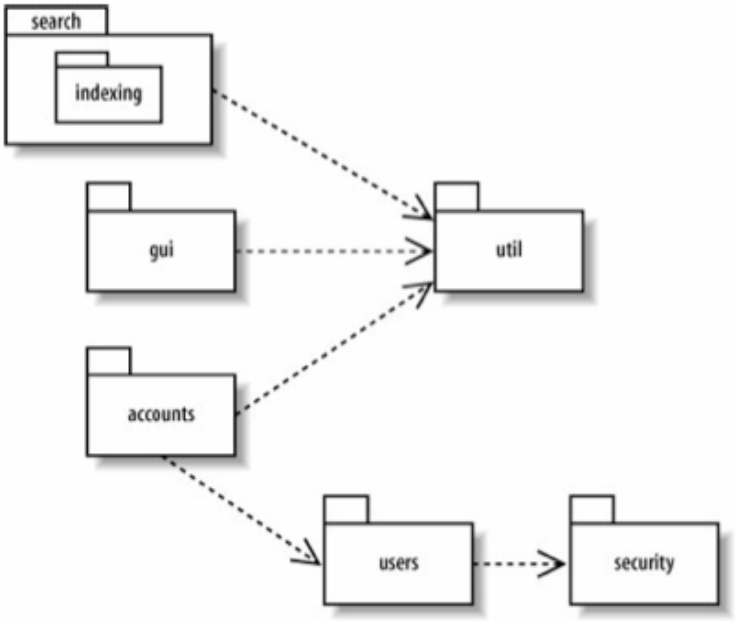
v. 2017/18

Diagramas de Package (cont.)

- Existem várias formas de especificar *dependências* entre pacotes:
  - *Dependência (simples)* - quando uma alteração no package de destino afecta o package de origem
  - <<*import*>> - o package origem importa o contéudo público do package destino. Logo, não necessita de qualificar completamente os elementos importados (na forma packageA::classeB).
    - Mecanismo similar ao que permite fazer import de um package em Java
      `import java.util.*;`
  - <<*access*>> - o package origem acede a elementos exportados pelo package destino, mas necessita de qualificar completamente os nomes desses elementos.
  - <<*merge*>> - o package origem é *fundido* com o package destino para gerar um novo.

Diagramas de Package (cont.)

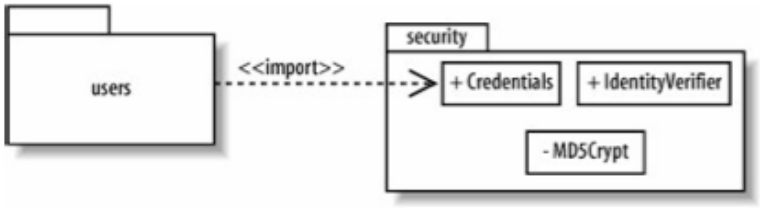
- Exemplos de utilização de dependências simples:



Diagramas de Package (cont.)

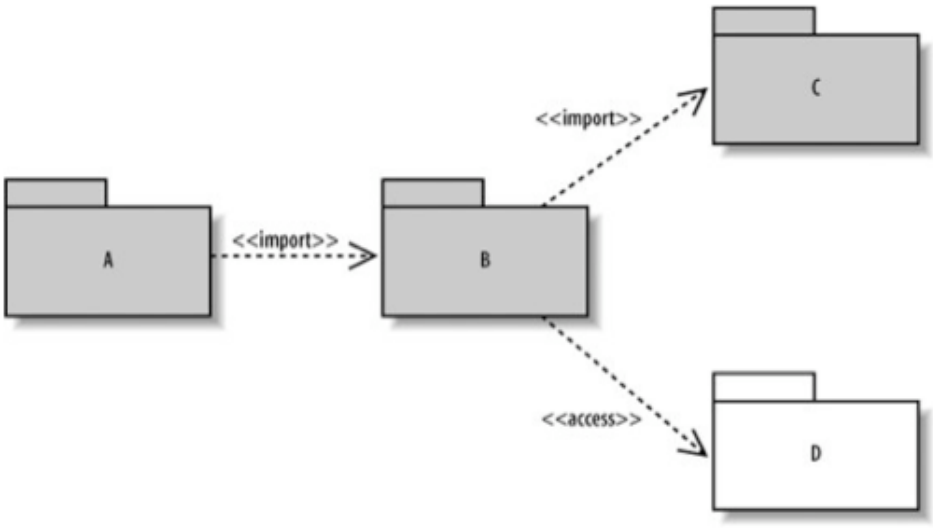
- Utilização de <<*import*>>
  - O package *users* importa todas as definições públicas de *security*, apenas por nome

  - Definições privadas de packages importados não são acessíveis por quem importa.
  - O package *users* apenas importa a classe *Credentials* do package *security*

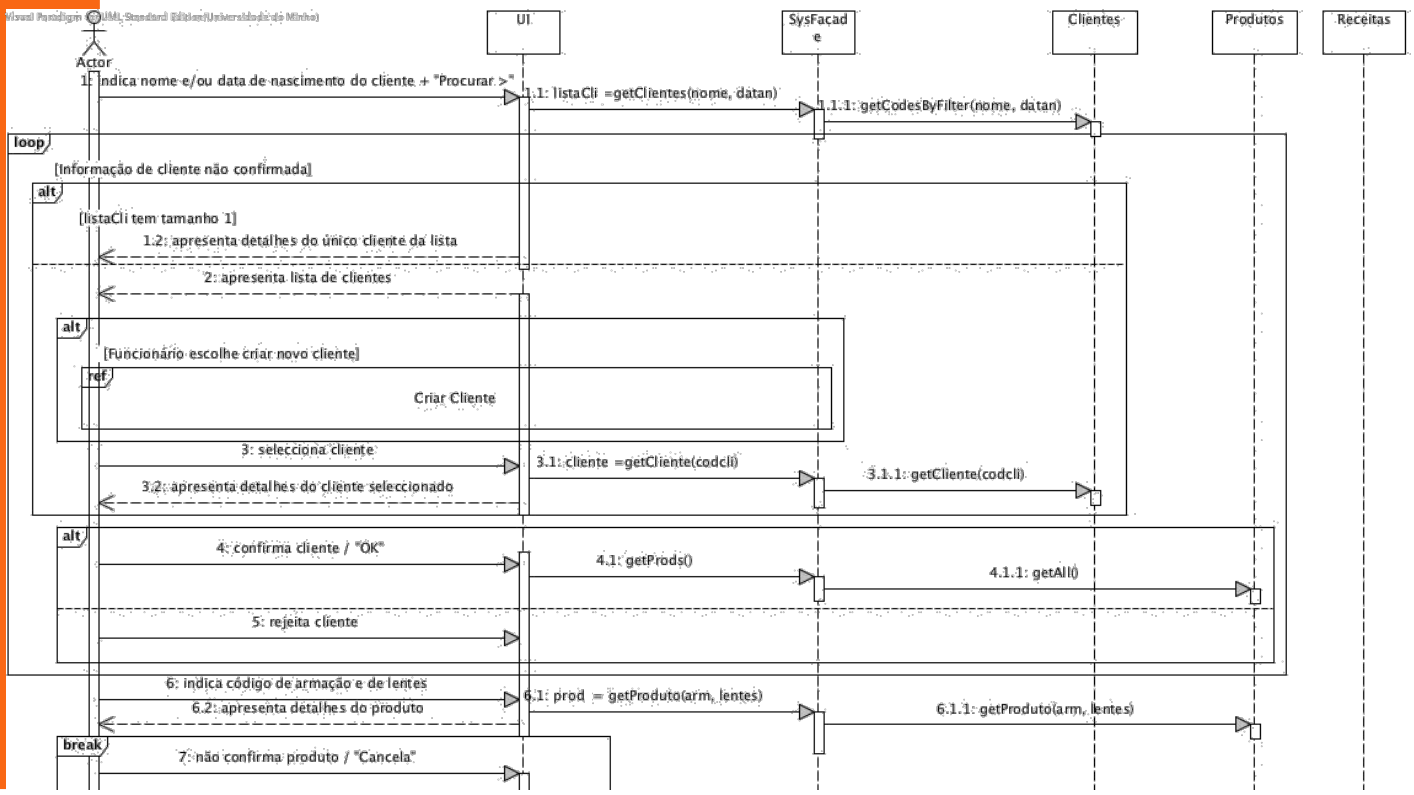


Diagramas de Package (cont.)

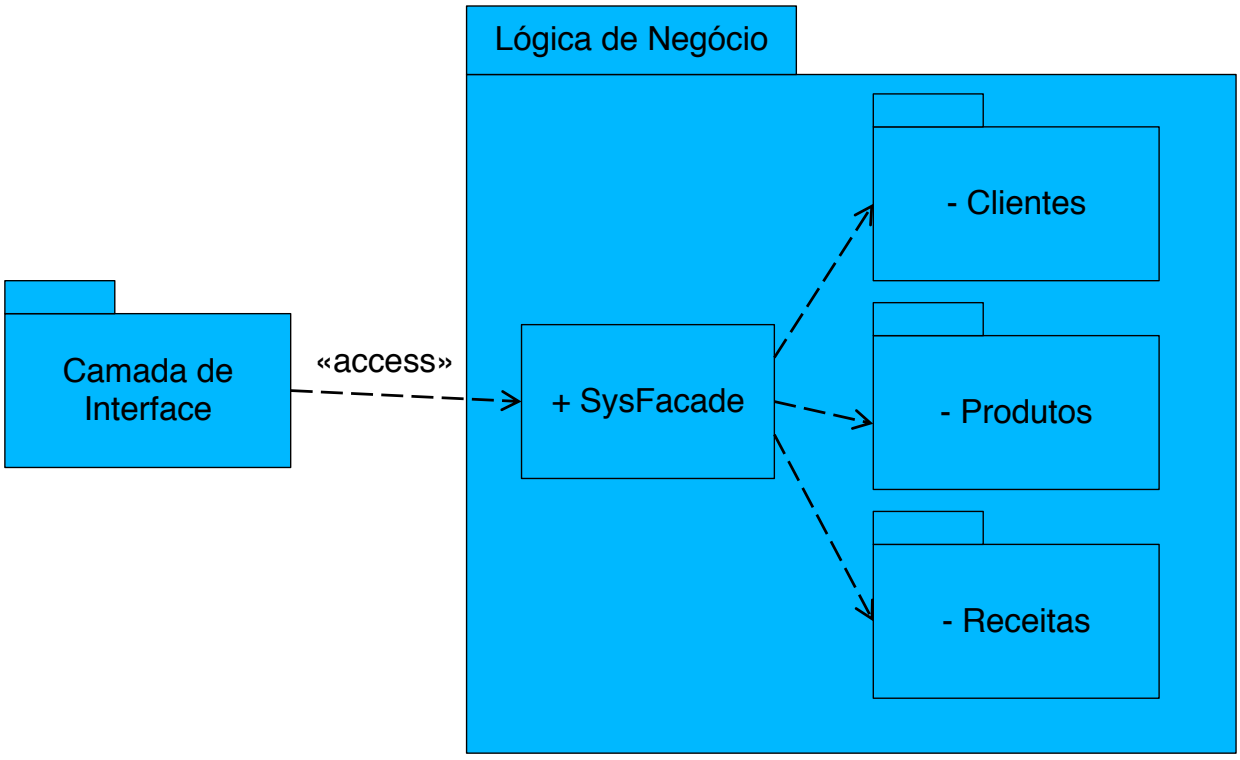
- Utilização de <<*import*>> *e* <<*access*>>
  - O package *B* vê os elementos públicos em *C* e *D*.
  - *A* importa *B*, pelo que vê os elementos públicos em *B* e em *C* (porque este é importado por *B*)
  - *A* não tem acesso a *D* porque *D* é apenas acedido por *B* (não é importado).



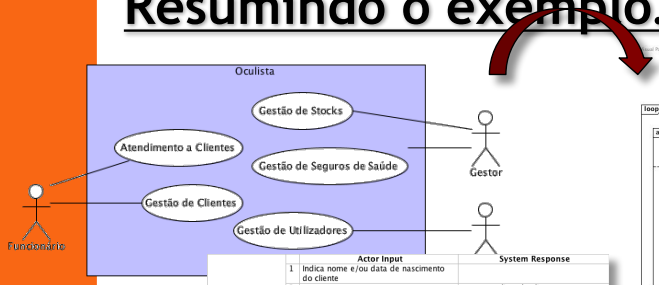
Relembrando o exemplo...



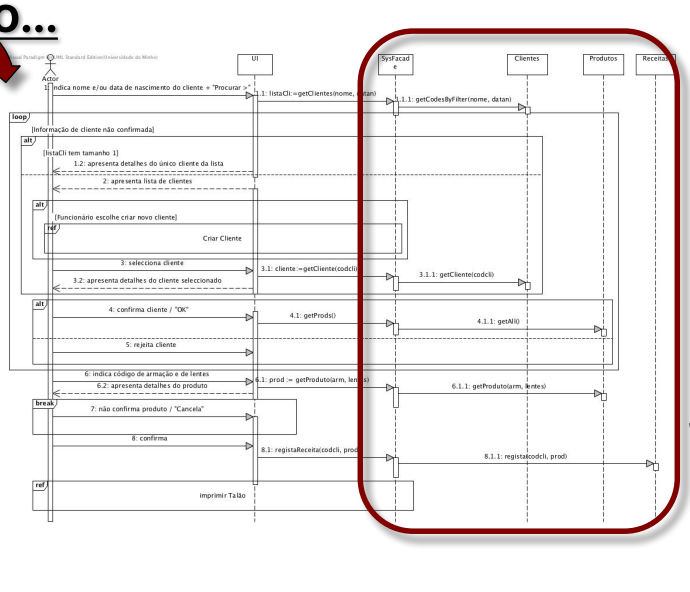
Primeira versão da *arquitectura* da solução...



Resumindo o exemplo...

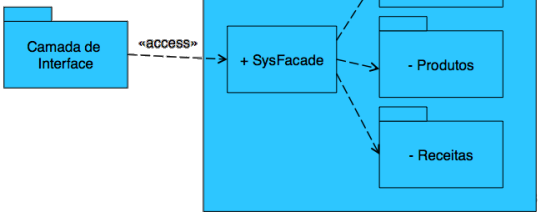


		Actor Input	System Response
Flow of Events	1	Indica nome e/ou data de nascimento do cliente	
	2		apresenta lista de clientes correspondentes
	3	selecciona cliente [ponto de extensão:seleção]	
	4		apresenta detalhes do cliente seleccionado
	5	confirma cliente	
	6	indica código de armação e de lentes	
	7		procura produto e apresenta detalhes
	8	confirma	
	9		regista reserva
	10		«include» Imprimir Talão
Alternative 1 [lista de clientes correspondentes tem tamanho 1] step: 2		Actor Input	System Response
	1		apresenta detalhes do único cliente da lista
	2		regressa a 5
Alternative 2 Step: 5		Actor Input	System Response
	1	não confirma cliente	
	2		regressa a 2
Exception 1 Step: 8		Actor Input	System Response
	1	não confirma produto	
	2		cancela reserva

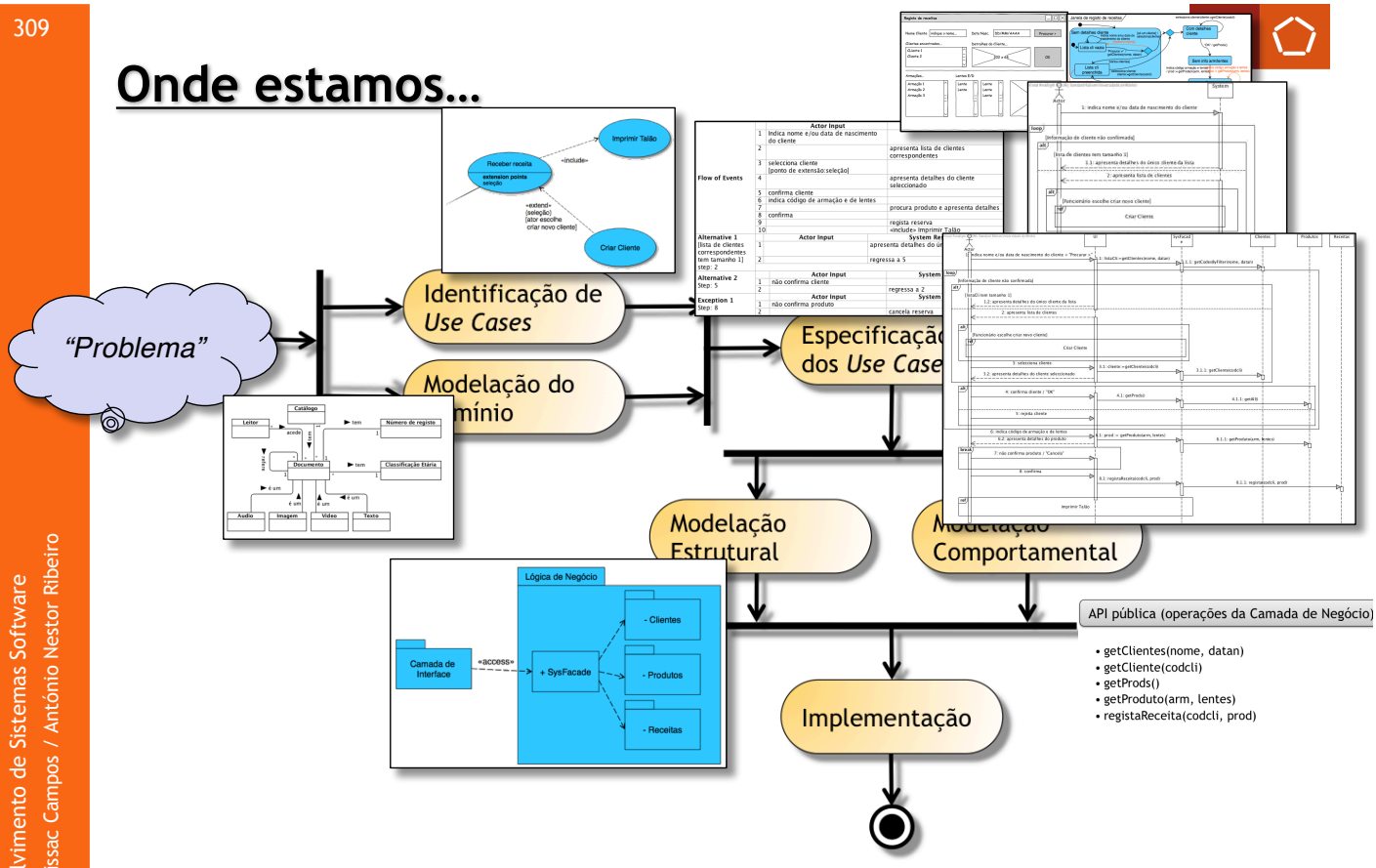


API pública (operações da Camada de Negócio)

- getClientes(nome, datan)
- getCliente(codcli)
- getProds()
- getProduto(arm, lentes)
- registaReceita(codcli, prod)



Onde estamos...



API pública (operações da Camada de Negócio)

- getClientes(nome, datan)
- getCliente(codcli)
- getProds()
- getProduto(arm, lentes)
- registaReceita(codcli, prod)

Modelação Estrutural

Sumário

- Diagramas de *Package*
  - Representação de *Packages*
  - Relações entre packages
    - Composição: diferentes representações gráficas, qualificação, visibilidade
    - Dependências: simples, «import», «access», «merge»
- Ponto da situação relativamente à abordagem ao desenvolvimento seguida