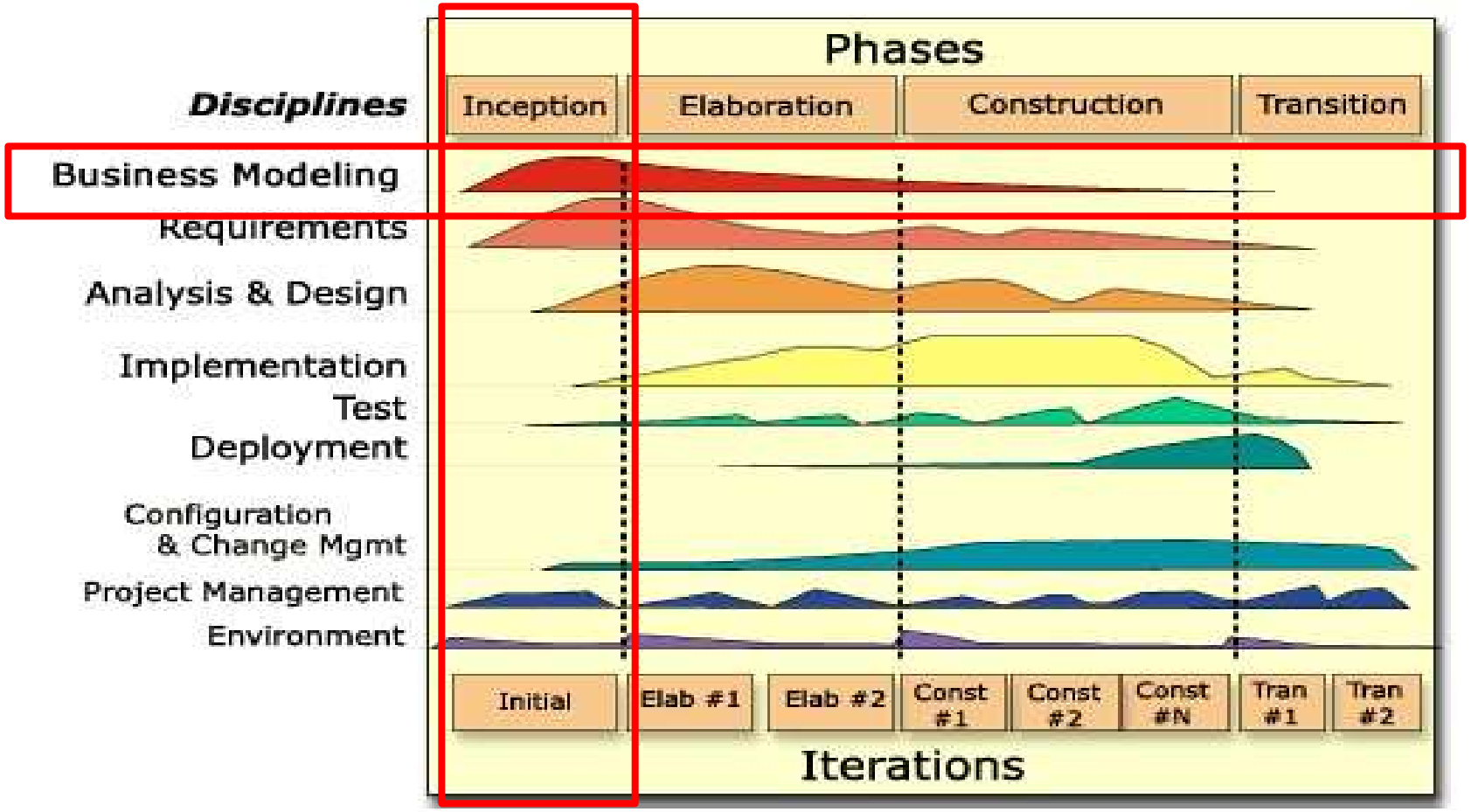
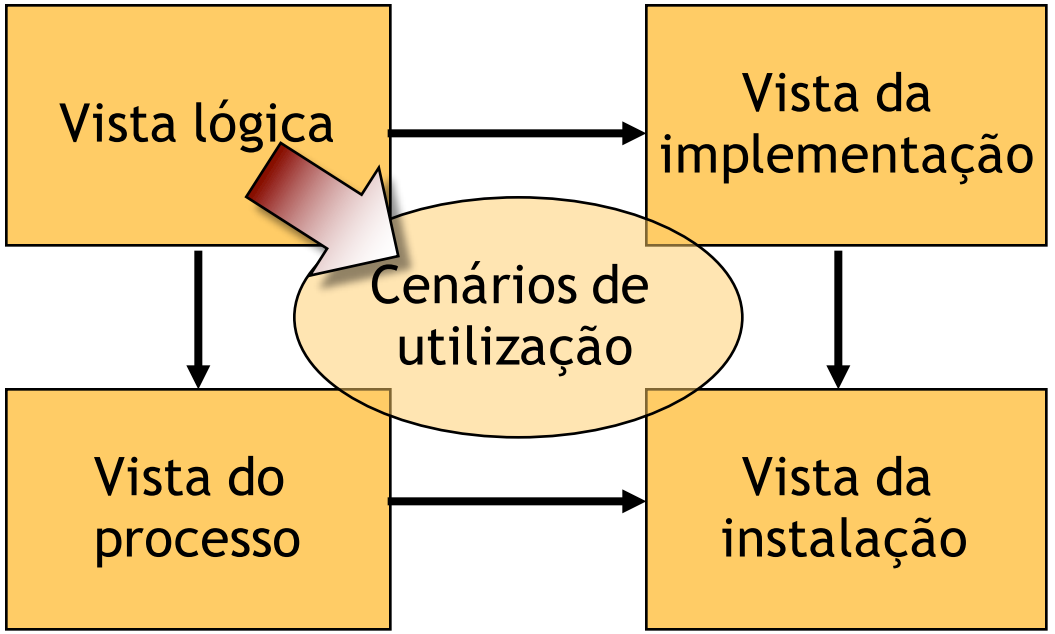
Desenvolvimento de Sistemas Software

Aula Teórica 5: Modelação do Requisitos Funcionais (Diagramas de Use Case)

Próximos passos…

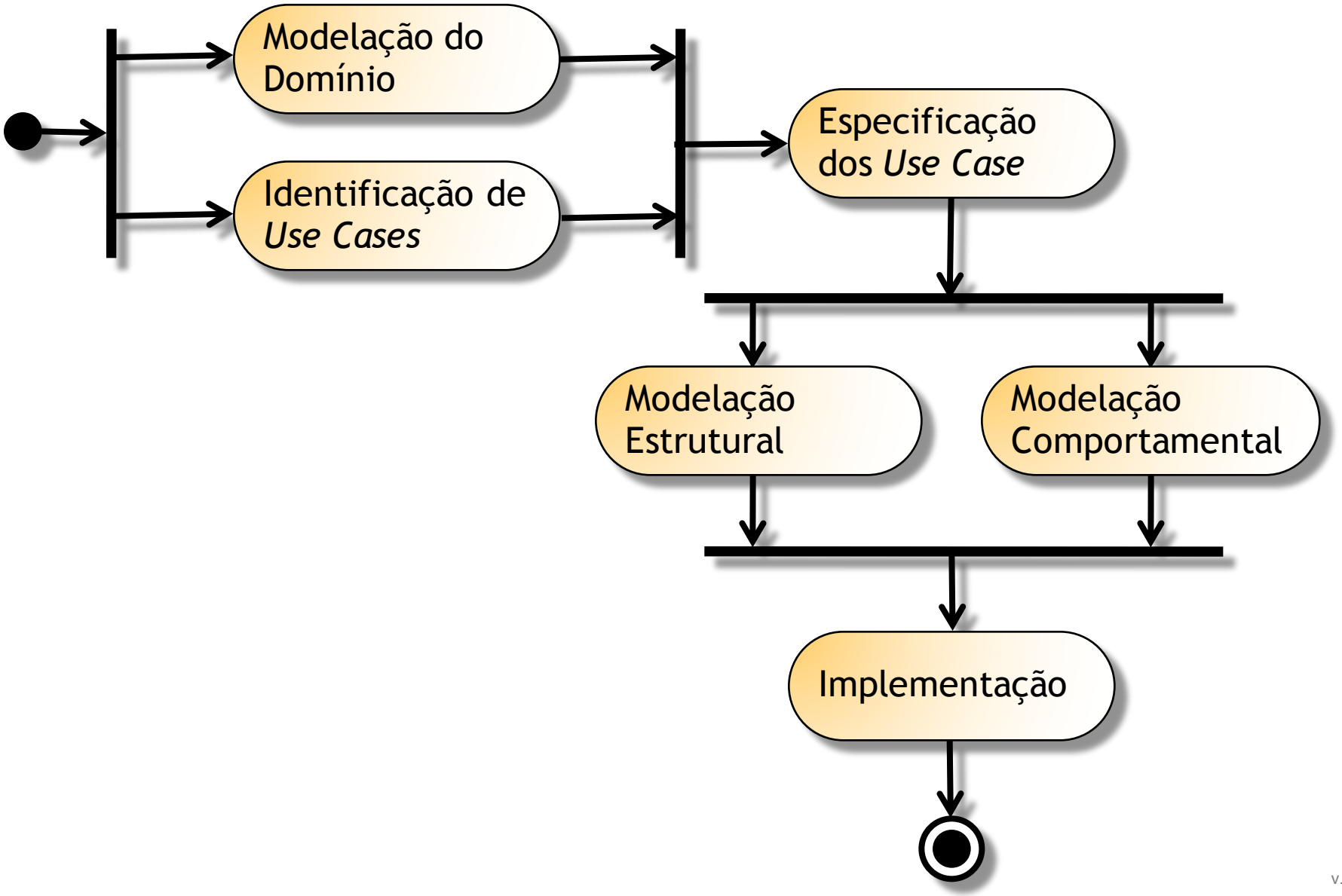


Onde estamos…



(Kruchten, 1995)

Onde estamos…



Definição de requisitos funcionais

Definição de requisitos do sistema, duas abordagens possíveis:

- Visão estrutural – interna
- Visão orientada aos *use case* – externa

Visão Estrutural (OO)

- Definir classes;
- Definir métodos das classes;
- Definir interface com o utilizador (comportamento do sistemas face ao utilizador);

Problemas: O que interessa ao utilizador é o comportamento do sistema, no entanto a interface com o utilizador só é definida no final do processo.

- Perigo de o sistema não fornecer toda a funcionalidade pretendida;
- Perigo de o sistema fornecer funcionalidade não pretendida

  (= despedício de trabalho).

Definição de requisitos funcionais

Visão orientada aos *Use Case*

- Identificar actores – quem vai interagir com o sistema?
- Identificar *Use Case* – o que se pretende do sistema?
- Identificar classes de suporte à realização dos *use case* – como vai a funcionalidade necessária ser implementada?

Vantagens:

- Não há trabalho desnecessário.
- O Sistema de Informação suporta as tarefas do cliente.
- As fronteiras do Sistema ficam bem definidas.

Definição de *Use Case*

- Uma unidade coerente de funcionalidade – um serviço
- Define um comportamento do sistema sem revelar a estrutura interna
  - mostra a comunicação entre sistema e actores
- O conjunto de todos os use case define a funcionalidade do sistema
  - resultam do diálogo com o cliente
  - definem as responsabilidades funcionais do sistema

Identificação de *Use Cases*

- Podemos identificar os *Use Case* do sistema a partir da identificação de cenários de utilização.
- Um cenário descreve um contexto concreto de interacção entre o utilizador e o sistema. Por Exemplo:

  Durante o semestre o Prof. Faísca foi enviando os sumários com breves resumos da matéria leccionada, via email, para o sistema Fly2. Após o fim das aulas, o Prof. Faísca utilizou a interface web do sistema para actualizar cada um dos sumários com descrições mais completas das matérias leccionadas. Finda essa actualização, imprimiu os sumários e enviou-os à Secretaria.

- A partir dos cenários podemos identificar o(s) Actor(es) e os Use Cases (serviços) necessários à correcta disponibilização da funcionalidade requerida pelo mesmo.

Identificação de *Use Cases*

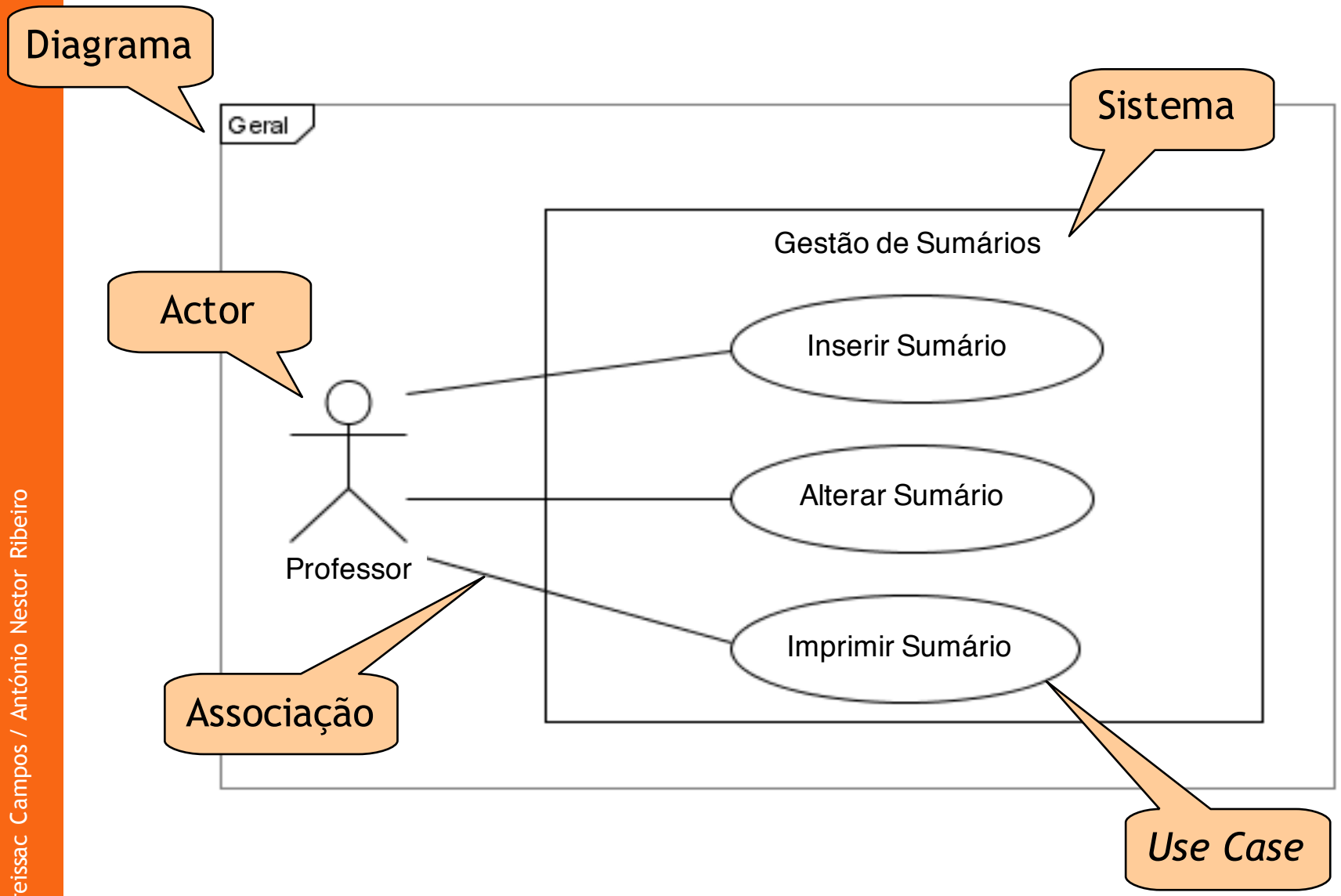
No cenário anterior podemos identificar os seguintes *Use Case:*

> Durante o semestre o Prof. Faísca **foi enviando os sumários** com breves resumos da matéria leccionada, **via email**, para o sistema Fly2. Após o fim das aulas, o Prof. Faísca **utilizou a interface web** do sistema **para actualizar cada um dos sumários** com descrições mais completas das matérias leccionadas. Finda essa actualização, **imprimiu os sumários** e enviou-os à Secretaria.

1. Inserir sumários via email
2. Actualizar sumários via web
3. Imprimir sumários (via web? / via e-mail?)
4. Enviar sumários à secretaria – deverá este *use case* ser considerado?

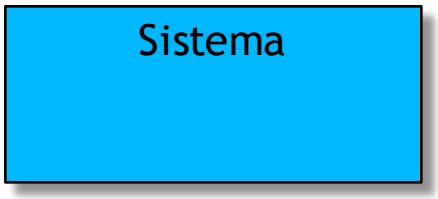
   No cenário descrito o envio é feito em papel. Não se trata, portanto, de um serviço fornecido pelo sistema. No entanto, podemos discutir a possibilidade de o envio passar a ser feito electronicamente – estariamos a alterar o modo de trabalho inicialmente previsto/actual!

Diagrama de *Use Cases*



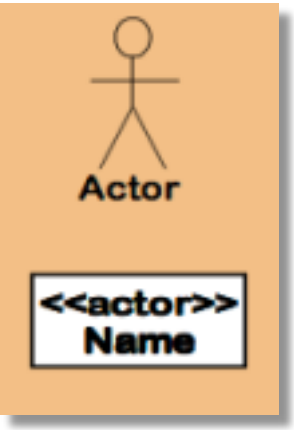
Sistema

- define as fronteiras da solução a desenvolver



Actor

- uma abstracção para uma entidade fora do sistema
- um actor modela um propósito (alguém que tem um interesse específico no sistema) – pode não mapear 1 para 1 com entidades no mundo real
- um actor não é necessariamente um humano – pode ser um computador, outro sistema, etc.
- cada actor representa um papel ("role") que "alguém" ou qualquer "coisa" externa ao sistema pode assumir



- o conjunto de todos os actores definem todas as formas de interacção com o sistema

Associação

- representa comunicação entre o actor e o sistema – através de *use cases*
- pode ser bi-direccional ou uni-direccional

Identificação *de Use Cases*

Etapas a cumprir (com o auxílio de cenários de utilização do sistema):

1. Identificar actores (quem utiliza o sistema)
2. Identificar *use cases* (o que se pode fazer no sistema)
3. Identificar associações (quem pode fazer o quê)

Identificar actores

- Quem vai utilizar o sistema?
- Neste caso: Docente, Secretaria?, Servidor Email?, WebApp?

Identificar use cases

- Objectivos dos utilizadores/actores?
- Resposta a estimulos externos.

Que Actores? Que Associações?

Identificar associações

- Que actores utilizam que *use cases*?
- Nem sempre é imediatamente evidente se a comunicação entre o sistema em análise e sistemas externos deve ser representada. Quatro abordagens podem ser identificadas:
  - ✗ mostrar todas as associações;
  - ✗ mostrar apenas as associações relativas a interacção iniciada por sistemas externos;
  - ✔ mostrar apenas as associações relativas a interacções em que é o sistema externo o interessado no *use case*;
  - ✗ não mostrar associações com sistemas externos.

Que Actores? Que Associações?

Todas as associações

- Todos os sistemas externos que interagem com o sistema em análise são apresentados como actores e todas as interacções são representadas nos diagramas.
- Demasiado abrangente, em muitos casos existem interacções com outros sistemas apenas por razões de implementação e não por se tratarem de requisitos do sistema.

Apenas as associações relativas a interacção iniciada por sistemas externos

- Só são representados como actores os sistemas externos que iniciem diálogo com o sistema em análise.
- Mesmo assim muito abrangente.

Que Actores? Que Associações?

Apenas as associações em que é o sistema externo o interessado

- Neste caso só são apresentados como actores os sistemas externos que necessitam de funcionalidade fornecida pelo sistema em análise.
- Usalmente esta é uma solução equilibrada.

Não mostrar associações com sistemas externos

- Apenas os utilizadores são actores, neste caso quando existem sistemas externos apresentam-se os seus actores em diálogo directo com o sistema a ser modelado.
- De uma outra forma esta solução também é demasiado abrangente e pode levar a confusão sobre quem está realmente a utilizar o sistema.

Diagramas de *Use Cases* – *revisão de conceitos*

- Modelam o contexto geral do sistema. Quais os actores que com ele se relacionam e que use case deve suportar.
- A concepção do sistema é guiada pelo modelo de *use case*:
  - Utilizam-se *use cases* para capturar os requisitos funcionais do sistema de uma forma sistemática;
  - O modelo de *use case* captura toda a funcionalidade requerida pelos utilizadores;
- A implementação do sistema é guiada pelo modelo de *use case*:
  - cada *use case* é implementado sucessivamente:
  - quando todos os *use cases* estiverem implementados obtém-se o sistema final;
  - fica facilitada a manutenção do sistema sempre que os requisitos sejam alterados;
- O modelo de *use case* é utilizado para o planeamento de testes:
  - Após a definição do modelo de *use case*: planear *black-box testing*.
  - Após a implementação dos *use cases*: planear *white-box testing*.

Black-box testing

- Utilizado para verificar se o sistema implementa toda a funcionalidade pretendida.
- Permite detectar erros de "omissão" (funcionalidade não implementada).

White-box testing

- Utilizado para verificar se o sistema implementa a funcionalidade de forma correcta.
- Permite detectar erros na implementação da funcionalidade pretendida.

Diagramas de *Use Case* - Resumo

- Os diagramas de *Use Case* permitem definir os requisitos funcionais de um sistema:
  - que serviços deve fornecer;
  - a quem os deve fornecer.
- Notação diagramática facilita o diálogo (com os clientes e dentro da quipa de desenvolvimento).
- Utilizando diagramas de *use case*, clientes e equipa de desenvolvimento podem chegar a um acordo sobre qual o sistema a desenvolver.
- A resolução de alterações nos requisitos funcionais fica facilitada.

No entanto:

- Os diagramas de *use case* não suportam a captura de requisitos não funcionais.

Quando utilizar diagramas de Use Case?

- Sempre que se estiverem a analisar requisitos (funcionais)!

Definição de *Use Case* – *Especificação*

- A UML não especifica como descrever *Use Cases*
  - *Tem que ser definido por cada organização ou projecto*
- *Muitas abordagens são possíveis/comuns*
  - *Desde descrições textuais até especificações via diagramas*
  - *Mais ou menos verbosas e detalhadas*
- *Em DSS vamos (começar por) utilizar uma notação tabular*

Use Case: Fazer alguma coisa		
Pré-condição: Sistema preparado		
Pós-condição: Objectivo atingido!		
	Actor	**Sistema**
Comportamento Normal	Faz algo	
		Responde

Use Cases - Especificação Tabular

Use Case: Fazer Telefonema		
Pré-condição: Telefone ligado e em descanso		
Pós-condição: Telefone ligado e em descanso após conversa		
	Actor	**Sistema**
Comportamento Normal	1. Digita número e inicia chamada	
		2. Emite sinal de chamada
	3. Aguarda	
		4. Estabelece ligação
	5. Conversa	
	6. Termina chamada	
		7. Desliga chamada
Comportamento Alternativo (passo 6)		6.1. Termina chamada
		6.2. Regressa a 7
Excepção (passo 2) [nº inválido]		2.1. Transmite informação de número inválido
		2.2 Desliga ligação

Use Cases - Especificação

- Não escrever *Use Cases* demasiados longos
  - Idealmente não mais de 10 passos
- Entidades referidas no Use Case devem estar presentes no Modelo de Domínio
  - Modelo de Domínio descreve o contexto do problema
  - Modelo de Use Case descreve uma solução
  - Conceitos têm que ser os mesmos!
- Deve ser expresso ao nível dos requisitos dos Actores (utilizadores/sistemas)
  - Não devem especificar a interface com o utilizador!!

Definição de *Use Case* – *Especificação*

- Descreve como os Actores atingem objectivos (realizam os *Use Cases*) utilizando o sistema
  - Definem relação entre *inputs* dos Actores e comportamento do Sistema
- Especificação deve incluir o comportamento tipicamente esperado, bem como variações
  - Comportamentos alternativos que ainda levam ao sucesso
  - Comportamentos de insucesso (Excepções)

  - vamos também definir as pré-condições e pós-condições de cada use case (cf. *design by contract*).

Design by contract

- *Design by contract* (DBC) baseia-se na noção de um contrato entre um cliente e um fornecedor para a realização de um serviço.
- O conceito central do DBC é a asserção (uma asserção é uma expressão booleana que nunca deverá ser falsa).
- Tipicamente as asserções são automaticamente testadas durante a fase de *debug*.
- O DBC identifica três tipos de asserções:
  - pré-condições – condições que se devem verificar para a invocação de um dado serviço ser válida;
  - pós-condições – condições que se devem verificar após a execução de um serviço;
  - invariantes – asserções que se devem verificar durante o tempo de vida da entidade a que se aplicam.
- A partir da versão 1.4 o Java passou a ter *asserts* que podem ser utilizados para definir pré- e pós-condições – no entanto não suporta invariantes .

Modelação do Requisitos Funcionais

Sumário:

- Requisitos funcionais vs. requisitos não funcionais
- Definição de requisitos funcionais
- Diagramas de Use Case: notação básica
- Definição de Use Case
- Representação textual de Use Cases
- Identificação de Use Cases e Actores