



# CONSIDERAÇÕES INICIAIS SOBRE DESENVOLVIMENTO DE SOFTWARE

1

# METÁFORA



How the customer explained it



How the Project Leader understood it



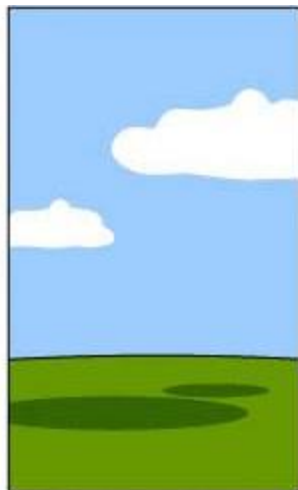
How the Analyst designed it



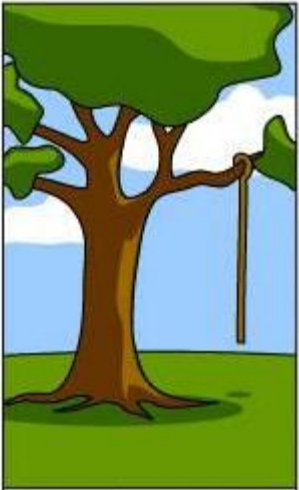
How the Programmer wrote it



How the Business Consultant described it



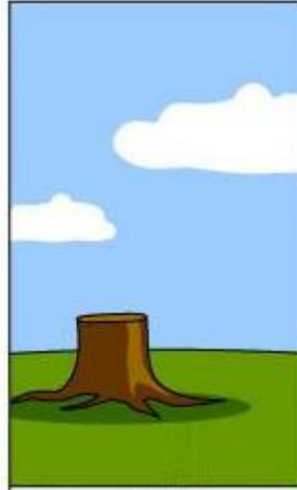
How the project was documented



What operations installed



How the customer was billed



How it was supported

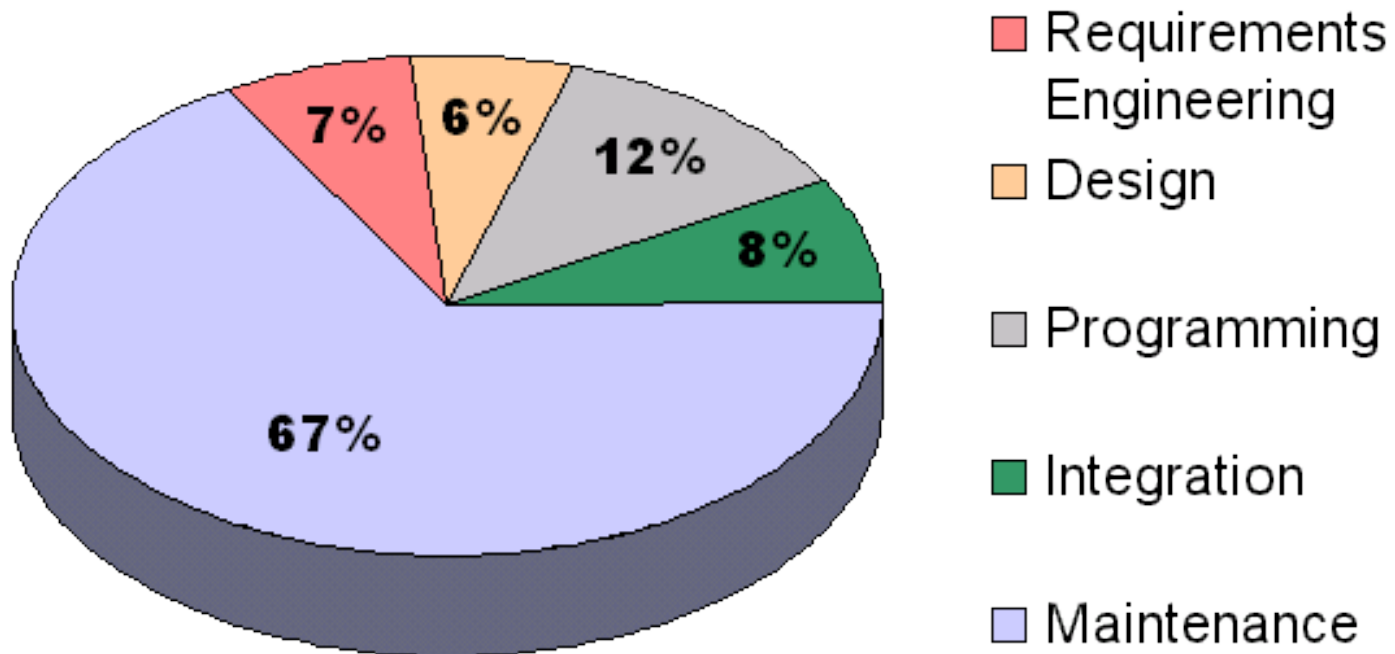


What the customer really needed

# PROBLEMAS INERENTES AO DESENVOLVIMENTO DE SOFTWARE

- Software é intangível
- Software é invulgarmente flexível
- Engenharia de software não é reconhecida como uma disciplina com o mesmo status formal de outras (e.g. mecânica, electrotécnica, civil)
- Desenvolvimento de software não é (e)standardizado
- Necessidade de comunicação com cliente e equipa
- Dinâmica da equipa de desenvolvimento
- Identificar problema, conhecer o negócio/domínio e discutir e descrever possíveis arquiteturas, etc.

# CUSTOS RELATIVOS POR FASES DO CICLO DE VIDA DE SOFTWARE

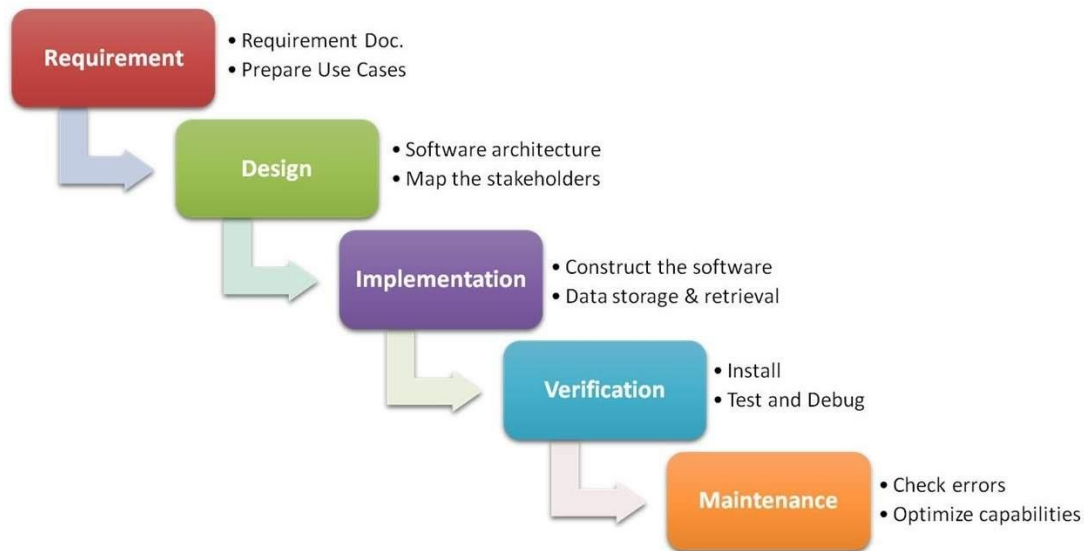


## E CONSIDERANDO ESTAS DIFICULDADES...

- Sistemas de software são cada vez mais importantes no nosso dia-a-dia
- Falhas no desenvolvimento de software podem custar muito mais que o custo do software e o tempo aplicado
- Pelo que se torna fundamental e urgente uma engenharia de software efetiva
  - Introduzindo a noção de método/modelo/processo
  - Que imponha um processo disciplinado sobre desenvolvimento de software
  - Mais efetivo e eficiente

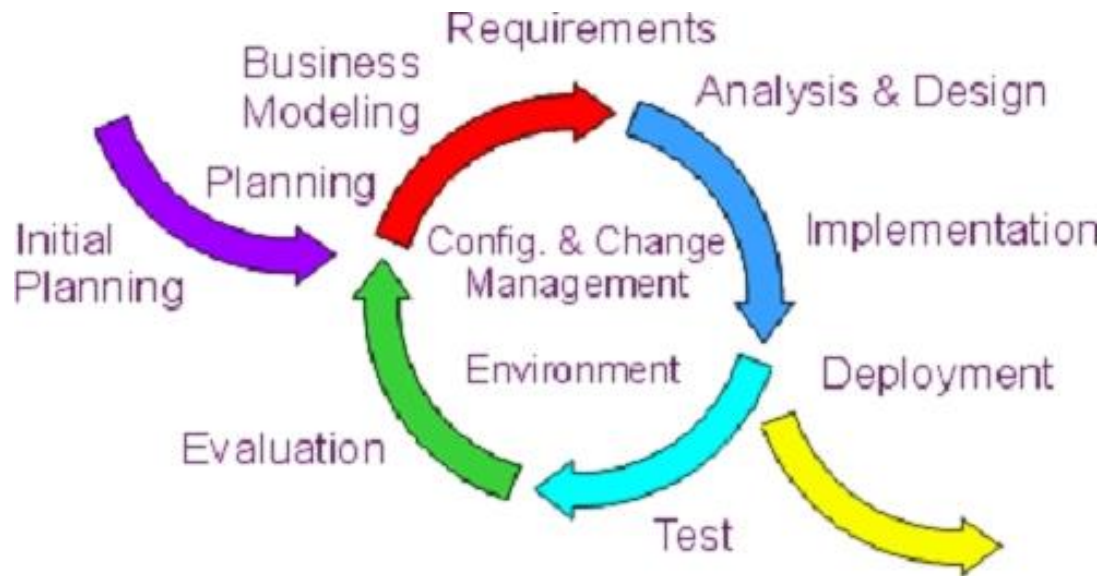
# PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- É necessário uma metodologia para documentar e desenvolver software.



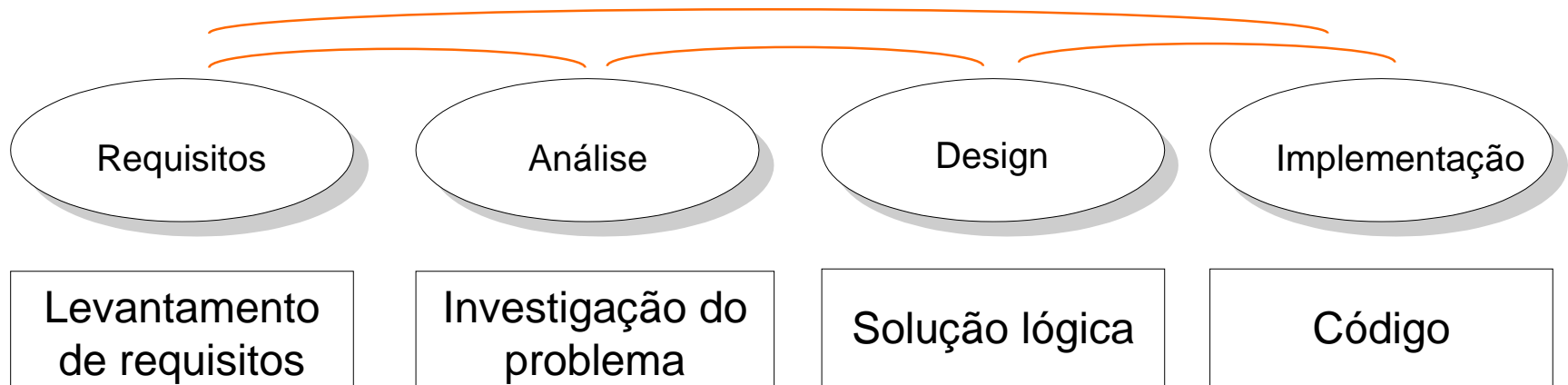
(Exemplo: Metodologia sequencial *Waterfall*)

# PROCESSO DE DESENVOLVIMENTO DE SOFTWARE



(Exemplo: Metodologia Iterativa Rational Unified Process Framework (RUP))

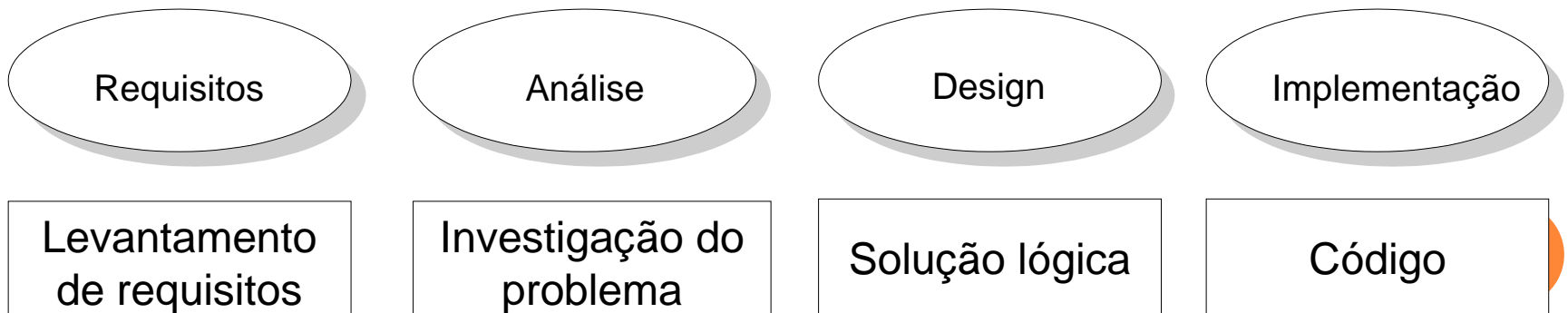
# DOS REQUISITOS ATÉ AO CÓDIGO





# ENGENHARIA DE SOFTWARE NÃO É SÓ PROGRAMAÇÃO/CODIFICAÇÃO

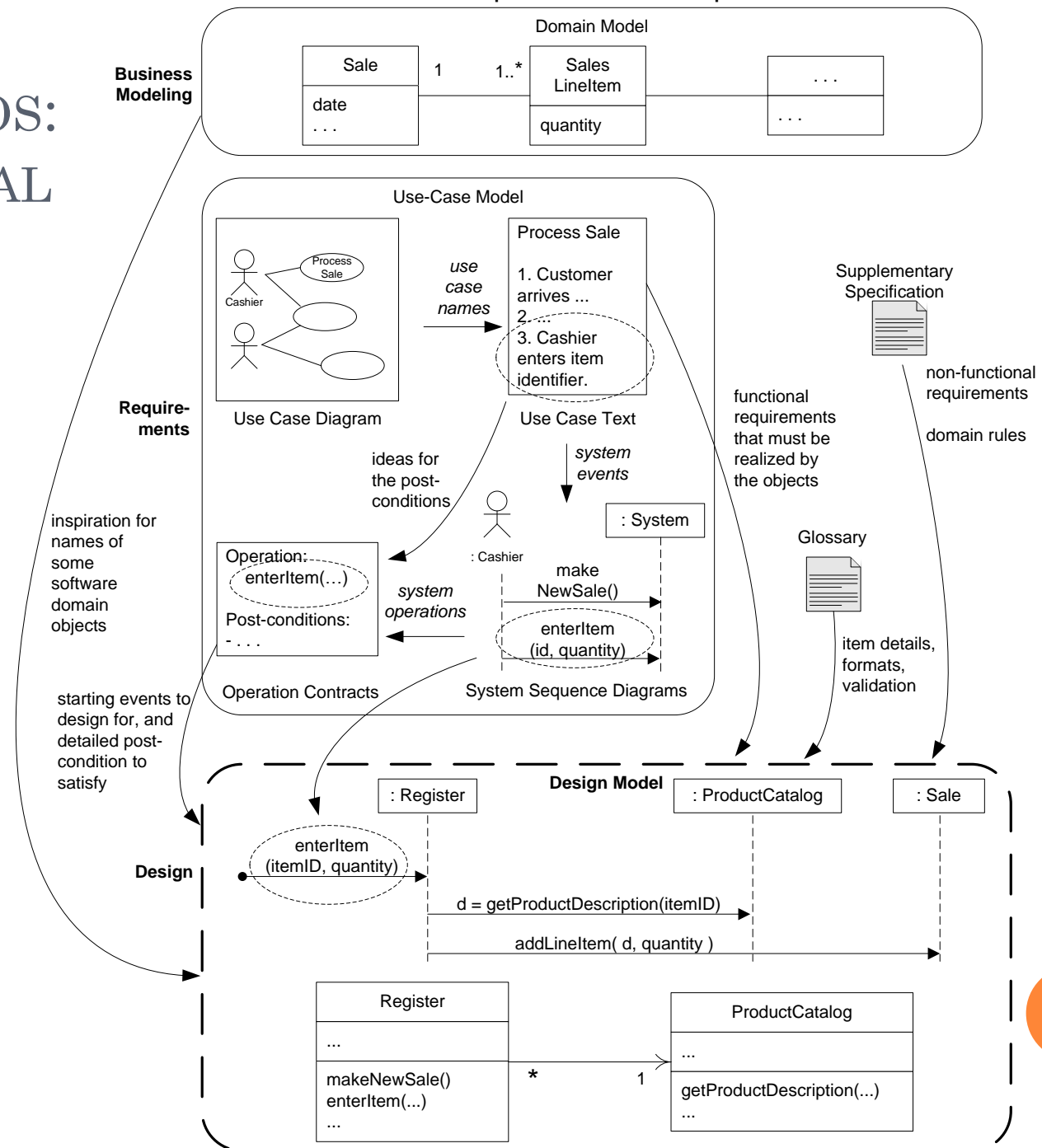
- Saber uma linguagem de programação é necessário mas não é suficiente para se criar (bom) software
- Entre uma boa ideia e bom software existe muito mais que programação, nomeadamente



# REQUISITOS VS. ANÁLISE VS. DESIGN

- Requisitos
  - Identificar o que é que as partes interessadas (“stakeholders”) querem
  - Clarificar e registar as restrições e requisitos
- Análise
  - Fazer a coisa correcta (“do the right thing”)
  - Investigação do problema e de requisitos
  - Explorar os detalhes dos requisitos
  - e.g. análise de requisitos ou análise OO
- Design
  - Fazer correctamente a coisa (“do the thing right”)
  - Solução conceptual que visa preencher os requisitos
  - Pode conduzir a implementação
  - Mas não é implementação
  - e.g. design arquitetura, design de casos de uso OO, design de base de dados
- Implementação
  - Construir a coisa. Código máquina (Java, C, C#,... )!
- e ainda...

# ARTEFACTOS: VISÃO GERAL





# ENGENHARIA DE REQUISITOS

12

# ARTEFACTOS DE REQUISITOS

- Artefactos habituais:
    - Modelo de Casos de Uso
    - Visão
    - Especificação Suplementar
    - Glossário
    - Contratos de operação do sistema
    - ...
  - Como os desenvolver:
    - Sinergia: cada um ajuda a clarificar o outro
- 
1. Escrever um breve esboço da visão
  2. Identificar os casos de uso e dos objectivos dos utilizadores
  3. Escrever alguns dos casos de uso
  4. Iniciar a especificação suplementar
  5. Refinar a Visão, sumariando e sistematizando a informação dos anteriores

# GLOSSÁRIO

- O Glossário é como um dicionário; define terminologia usada por outros artefactos
- Sugestão: iniciar cedo o glossário!
- Exemplo:

Termo/Expressão	Significado/Descrição
Caixa	Funcionário que opera o POS durante a venda
POS	Point-of-sale. Espécie de caixa registadora com funcionalidades extra.
Venda	Processo de negócio no qual se transaccionam bens com o cliente, que paga pelos bens.
...	...

# MODELO DE CASOS DE USO

- **Caso de Uso** (“Use Case”) é uma história de como se usa o sistema para atingir determinado objectivo
- **Actor** é algo com comportamento, tal como uma pessoa, computador ou organização (e.g. funcionário de caixa)
- **Cenário** – também chamado instância de caso de uso – é uma sequência específica de acções e interacções entre actores e o sistema. É um caminho seguido num caso de uso.
  - Cada caso de uso é uma colecção de cenários de sucesso e insucesso relacionados
- **Modelo de Casos de Uso** (“Use Case Model”) é o conjunto de todos os Casos de Uso desenvolvidos no âmbito da disciplina de Requisitos
  - Promove a compreensão e a descrição de requisitos (especialmente os funcionais)

# REGRAS PARA ESCREVER CASOS DE USO

- Tente-se responder à questão: Como é que o uso do sistema suporta de forma visível os utilizadores para atingirem os seus objectivos?
- Os casos de uso enfatizam os requisitos funcionais
- Casos de uso são documentos de texto, não diagramas
- Casos de uso baseados em caixas pretas são recomendáveis: descrevem “o quê” (responsabilidades) e não “o como” (procedimentos)
  - Exemplo: o sistema regista a venda...
  - Contra-exemplo: o sistema executa o seguinte comando SQL para registar a venda
  - Exemplo: o utilizador confirma
  - Contraexemplo: o utilizador carrega no botão de “ok”



# FORMATOS DE CASOS DE USO

## ○ Breve (“Brief”); de alto nível:

- um parágrafo descrevendo o cenário de sucesso principal
- (no início do projeto escrever a maior parte dos UCs nesta forma)

### **Processar Venda:**

Um cliente chega à caixa com as compras para pagar. O caixa usa o POS para registar cada item. O sistema apresenta o sub-total e os detalhes de cada item introduzido.

O cliente introduz informação de pagamento, que é validada e registada pelo sistema. O sistema actualiza os stocks. O cliente recebe o talão de compra do sistema e sai com os item comprados.

## ○ Casual:

- múltiplos parágrafos sobre vários cenários
- (na fase de Início, escrever alguns nesta forma)

### **Processar Venda:**

Cenário de sucesso principal: o cliente chega à caixa com as compras para pagar...

Cenários alternativos:

- se o sistema rejeitar o cartão de crédito...
- se o cliente não tiver saldo suficiente...

# FORMATOS DE CASOS DE USO (CONTINUAÇÃO)

- Completo (“Fully-dressed”):
  - contém as secções:
    - Actor primário
    - Partes interessadas (stakeholders) e seus interesses
    - Pré-condições
    - Pós-condições
    - Cenário de sucesso principal (ou fluxo básico)
    - Extensões (ou fluxos alternativos)
    - Requisitos especiais
    - Tecnologia e Lista de Variações dos Dados
    - Frequência de Ocorrência
    - Questões em aberto
  - (na fase de Elaboração, escrever os casos de uso neste formato)

# CASO DE USO COMPLETO: EXEMPLO

- Designação do Caso de Uso: **Processar venda**
- Actor principal
  - O funcionário de caixa (“o caixa”)
- Partes interessadas e seus interesses:
  - O caixa: introdução rápida de informação, pagamento sem erros
  - Clientes: serviço rápido e esforço mínimo
  - Empresa: ?
  - Organismos de cobrança de impostos: ?
  - Serviços de autorização de pagamento: ?
- Pré-condições:
  - O caixa está identificado e autenticado
- Pós-condições:
  - A contabilidade e stocks correctamente actualizados.
  - O talão é emitido

# CASO DE USO COMPLETO: EXEMPLO (CONTINUAÇÃO)

- Cenário de sucesso principal (ou fluxo básico):

1. O cliente chega à caixa POS com os produtos para comprar
2. O caixa inicia a nova venda
3. O caixa introduz o identificador do item
4. O sistema regista o item e apresenta a sua descrição, preço e subtotal

O caixa repete passos 3-4 até todos os itens estarem introduzidos

5. O sistema apresenta o total com os impostos calculados
6. O caixa informa o cliente do total a pagar e pede o pagamento
7. O cliente paga e o sistema processa o pagamento
8. O sistema apresenta o talão
9. O cliente sai da loja com o talão e os produtos

# CASO DE USO COMPLETO: EXEMPLO (CONTINUAÇÃO)

- Extensões (ou fluxos alternativos):
  - \*a. a qualquer momento o sistema falha: garantir que o estado de toda a transacção é recuperável
    1. O caixa reinicia o sistema, identifica-se, autentica-se e pede recuperação do estado anterior
    2. O sistema reconstrói o estado anterior
  - 3a. Identificador inválido:
    1. O sistema assinala o erro e rejeita o objecto
  - 3-6a. O cliente pede para cancelar a venda:
    1. O caixa cancela a venda no sistema

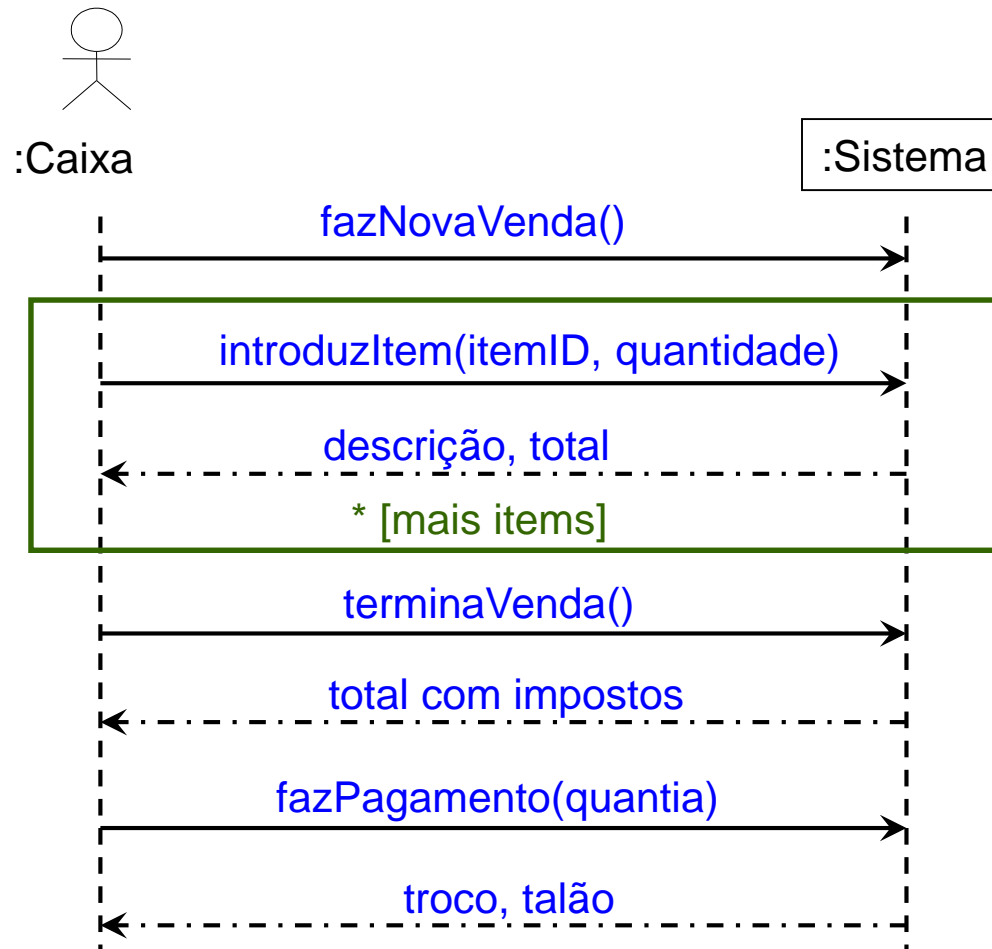
# CASO DE USO COMPLETO: EXEMPLO (CONTINUAÇÃO)

- Requisitos especiais:
  - GUI em ecrã tátil. Texto visível a 1 metro de distância
  - Autorização de crédito em 30" em média, em 90% dos casos
- Tecnologia e Lista de Variações de Dados
  - 3a. ID do item introduzido por leitor de código de barras ou teclado
  - 3b. ID do Item de acordo com esquema UPC, EAN, JAN ou SKU
  - etc.
- Frequência de ocorrência
  - Pode ser quase contínuo
- Questões em aberto:
  - Quais são as variações nas taxas e nos impostos?
  - Recuperação dos serviços remotos

# DIAGRAMA DE SEQUÊNCIA DE SISTEMA (SSD: “SYSTEM SEQUENCE DIAGRAMS”)

- Os casos de uso descrevem como os actores interagem com o sistema de software
- SSD são visualizações das interacções descritas nos casos de uso
- São a notação UML para ilustrar as interacções do actor
- SSD são parte do modelo de casos de uso
- Dado um cenário dum caso de uso, um SSD ilustra:
  - Os actores externos que interagem directamente com o sistema
  - O sistema como uma caixa negra
  - Os eventos do sistema que o actor gera
  - A ordem dos eventos segue a ordem no caso de uso

# EXEMPLO DE SSD



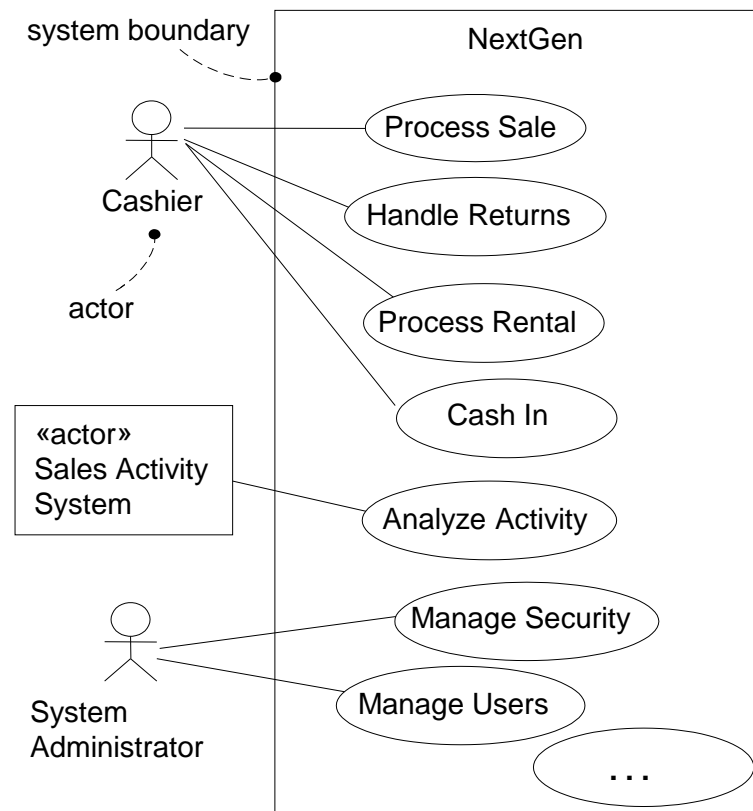


# GRANULARIDADE DOS CASOS DE USO

- Qual é o nível correcto de expressar casos de uso?
- Que casos de uso devem ser seleccionados, nomeadamente no que respeita à granularidade:
  - Negociar um contracto com o fornecedor?
  - Processar devoluções?
  - Login?
- Devem focar-se no processo de negócio elementar (“elementary business processes” EBP). Um EBP é uma tarefa:
  - realizada por uma pessoa
  - num local específico
  - num determinado momento
  - em resposta a um evento de negócio
  - que adiciona valor de negócio mensurável
  - deixa os dados num estado consistente
- Casos de uso de baixo nível são úteis quando correspondem a subtarefas repetidas por múltiplos casos de uso

# DIAGRAMAS DE CASOS DE USO

- Serve para fornecer uma perspectiva visual dos casos de uso
- Não substitui de todo o documento de texto



# ESPECIFICAÇÃO SUPLEMENTAR (ES)

- "Supplementary Specification" (SS)
- Capta os requisitos não capturados no Modelo de Casos de Uso
- Modelo FURPS+:
  - Funcionalidade:
    - Capacidades, auditoria, reporte, interoperabilidade, compatibilidade, segurança
  - Usabilidade:
    - Factores humanos de utilização do sistema, documentação, estética, ...
  - Desempenho:
    - tempo de resposta, consumo de memória/CPU
  - Fiabilidade/Confiabilidade:
    - frequência e gravidade admissível de erros, previsão, recuperação, exatidão

# ESPECIFICAÇÃO SUPLEMENTAR (ES)

- Suportabilidade:
  - testabilidade, adaptabilidade, manutenibilidade, configurabilidade, escalabilidade, ...
- Restrições de design
  - Padrões de design, PDS, ferramentas, bibliotecas
- Restrições de implementação
  - SO, linguagens de programação, SGBD,
- Restrições de interface (com outros sistemas)
  - API disponibilizadas/requeridas por sistemas terceiros
- Restrições físicas
  - hardware



31

## EXEMPLO

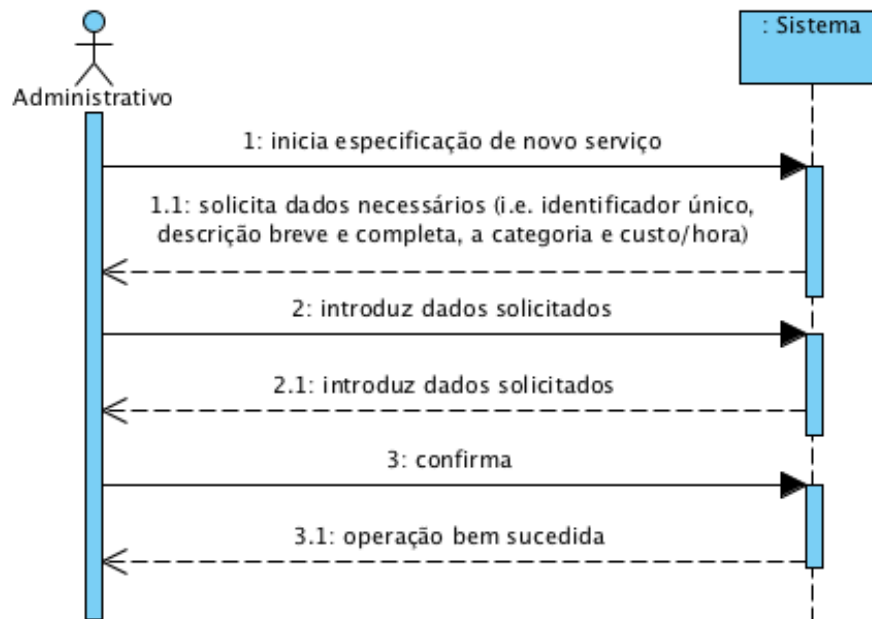
Engenharia de Requisitos:  
UC Especificar Serviço

# UC ESPECIFICAR SERVIÇO (1/7)

## ○ Formato Breve

O administrativo inicia a especificação de um novo serviço. O sistema solicita os dados necessários (i.e. identificador único, descrição breve e completa, a categoria em que é catalogado e custo/hora). O administrativo introduz os dados solicitados. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme. O administrativo confirma. O sistema regista os dados e informa o administrativo do sucesso da operação.

## ○ SSD



# UC ESPECIFICAR SERVIÇO (2/7)

- Actor principal
  - Administrativo
- Partes interessadas e seus interesses:
  - Administrativo: pretende especificar os serviços prestados para que estes possam ser solicitados pelos clientes.
  - Cliente: pretende conhecer os serviços que pode solicitar.
  - Empresa: pretende que os serviços estejam descritos em rigor/detalhe e bem catalogados.
- Pré-condições:
  - O administrativo está identificado e autenticado
- Pós-condições:
  - A informação do serviço é registada no sistema.

## UC ESPECIFICAR SERVIÇO (3/7)

- Cenário de sucesso principal (ou fluxo básico):
  1. O administrativo inicia a especificação de um novo serviço.
  2. O sistema solicita os dados necessários (i.e. identificador único, descrição breve e completa e o custo/hora).
  3. O administrativo introduz os dados solicitados.
  4. **O sistema mostra a lista de categorias existentes para que seja selecionada uma.**
  5. **O administrativo seleciona a categoria em que pretende catalogar o serviço.**
  6. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme.
  7. O administrativo confirma.
  8. O sistema regista os dados e informa o administrativo do sucesso da operação.



# UC ESPECIFICAR SERVIÇO (4/7)

- Extensões (ou fluxos alternativos):

- \*a. O administrativo solicita o cancelamento da especificação de serviço.

- 1. O caso de uso termina.

- 4a. Não existem categorias de serviços definidas no sistema:

- 1. O sistema informa o administrativo de tal facto.

- 2. O sistema permite a criação de uma nova categoria (UC 3).

- 2a. O administrativo não cria uma categoria. O caso de uso termina.

- 6a. Dados mínimos obrigatórios em falta.

- 1. O sistema informa quais os dados em falta.

- 2. O sistema permite a introdução dos dados em falta (passo 3)

- 2a. O administrativo não altera os dados. O caso de uso termina.

## UC ESPECIFICAR SERVIÇO (5/7)

### ○ Extensões (continuação):

6b. O sistema deteta que os dados (ou algum subconjunto dos dados) introduzidos devem ser únicos e que já existem no sistema.

1. O sistema alerta o administrativo para o facto.

2. O sistema permite a sua alteração (passo 3)

2a. O administrativo não altera os dados. O caso de uso termina.

6c. O sistema deteta que os dados introduzidos (ou algum subconjunto dos dados) são inválidos.

1. O sistema alerta o administrativo para o facto.

2. O sistema permite a sua alteração (passo 3).

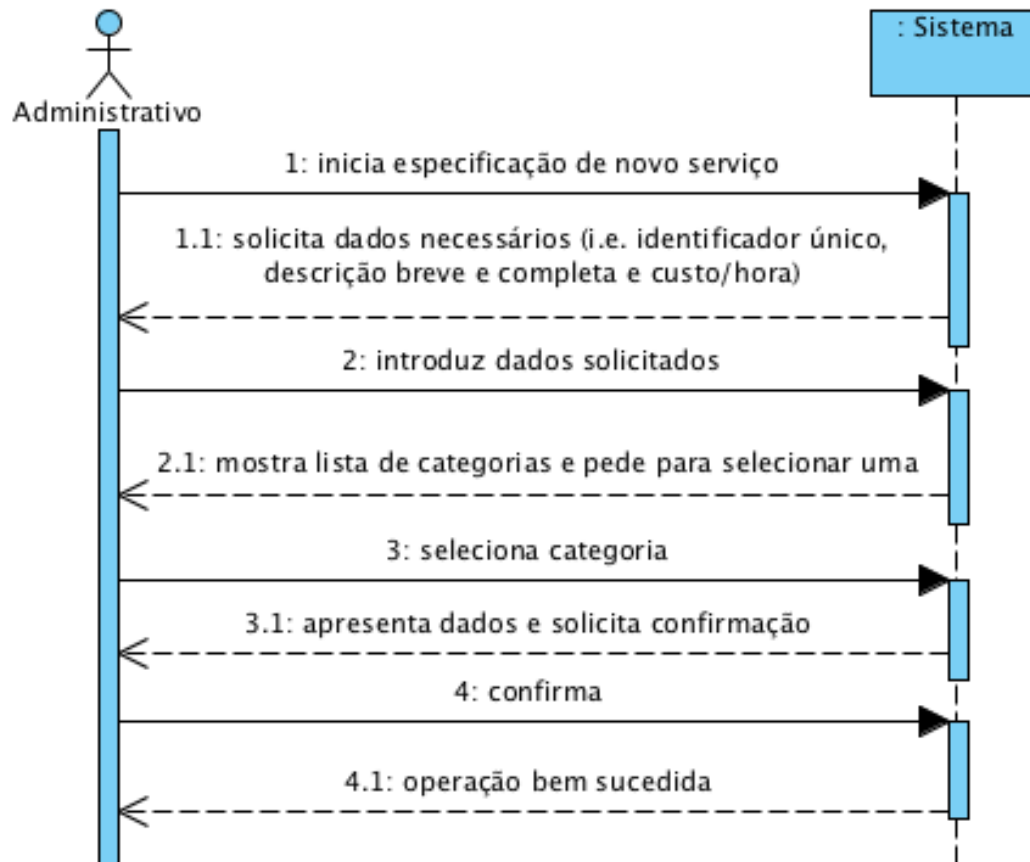
2a. O administrativo não altera os dados. O caso de uso termina.

# UC ESPECIFICAR SERVIÇO (6/7)

- Requisitos especiais:
  - -
- Tecnologia e Lista de Variações de Dados
  - -
- Frequência de ocorrência
  - -
- Questões em aberto:
  - Existem outros dados que são necessários?
  - Todos os dados são obrigatórios para a especificação de um serviço?
  - É possível especificar um serviço sem categoria associada?
  - Pode um serviço pertencer a mais do que uma categoria?
  - Quais os dados que em conjunto permitem detetar a duplicação de serviços?
  - O identificador único é sempre introduzido pelo administrativo ou o sistema deve gerá-lo automaticamente?
  - Qual é a diferença entre descrição breve e completa? Apenas o comprimento de texto? Outra?
  - É preciso guardar o histórico de alteração de custo de um serviço?
  - Qual a frequência de ocorrência deste caso de uso?

# UC ESPECIFICAR SERVIÇO (7/7)

## ○ SSD – Cenário de Sucesso do Formato Completo



**Nota:** serve apenas para evidenciar que o SSD varia em função do formato/cenário adotado



# ANÁLISE OO

39

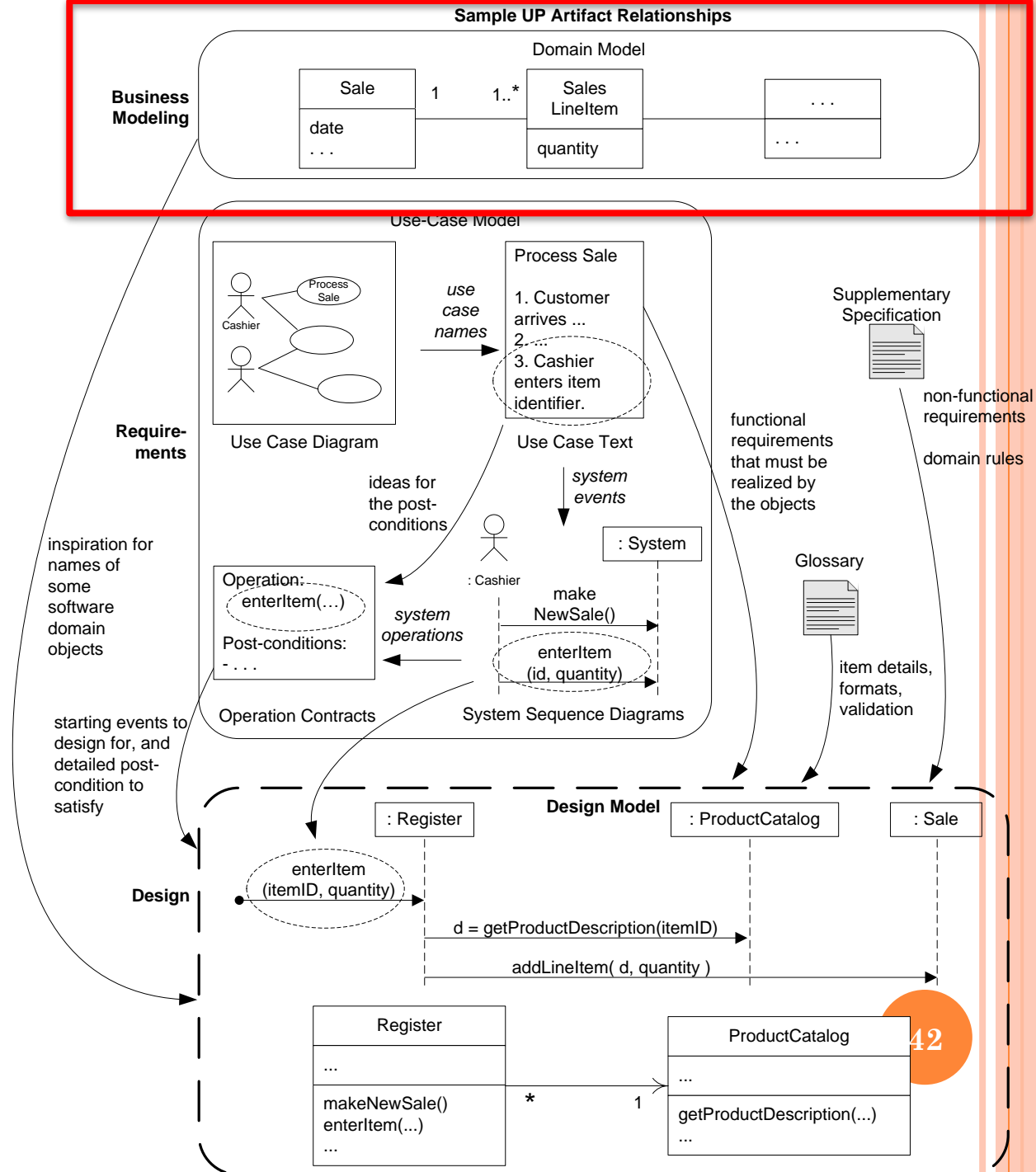
# ANÁLISE

- Analisar, implica:
  - Adquirir conhecimento sobre o Domínio/Negócio
  - Compreender o Domínio/Negócio
  - Pensar sobre o Domínio/Negócio
- Análise Orientada por:
  - **Objetos**
  - Processos/Atividades

# ANÁLISE ORIENTADA POR OBJETOS

- Nota: Engenharia de Requisitos não é uma disciplina OO
- Quais são os objetos de domínio?
- Descritos num modelo de objetos de domínio (“domain (object) model”)
  - Perspectiva de classificação de objetos
  - Objetivo: **diminuir o fosso representacional** entre requisitos e design

# ARTEFACTOS: VISÃO GERAL





# MODELO DE DOMÍNIO

- Artefacto da disciplina de “Modelação de Negócio”;
- Representação visual (UML) das classes conceptuais ou de objetos reais do domínio de interesse
  - Posteriormente, poderão ou não vir a ser elementos de software
  - Classes conceptuais  $\approx$  Entidades/Conceitos de negócio
- Identificação:
  - dos conceitos principais
  - das associações entre conceitos
  - dos seus atributos
- Dados de Entrada:
  - Artefactos descritivos do domínio (e.g. UC, ES)

# MODELO DE DOMÍNIO - ELEMENTOS

- **Conceitos** (ou classes conceptuais)
  - Tipicamente, algo mais complexo que um número ou um texto
  - E.g. Estudante, Disciplina
- **Associações** entre conceitos
  - São relações entre conceitos
  - São dependentes do que interessa para o problema específico
  - E.g. : Estudante **está inscrito em** Disciplina
- **Atributos**
  - Tipicamente, algo associado a um conceito que é expresso numericamente ou em texto
  - E.g. O número, nome e data de nascimento de um Estudante

# COMO IDENTIFICAR CLASSES CONCEPTUAIS?

- Usar **lista de categorias** comuns
  - Ênfase nas necessidades de informação do negócio
  - Resulta numa lista de classes conceptuais candidatas
- Identificação de **substantivos** nas frases
  - Análise linguística dos dados de entrada
  - Resulta numa lista de classes conceptuais candidatas
  - **Aviso: não aplicar um mapeamento mecânico de substantivos em conceitos; ter em atenção a ambiguidade das palavras**
- Alterar/Reutilizar modelos existentes
  - Existem para os domínios mais comuns como vendas, stocks, finanças, saúde, etc...

# LISTA DE CATEGORIAS COMUNS

## ○ Classes conceptuais candidatas:

- Transações (do negócio)
- Linhas de transações
- Produtos ou serviços relacionados com transações
- Registos (de transações)
- Papéis das pessoas
- Lugares
- Eventos
- Objetos físicos
- Especificações e descrições
- Catálogos
- Conjuntos (*containers*)
- Elementos de conjuntos
- (Outras) Organizações
- Outros sistemas (externos)
- Registos (financeiros), de trabalho, contractos, documentos legais
- Instrumentos financeiros
- Documentos referidos/para executar as tarefas

# IDENTIFICAÇÃO DAS CLASSES CONCEPTUAIS

- Listar a partir dos casos de uso, classes conceptuais candidatas:

## UC 2: Definir Área de Atividade:

1. O **administrativo** inicia a definição de uma nova **área de atividade**.
2. O sistema solicita os dados necessários (i.e. código único, descrição breve e detalhada).
3. O administrativo introduz os dados solicitados.
4. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme.
5. O administrativo confirma.
6. O sistema regista os dados e informa o administrativo do sucesso da operação.

Categoria	Classes candidatas
Objectos físicos	
Especificações e descrições	Área Atividade, Competência Técnica
Lugares	Endereço Postal
Transacções	
Linhas de transacções	
Papéis das pessoas	Administrativo, Freelancer
Produtos ou serviços relacionados com transações	
Catálogos	
Organizações	Organização, T4J (Plataforma)
Conjuntos (containers)	
Elementos de conjuntos	
etc.	

# EXEMPLO: CLASSES CONCEPTUAIS CANDIDATAS

## UC 2: Definir Área de Atividade:


1. O **administrativo** inicia a definição de uma nova **área de atividade**.
2. O sistema solicita os dados necessários (i.e. código único, descrição breve e detalhada).
3. O administrativo introduz os dados solicitados.
4. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme.
5. O administrativo confirma.
6. O sistema regista os dados e informa o administrativo do sucesso da operação.



Categoria	Classes candidatas
Objectos físicos	
Especificações e descrições	Área Atividade, Competência Técnica
Lugares	Endereço Postal
Transacções	
Linhas de transacções	
Papéis das pessoas	Administrativo, Colaborador
Produtos ou serviços relacionados com transacções	
Catálogos	
Organizações	Organização, T4J (Plataforma)


 Plataforma

 AreaAtividade

 Colaborador

 EnderecoPostal

 CompetênciaTécnica

 Organização

 Administrativo

# IDENTIFICAÇÃO DE ASSOCIAÇÕES

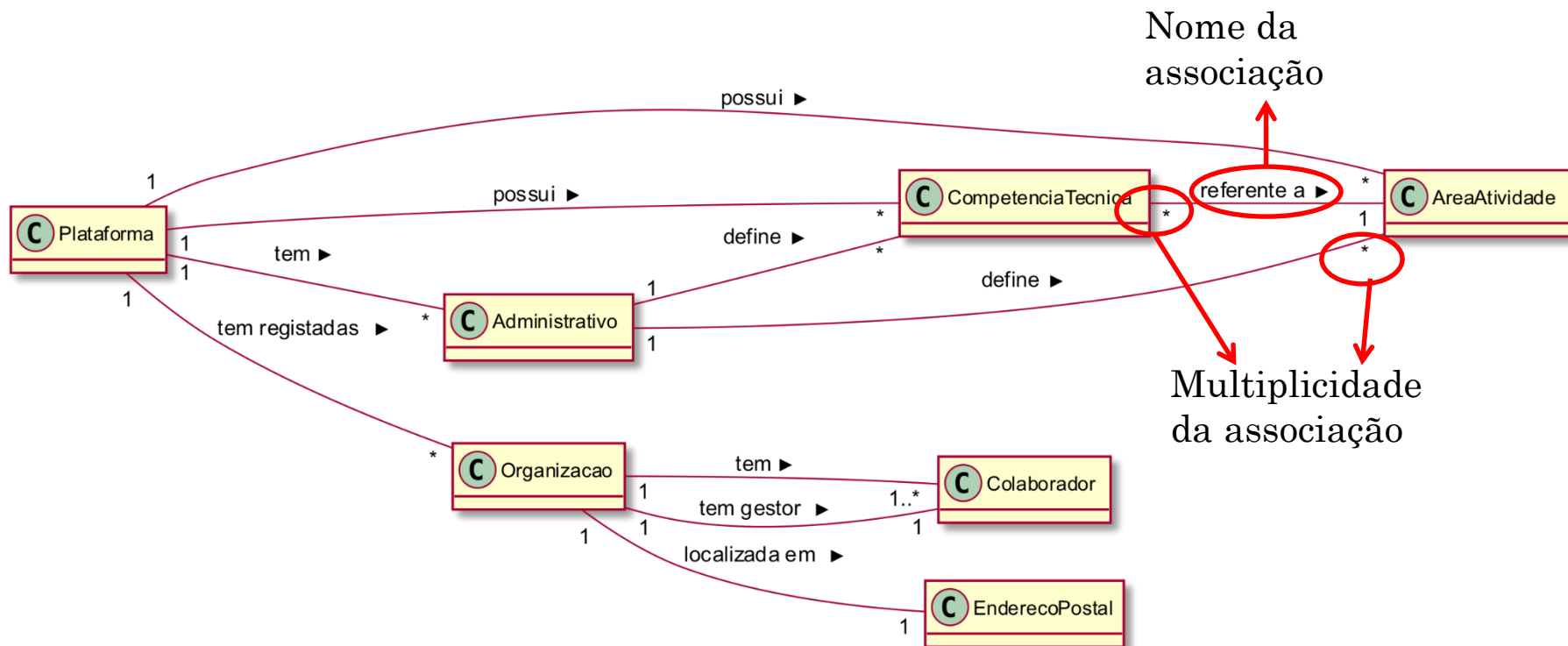
- Uma associação é uma relação entre instâncias de objetos que indica uma conexão relevante e que:
  - vale a pena recordar, ou
  - é derivável da Lista de Associações Comuns:
    - A é fisicamente (ou logicamente) parte de B
    - A está fisicamente (ou logicamente) contido em B
    - A é uma descrição de B
    - A é conhecido/capturado/registado por B
    - A usa ou gere B
    - A está relacionado com uma transacção de B
    - etc.
- Em cada iteração, deve-se focar os esforços nas relações mais importantes

# TABELA DE ASSOCIAÇÕES CANDIDATAS - EXEMPLO

Conceito A	Associação	Conceito B
Administrativo	<ul style="list-style-type: none"> <li>• define</li> <li>• define</li> <li>• trabalha para</li> </ul>	<ul style="list-style-type: none"> <li>• ÁreaAtividade</li> <li>• CompetênciaTécnica</li> <li>• Plataforma</li> </ul>
Organização	<ul style="list-style-type: none"> <li>• possui</li> <li>• tem</li> <li>• tem gestor</li> </ul>	<ul style="list-style-type: none"> <li>• EndereçoPostal</li> <li>• Colaborador</li> <li>• Colaborador</li> </ul>
CompetênciaTécnica	<ul style="list-style-type: none"> <li>• referente a</li> </ul>	<ul style="list-style-type: none"> <li>• ÁreaAtividade</li> </ul>
Plataforma	<ul style="list-style-type: none"> <li>• tem registadas</li> <li>• tem/possui</li> </ul>	<ul style="list-style-type: none"> <li>• Organização</li> <li>• Administrativo</li> </ul>
...	<ul style="list-style-type: none"> <li>• ...</li> </ul>	<ul style="list-style-type: none"> <li>• ...</li> </ul>

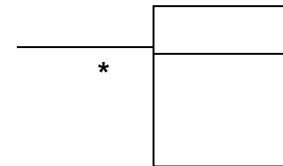


## EXEMPLO: ASSOCIAÇÕES (DENOMINAÇÃO E MULTIPLICIDADE)

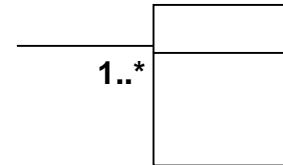


# MULTIPLICIDADE

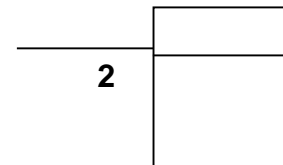
❖ Zero ou  
mais



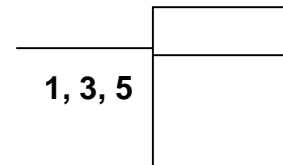
❖ Um ou mais



❖ Exatamente dois



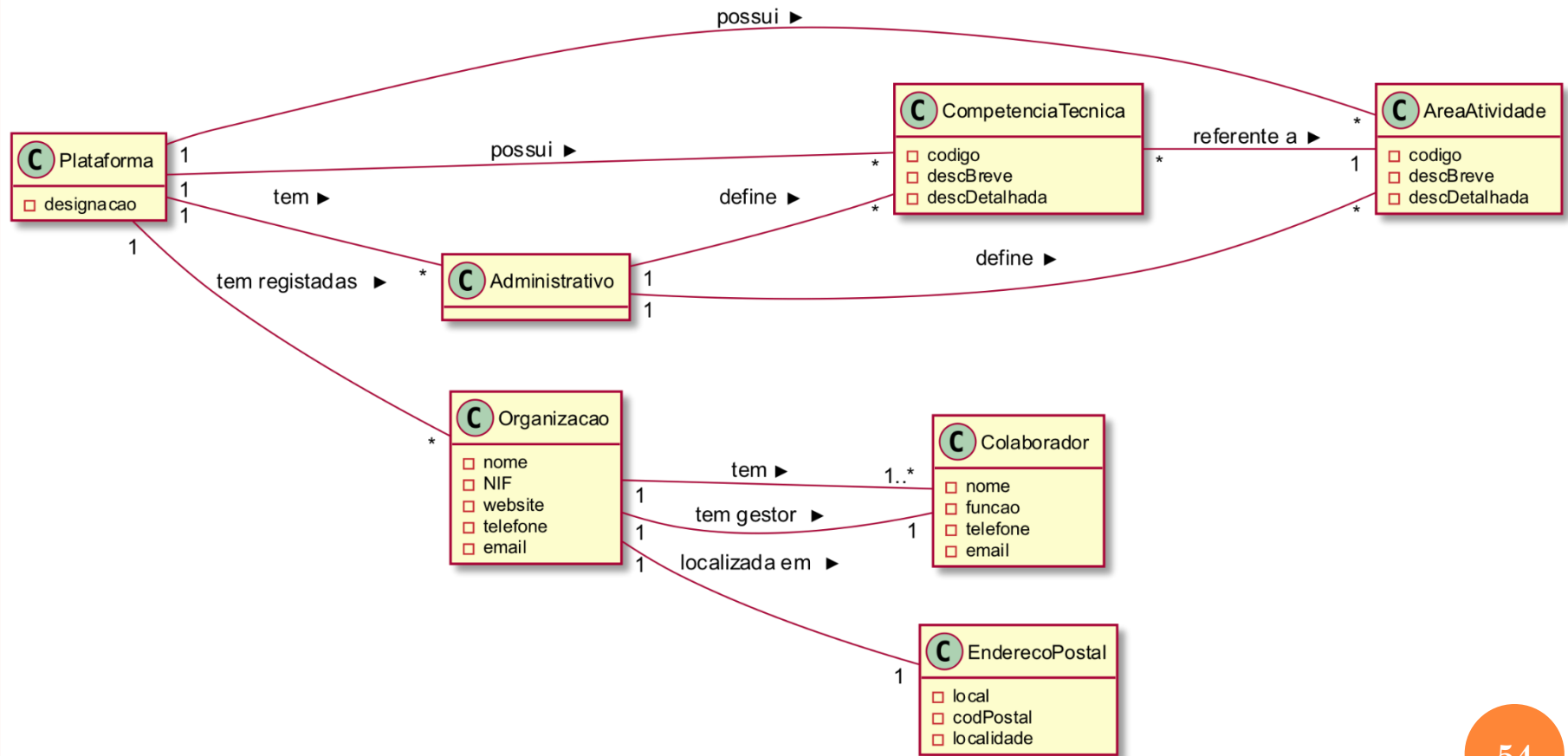
❖ Um, três ou cinco



# IDENTIFICAÇÃO DE ATRIBUTOS

- Preferencialmente devem ser de tipos simples e primitivos
  - E.g. booleano, data, número, string, texto, tempo
- Deve representar-se atributos como classes conceptuais se:
  - O “atributo” é constituído por secções separadas
    - E.g.: Endereço: rua, nº, código postal, país
  - Tem operações associadas (e.g. parsing)
  - Tem outros atributos
  - É uma abstracção de um ou mais tipos
    - E.g.: Código Barras: UPC, EAN

# EXEMPLO: ATRIBUTOS



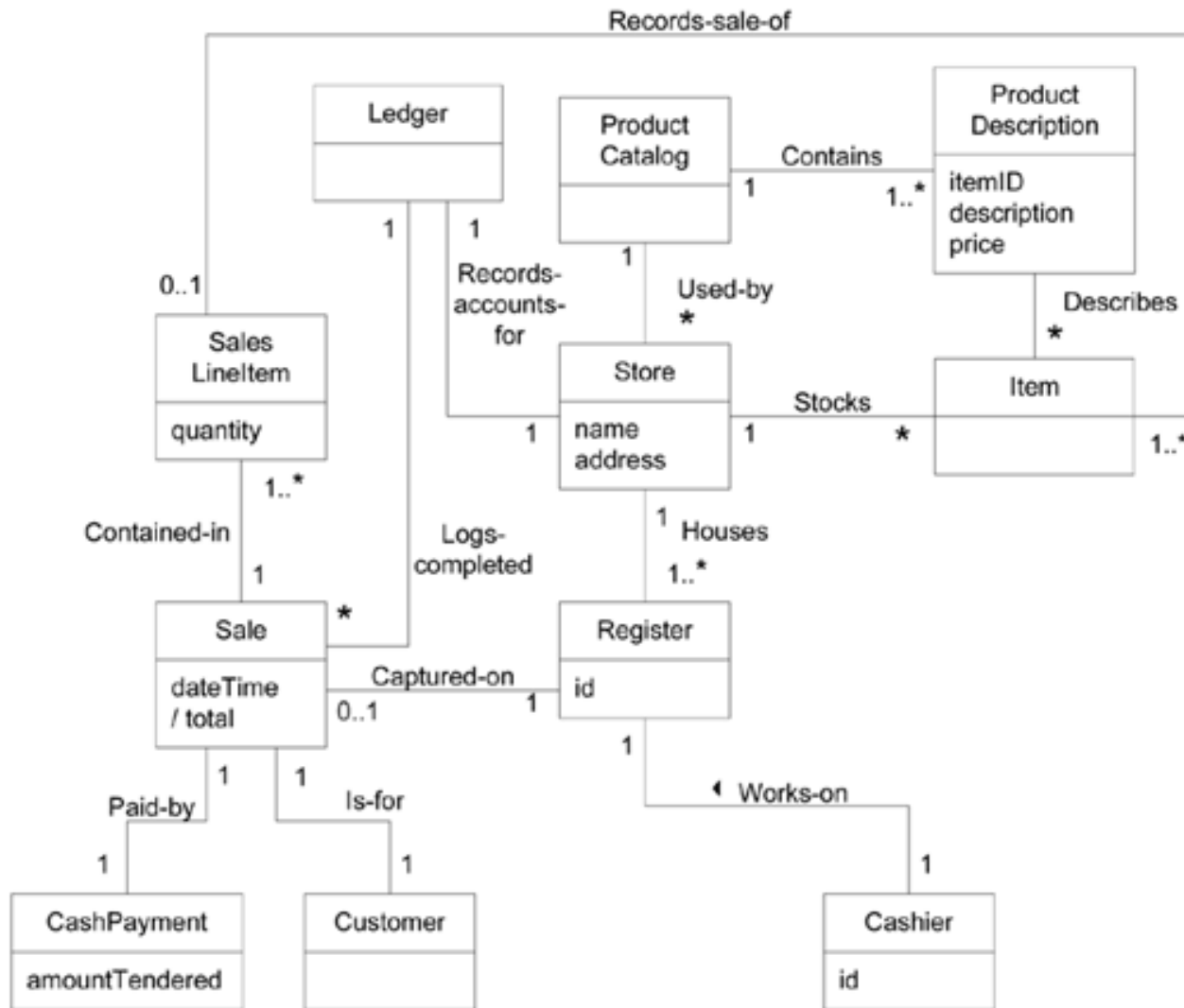
# CONVENÇÕES

- Nomes de classes conceptuais
  - Começam com uma letra maiúscula
  - Estão no singular
- Nomes de associações
  - Começam com uma letra minúscula
  - Escolhido de forma a que
 

`<Classe1> <nome da associação> <Classe2>`  
 seja uma frase com sentido
  - Leitura
    - Da esquerda para a direita
    - De cima para baixo
    - (caso contrário tem que ter indicação de sentido)
- Nomes de atributos
  - Começam com uma letra minúscula

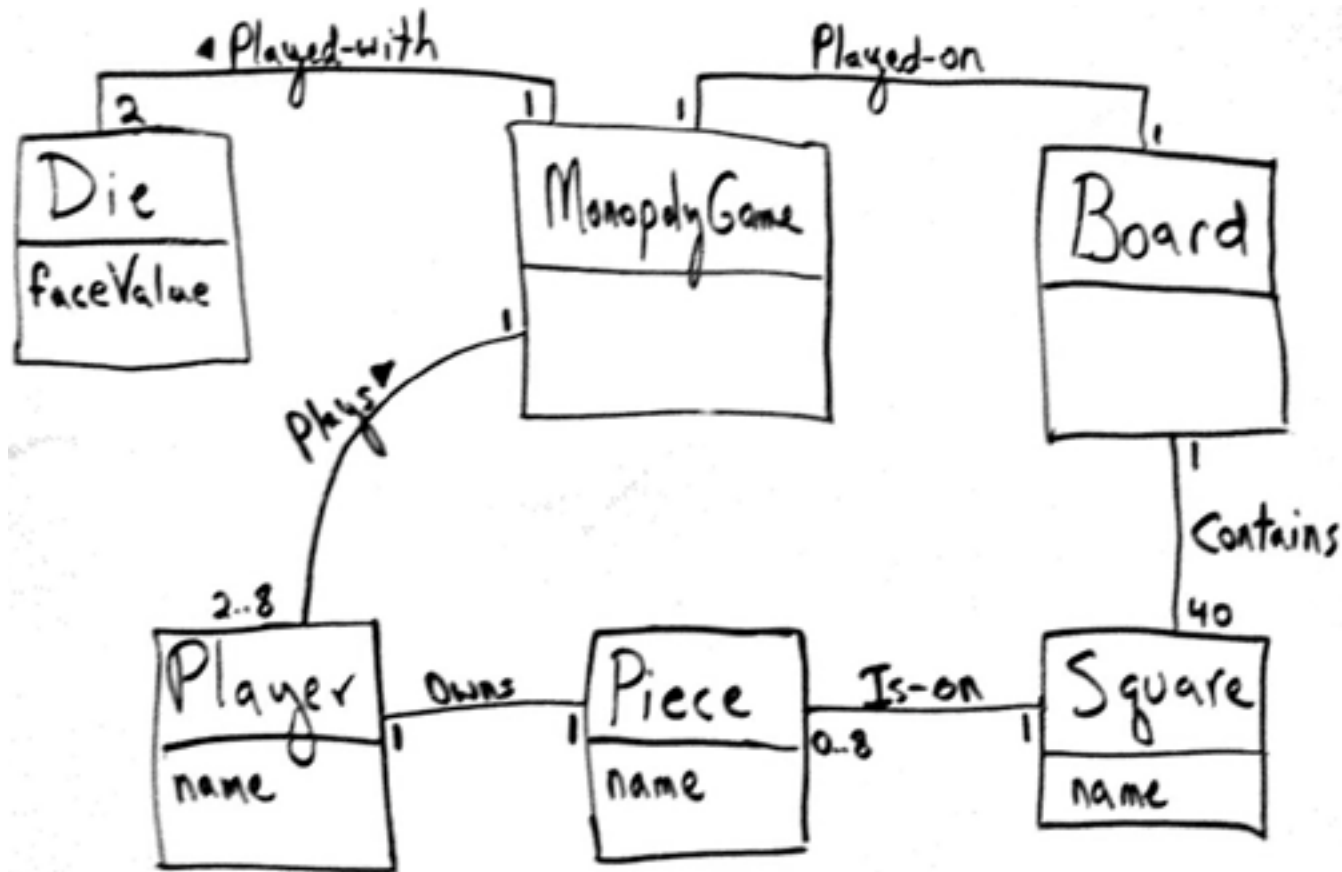
# OUTROS EXEMPLOS:

## → MD PARCIAL DO NEXTGEN POS



## OUTROS EXEMPLOS:

→ MD PARCIAL DO JOGO MONOPÓLIO



# ESTÁ O MEU MODELO DE DOMÍNIO CORRETO?

- Não existe apenas um MD que seja o correto
  - Diferentes MD podem estar corretos
  - Todos são aproximações ao domínio que estamos a tentar compreender
- MD deve ser visto como uma ferramenta
  - Para compreender o domínio
  - Para comunicar com
    - O Cliente
    - A equipa de desenvolvimento
- MD é útil quando
  - Capta as abstrações e informações essenciais necessárias para a compreensão do domínio no contexto das requisitos atuais
  - Auxilia as pessoas na compreensão dos conceitos, da terminologia e das relações de domínio





60

## EXEMPLO

**Modelo de Domínio:  
Enfâse no UC Especificar Serviço**

# IDENTIFICAÇÃO DAS CLASSES CONCEPTUAIS

- Listar a partir dos casos de uso, classes conceptuais candidatas:

Categoria	Classes candidatas
Objectos físicos	
Especificações e descrições	Categoria, Serviço
Lugares	Endereço Postal
Transacções	Pedido Prestação Serviço
Linhas de transacções	Descrição Serviço Pedido
Papéis das pessoas	Administrativo, Funcionário RH, Cliente
Produtos ou serviços relacionados com transações	Serviço Prestado
Catálogos	
Organizações	Empresa
Conjuntos (containers)	
Elementos de conjuntos	
etc.	

## UC Especificar Serviço:

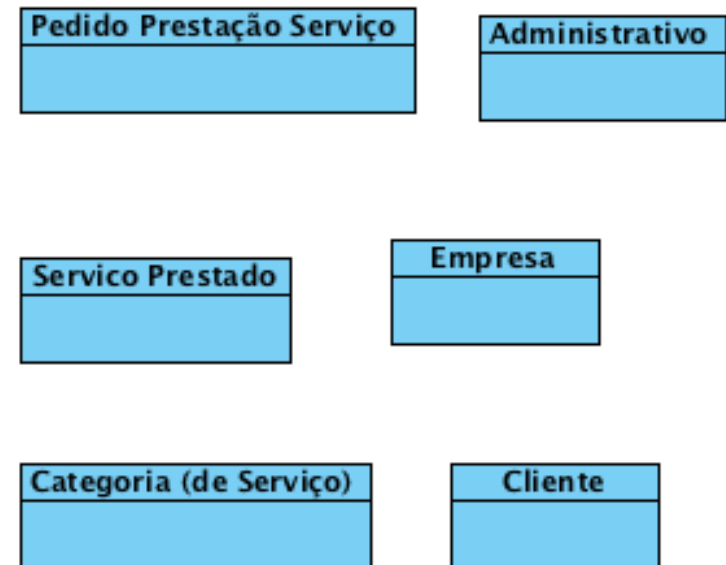
1. O **administrativo** inicia a **especificação de um novo serviço**.
2. O sistema solicita os dados necessários (i.e. identificador único, descrição breve e completa e o custo/hora).
3. O administrativo introduz os dados solicitados.
4. O sistema mostra a **lista de categorias** existentes para que seja selecionada uma.
5. O administrativo seleciona a **categoria** em que pretende **catalogar o serviço**.
6. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme.
7. O administrativo confirma.
8. O sistema regista os dados e informa o administrativo do sucesso da operação.

# EXEMPLO: CLASSES CONCEPTUAIS CANDIDATAS

## UC Especificar Serviço:

1. O **administrativo** inicia a **especificação de um novo serviço**.
2. O sistema solicita os dados necessários (i.e. identificador único, descrição breve e completa e o custo/hora).
3. O administrativo introduz os dados solicitados.
4. O sistema mostra a **lista de categorias** existentes para que seja selecionada uma.
5. O administrativo seleciona a **categoria** em que pretende **catalogar o serviço**.
6. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme.
7. O administrativo confirma.
8. O sistema regista os dados e informa o administrativo do sucesso da operação.

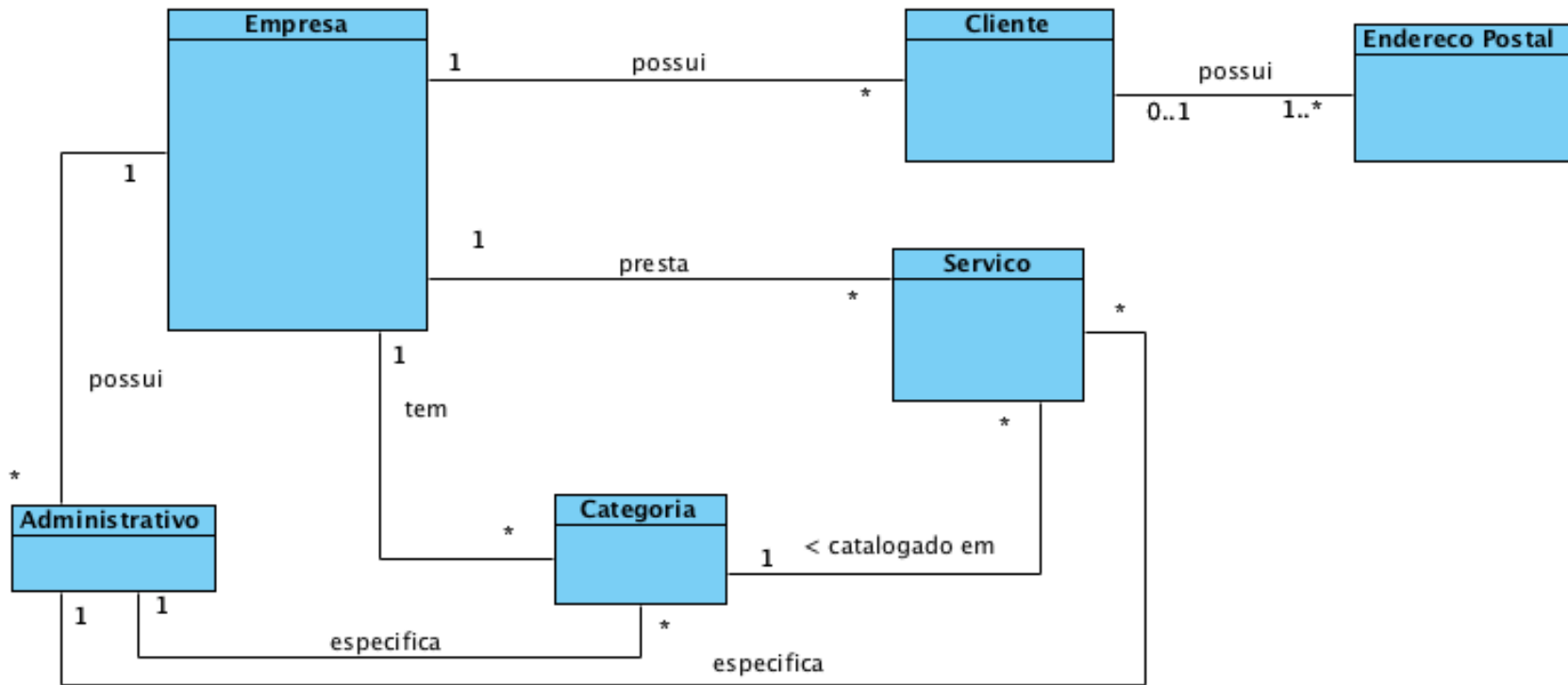
Categoria	Classes candidatas
Objectos físicos	
Especificações e descrições	Categoria, Serviço
Lugares	Endereço Postal
Transacções	Pedido Prestação Serviço
Linhas de transacções	Descrição Serviço Pedido
Papéis das pessoas	Administrativo, Funcionário RH, Cliente
Produtos ou serviços relacionados com transações	Serviço Prestado
Catálogos	
Organizações	Empresa



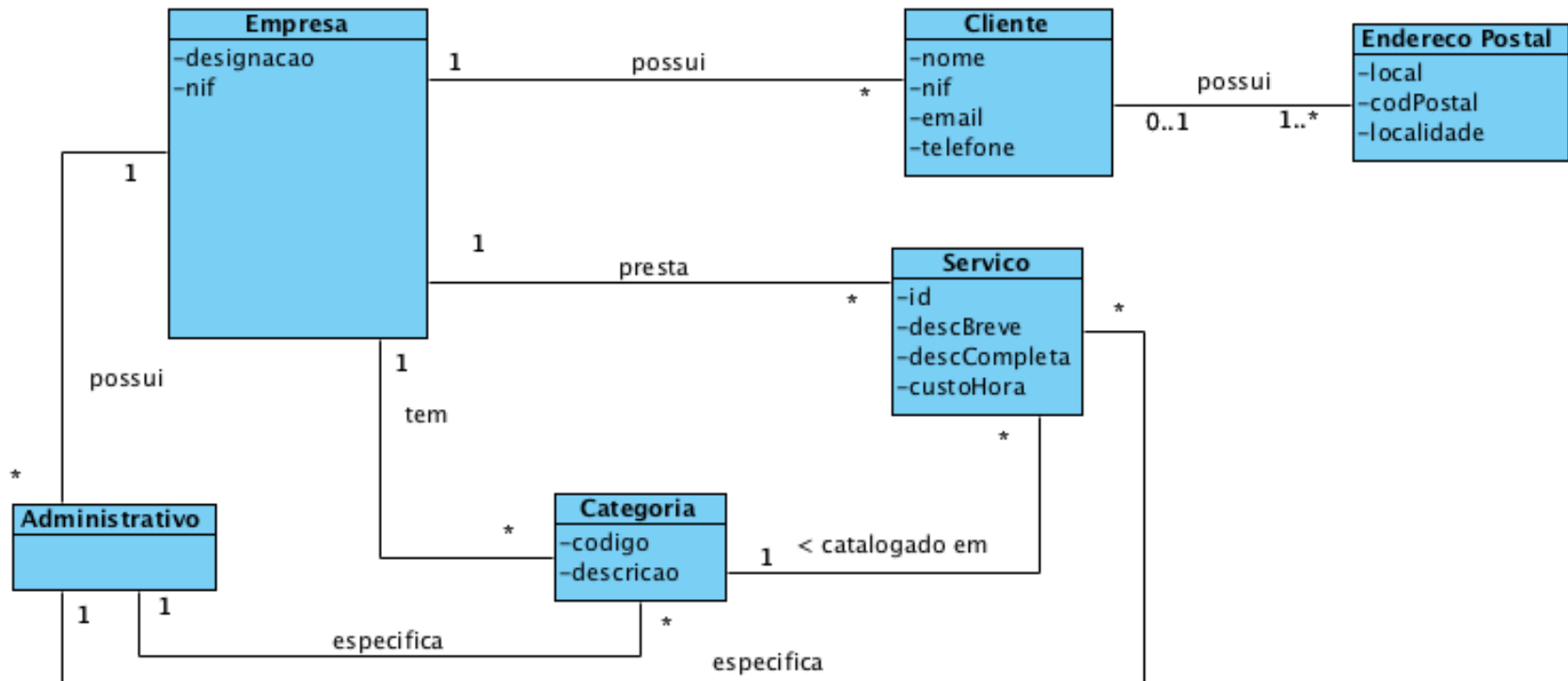
# TABELA DE ASSOCIAÇÕES CANDIDATAS - EXEMPLO

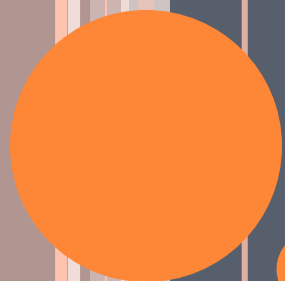
Conceito A	Associação	Conceito B
Administrativo	<ul style="list-style-type: none"><li>• específica</li><li>• específica</li><li>• específica</li></ul>	<ul style="list-style-type: none"><li>• Categoria (de Serviço)</li><li>• Serviço (Prestado)</li><li>• Área Geográfica</li></ul>
Cliente	<ul style="list-style-type: none"><li>• efetua</li></ul>	<ul style="list-style-type: none"><li>• Pedido de Prestação de Serviços</li></ul>
Serviço	<ul style="list-style-type: none"><li>• catalogado</li></ul>	<ul style="list-style-type: none"><li>• Categoria</li></ul>
...	<ul style="list-style-type: none"><li>• ...</li></ul>	<ul style="list-style-type: none"><li>• ...</li></ul>

# EXEMPLO: ASSOCIAÇÕES (DENOMINAÇÃO E MULTIPLICIDADE)



# EXEMPLO: ATRIBUTOS





DESIGN OO

66

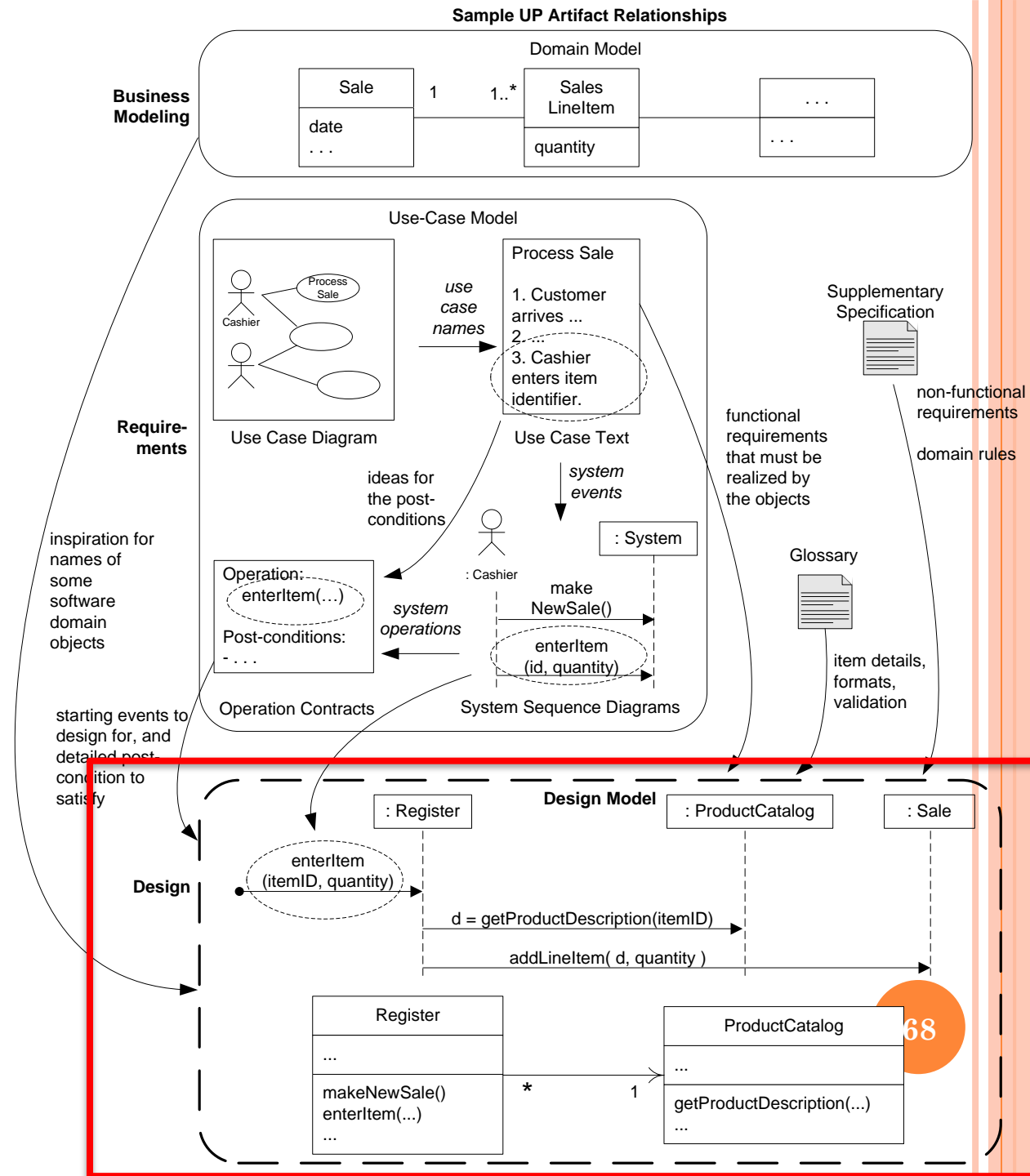
# SUMÁRIO

- Design:
  - O que é?
  - Em que consiste?
  - Como fazer?
- Responsibility-Driven Design (RDD)
- GRASP – General Responsibility Assignment Software Patterns
- Exemplos: Aplicação dos padrões GRASP



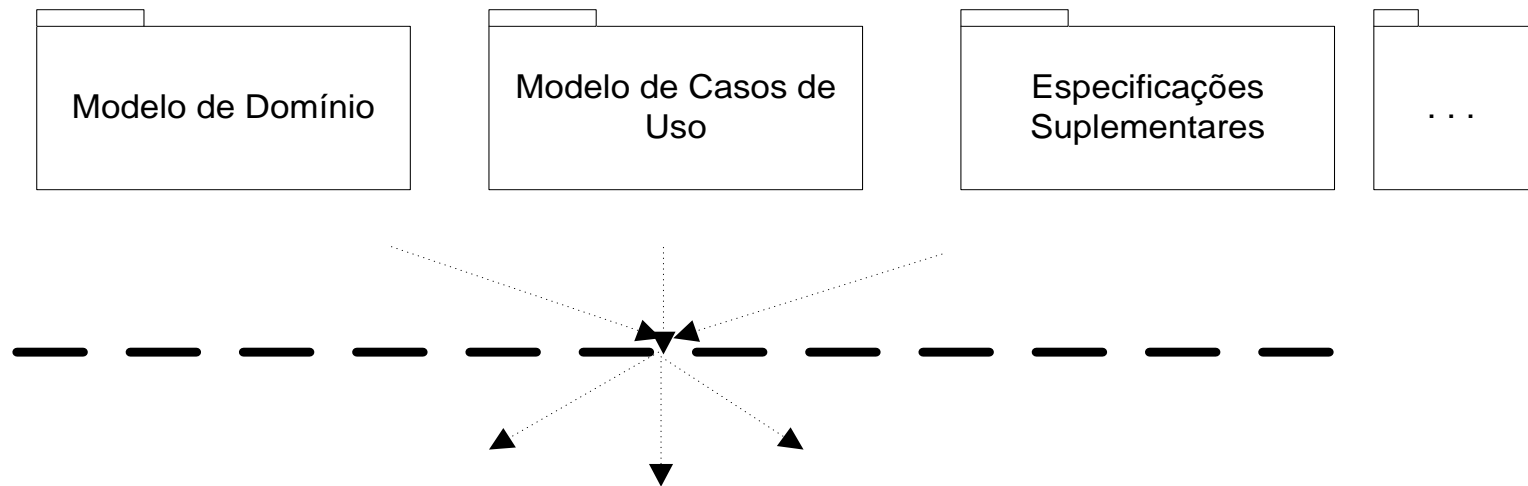
# ARTEFACTOS: VISÃO GERAL

Dos  
requisitos  
para o  
modelo de  
objectos



# DOS REQUISITOS PARA O DESIGN

- Conjunto de artefactos orientados pelos requisitos



- Inspira artefactos orientados para o design



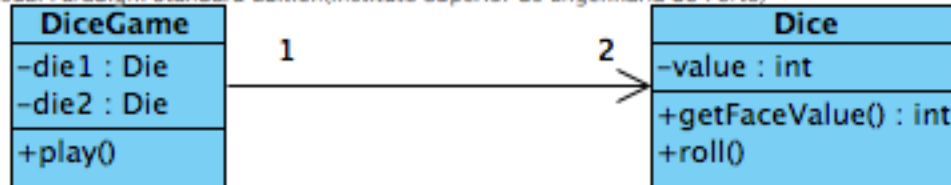
# DESIGN OO

- Solução lógica baseada no paradigma Object-Oriented (OO)
- Modelo de Design inclui:
  - Vista estática: **Diagrama de Classes (DC)**
  - Vista dinâmica (diagramas de interação):
    - **Diagrama de Sequência (DS)**: ilustra as interações dos objetos num formato em “grade”, no qual os objetos são adicionados sucessivamente à direita e em que a ordem das mensagens ocorre de cima para baixo
    - Diagrama de comunicação: ilustra as interações num formato em grafo ou rede, no qual os objetos podem ser colocados em qualquer sítio do diagrama, e em que a ordem das mensagens é definida por um número
- Criar em paralelo
  - Diagramas de Classe
  - Diagramas de Interação

# DIAGRAMAS DE DESIGN

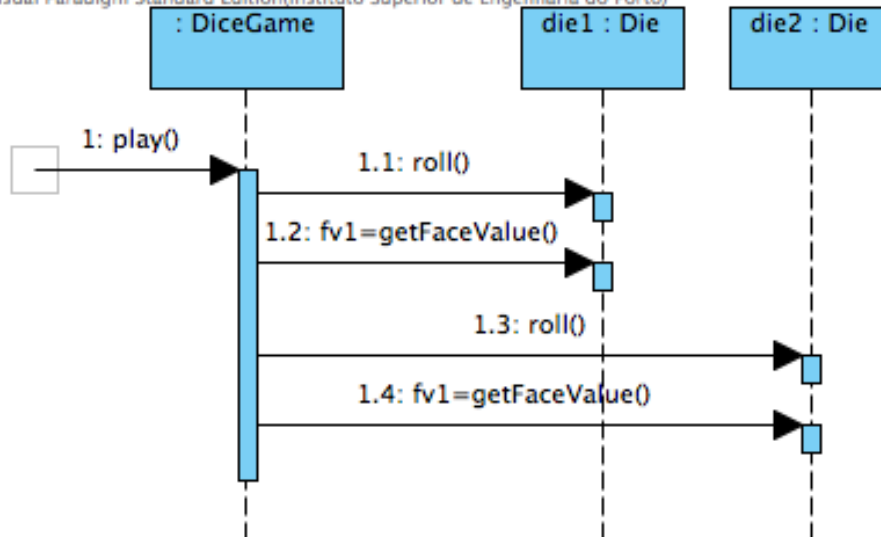
## Diagrama de Classes

Visual Paradigm Standard Edition(Instituto Superior de Engenharia do Porto)



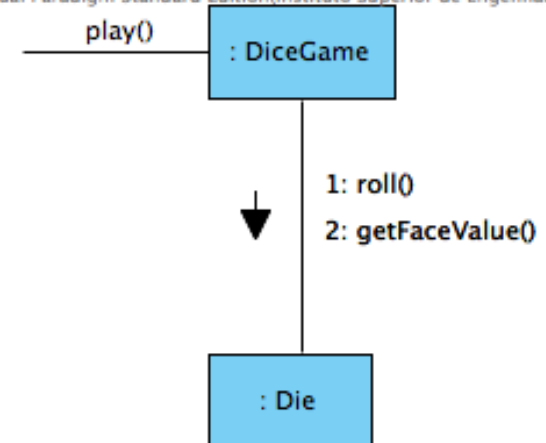
## Diagrama de Sequência

Visual Paradigm Standard Edition(Instituto Superior de Engenharia do Porto)



## Diagrama de Comunicação

Visual Paradigm Standard Edition(Instituto Superior de Engenharia do Porto)



# DESIGN OO

- Design Orientado por Objetos
  - Solução de SW em termos de objetos colaborantes e com responsabilidades
  - Descritos num modelo de design (“design model”)
- Após:
  - identificar os requisitos
  - desenhar o modelo de domínio (classes candidatas)
- então:
  - adicionar métodos às classes de software (derivadas das classes candidatas)
  - definir as mensagens entre classes/objetos
- para dar resposta aos requisitos.

# METODOLOGIA

- Orientada pela **realização de casos de uso**
- Utiliza ainda outros artefactos:
  - E.g.: SSDs e Modelo de Domínio
- Portanto, para cada UC serão criados os seguintes artefactos:
  - Racional de atribuição de responsabilidades segundo
    - **GRASP** (General Responsibility Assignment Software Patterns)
    - SOLID
    - GoF
    - de Persistência
  - Diagrama de Sequência que evidencia as interações
  - Diagrama de Classes parcial
- Diagrama de Classes completo resulta dos DC parciais de cada realização de caso de uso

# CASOS DE USO E REALIZAÇÃO DE CASOS DE USO

## ○ *Use Case Realization*

- “A use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects” [RUP]
- *Realização de um caso de uso* realça a ligação entre os requisitos, expressos por um caso de uso e o design dos objetos que garantem esses requisitos
- Realização de um cenário de um caso de uso: Cobre um cenário de um caso de uso, habitualmente o cenário de sucesso, e não todos os cenários abrangidos num caso de uso, nomeadamente se descrito no formato completo

# ATIVIDADES NO DESENHO DE OBJETOS

- Design OO é baseada na metáfora *Responsibility-Driven Design (RDD)* – *pensar como atribuir* responsabilidades aos objetos
- Aplicar princípios ou padrões GRASP, SOLID, GoF e outros durante o desenho e codificação
  - Padrões de design de software para atribuição de responsabilidades em circunstâncias caracterizadas



A series of vertical stripes in shades of brown, tan, and grey run down the left side of the slide. Overlaid on these stripes are several orange circles of varying sizes. One large circle is positioned near the top left, with several smaller circles scattered below it and to the right.

# RESPONSIBILITY DRIVEN-DESIGN

76

# RESPONSIBILITY-DRIVEN DESIGN (RDD)

- Metáfora para ajudar no processo de desenho de software OO
- O que são responsabilidades?
  - Uma responsabilidade é uma abstração do que o objeto faz
  - São obrigações e comportamentos de um objeto em função do papel que executa
  - Uma responsabilidade não é o mesmo que um método, mas os métodos implementam responsabilidades
- Pensar em objetos de software de forma similar à que pensamos em pessoas com responsabilidades, que colaboram com outras pessoas para executarem o seu trabalho

# RESPONSIBILITY-DRIVEN DESIGN (RDD)

- Responsabilidades de “fazer”:
  - Fazer ele próprio, criar um objeto, fazer cálculos, etc.
  - Delegar, iniciar ações noutros objetos
  - Controlar e coordenar actividades noutros objetos
- Responsabilidades de “saber”
  - Conhecer a sua informação privada
  - Conhecer objetos relacionados
  - Conhecer como obter ou calcular nova informação

# RESPONSIBILITY-DRIVEN DESIGN (RDD)

- *RDD* inclui a ideia de **Colaboração** entre objetos
- Responsabilidades são implementadas através de métodos
  - Que atuam sozinhos; e/ou
  - Colaboram com outros métodos e objetos

- Exemplo (de outro projeto):

A classe *Empresa* pode ter um método *getServicosDeCategoria(catId)* para disponibilizar a lista de todos os *Serviços* catalogados numa *Categoria* identificável através de *catId*. Para dar resposta a essa responsabilidade a *Empresa* pode colaborar com cada objeto *Servico* para inquirir se o mesmo está catalogado na categoria cujo *id* é igual ao especificado através de uma mensagem *estaCatalogado(catId)*. Por sua vez, o *Serviço* pode colaborar com o objeto *Categoria* a que está associado para inquirir este se se identifica pelo *id* em causa através de uma mensagem *temId(catId)*.



# GRASP - GENERAL RESPONSIBILITY ASSIGNMENT SOFTWARE PATTERNS

80

# GENERAL RESPONSIBILITY ASSIGNMENT SOFTWARE PATTERNS (GRASP)

- *GRASP* é uma abordagem metodológica ao Desenho OO
  - Baseada em princípios/padrões de atribuição de responsabilidades
  - Auxílio para perceber o essencial de desenho de objetos
  - Permite aplicar o raciocínio de desenho de uma forma metodológica, racional e compreensível
- Em UML, o desenho dos Diagramas de Interação (DI) é um meio para considerar essas responsabilidades
- Quando se desenha os DI, está-se a decidir sobre quais as responsabilidades a atribuir a cada objecto

# OS PADRÕES GRASP

- Information Expert
- Creator
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Indirection
- Pure Fabrication
- Protected Variations

# INFORMATION EXPERT

- **Problema:** qual é o princípio geral para a atribuição de responsabilidades aos objetos?
- **Solução:** Atribuir a responsabilidade ao *information expert*
  - *i.e.* a classe que contém a informação necessária para desempenhar essa responsabilidade
  - que classe? Inspiramo-nos no Modelo de Domínio



# INFORMATION EXPERT – EXEMPLO (DE OUTRO PROJETO)

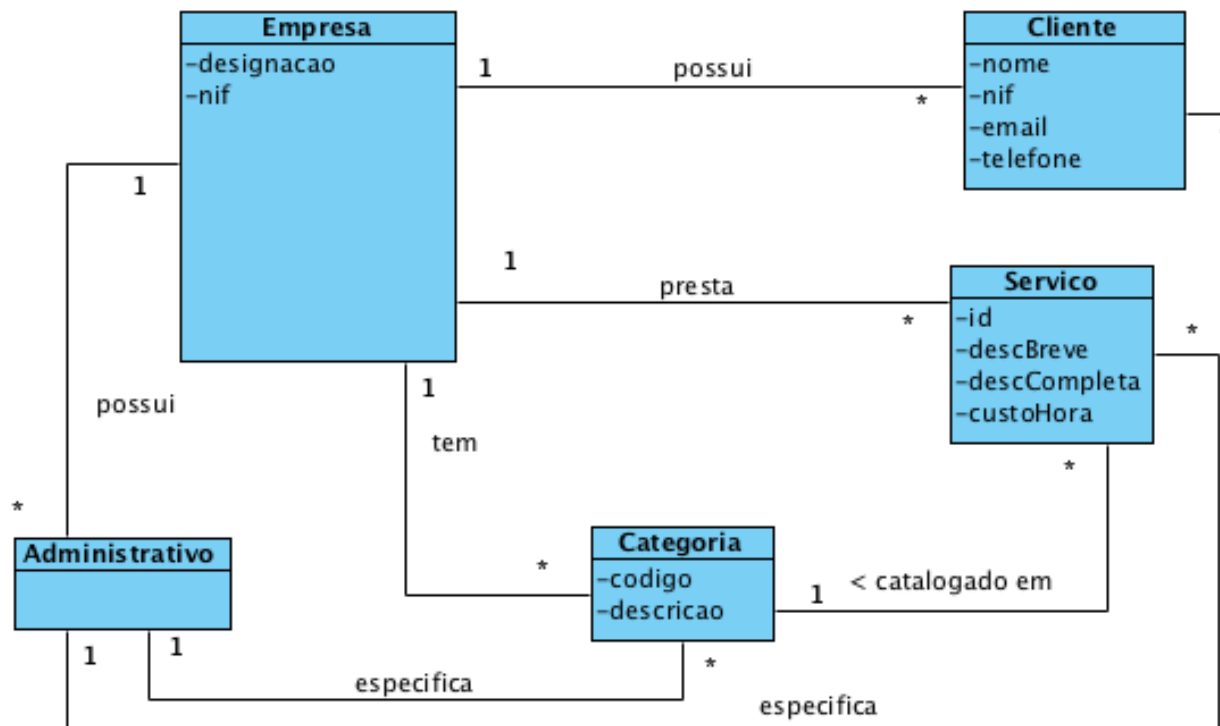
Que classe deve ser responsável por disponibilizar a lista de todos os *Serviços* catalogados numa *Categoria* identificável através de *catId*?



# INFORMATION EXPERT – EXEMPLO (DE OUTRO PROJETO)

Que classe deve ser responsável por disponibilizar a lista de todos os *Serviços* catalogados numa *Categoria* identificável através de *catId*?

- Modelo de domínio: associações entre conceitos do domínio



# INFORMATION EXPERT – EXEMPLO (DE OUTRO PROJETO)

Que classe deve ser responsável por disponibilizar a lista de todos os *Serviços* catalogados numa *Categoria* identificável através de *catId*?

- Que informação precisamos para tal?
  - Quem conhece todos os serviços que existem?  
→ *Empresa*
  - Quem conhece em que categoria um serviço está catalogado?  
→ *Serviço*
  - Quem tem a informação (id) sobre cada categoria?  
→ *Categoria*

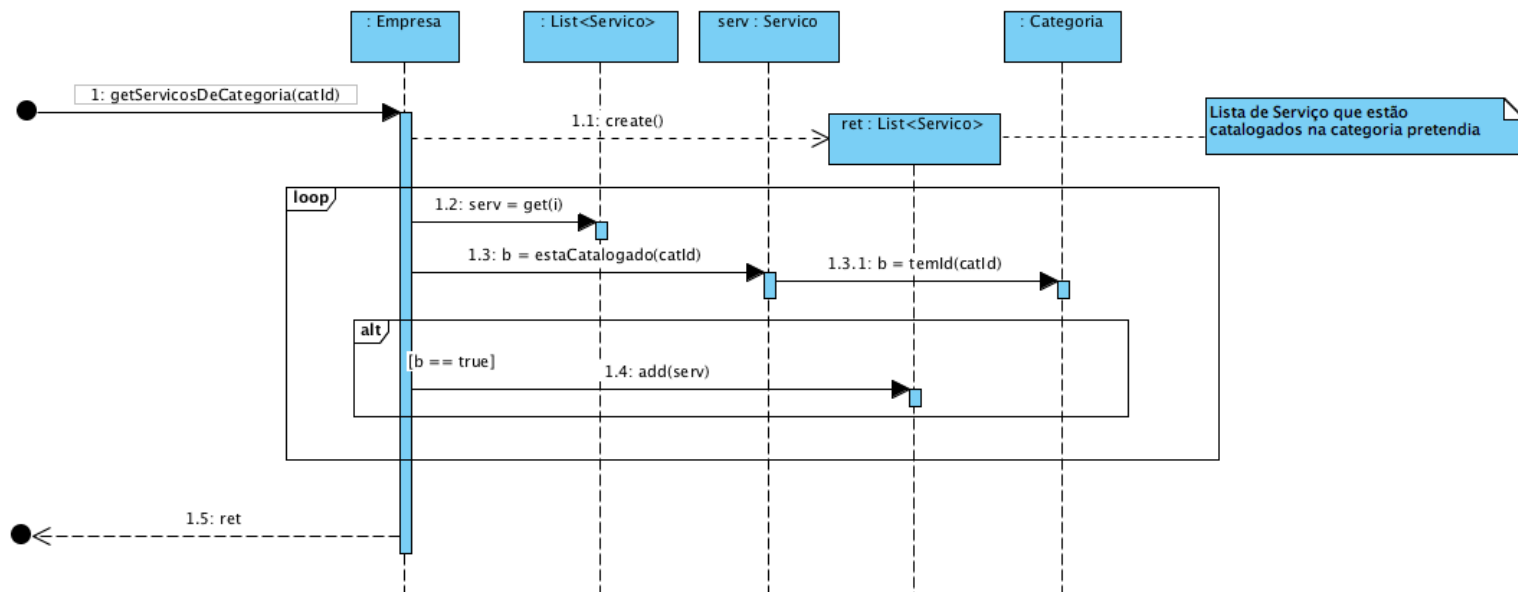
# INFORMATION EXPERT – EXEMPLO (DE OUTRO PROJETO)

- Atribuição de responsabilidades

Classe de Design	Responsabilidade
Empresa	Sabe todos os Serviços
Categoria	Sabe o seu ID
Serviço	Sabe em que categoria está catalogado

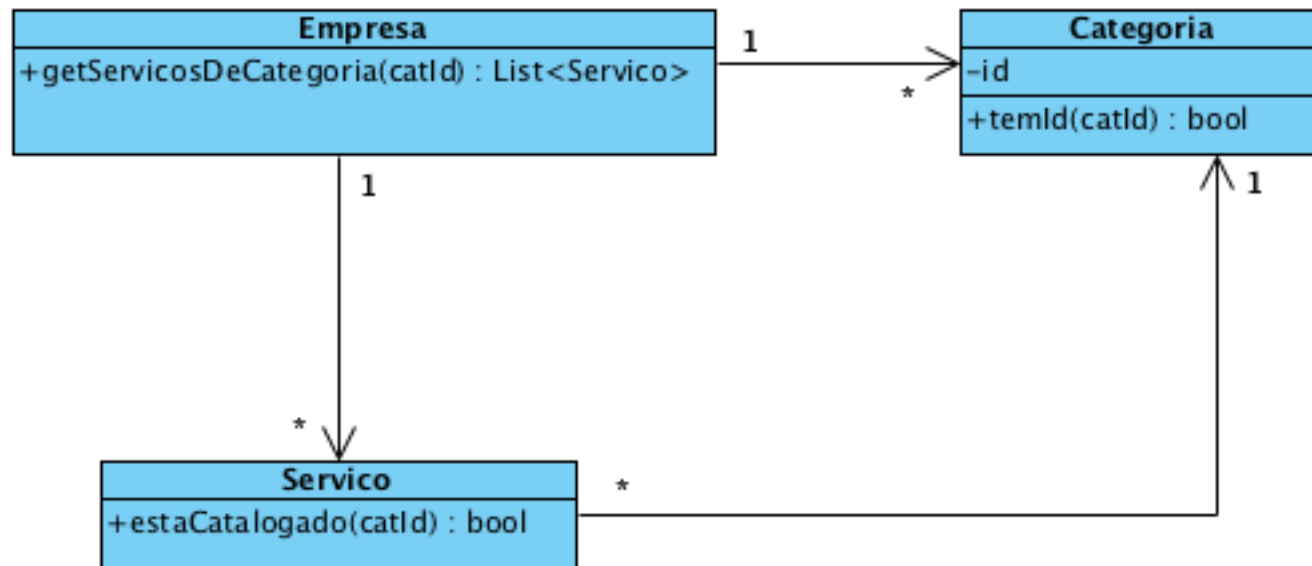
# INFORMATION EXPERT – EXEMPLO (DE OUTRO PROJETO)

Que classe deve ser responsável por disponibilizar a lista de todos os *Serviços* catalogados numa *Categoria* identificável através de *catId*?



# INFORMATION EXPERT – EXEMPLO (DE OUTRO PROJETO)

Que classe deve ser responsável por disponibilizar a lista de todos os *Serviços* catalogados numa *Categoria* identificável através de *catId*?



# INFORMATION EXPERT – CONTRA-INDICAÇÃO

- A utilização de Information Expert por vezes pode causar
  - Problemas de coesão
  - Problemas de acoplamento

Aplicar padrões

- Low Coupling
  - High Cohesion
- E.g.: Quem é o responsável pela gravação da informação de uma venda na base de dados?

# CREATOR

- **Problema:** Quem deve ser responsável pela criação de objetos de uma classe?
- **Solução:** Atribuir à classe B a responsabilidade de criar instâncias da classe A nas seguintes condições (por ordem de preferência):
  - 1) B contém ou agrega objetos da classe A
  - 2) B regista instâncias da classe A
  - 3) B possui os dados usados para inicializar A
  - 4) B está diretamente relacionado com A



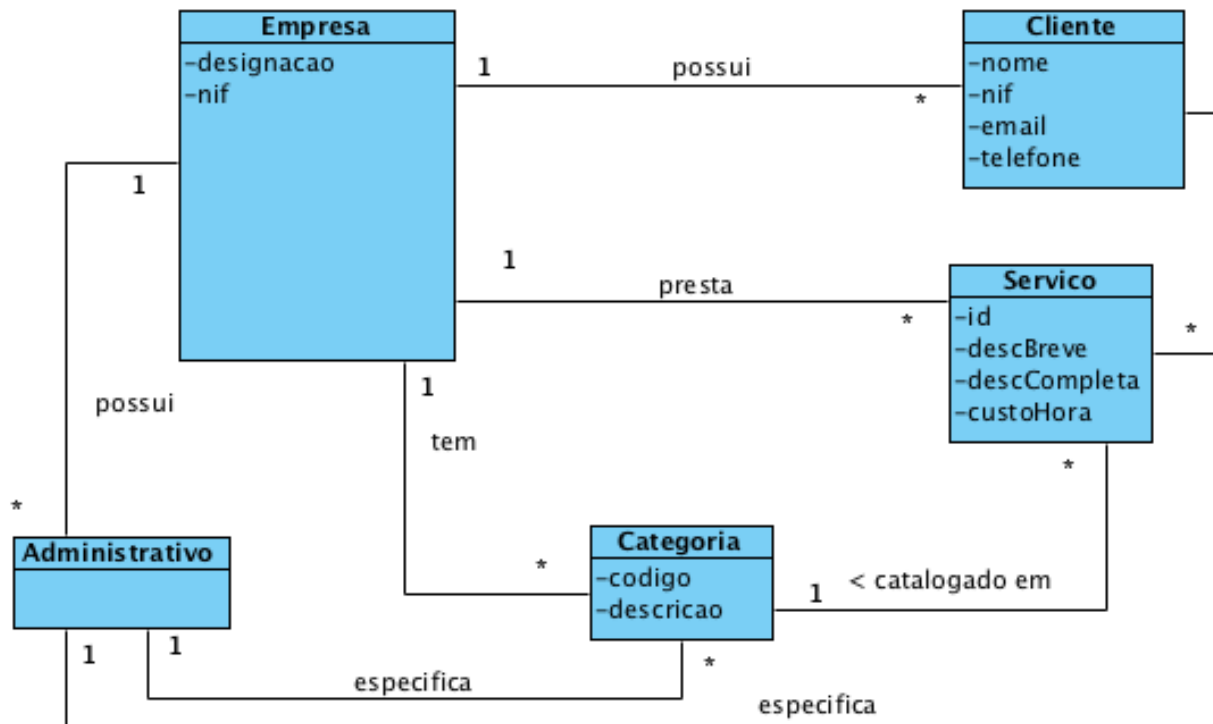
# CREATOR

- ❖ Guia a atribuição da responsabilidade de criar objetos
- ❖ Procuram-se relações de agregação, composição e registo
- ❖ Deve-se escolher a classe que tem a informação necessária para inicializar o objeto
- ❖ A atribuição dessa responsabilidade a uma classe estabelece uma ligação (*coupling*) entre as duas classes

# CREATOR – EXEMPLO (DE OUTRO PROJETO)

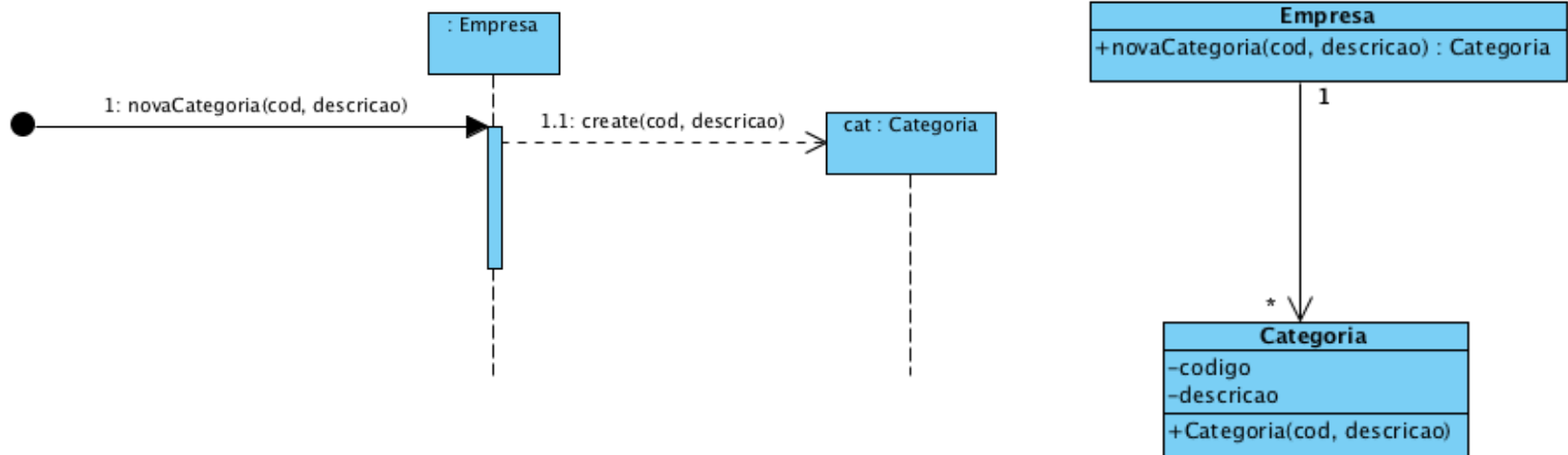
- B contém ou agrega objetos da classe A
- B regista instâncias da classe A
- B possui os dados usados para inicializar A
- B está diretamente relacionado com A

Que classe deve ser responsável pela criação de um novo objeto *Categoria*?



# CREATOR – EXEMPLO (DE OUTRO PROJETO)

Que classe deve ser responsável pela criação de um novo objeto *Categoria*?



- *Empresa* contém ou agrega objetos *Categoria*
- A classe *Empresa* deverá ter um método  
*novaCategoria(cod, descricao)*

# CREATOR – CONTRA-INDICAÇÃO

- ❖ A criação de objetos pode ser
  - ❖ **Complexa**  
e nessas situações pode ser vantajoso
  - ❖ **Delegar** a instanciação noutras classes

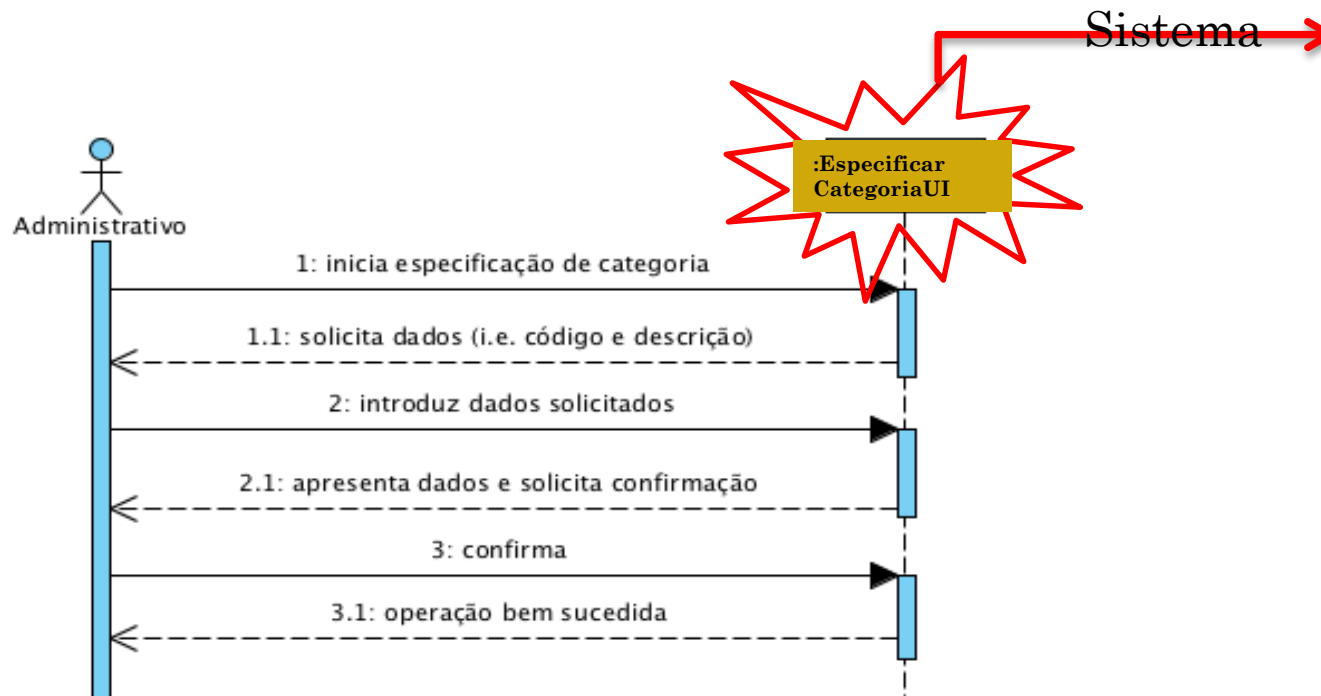
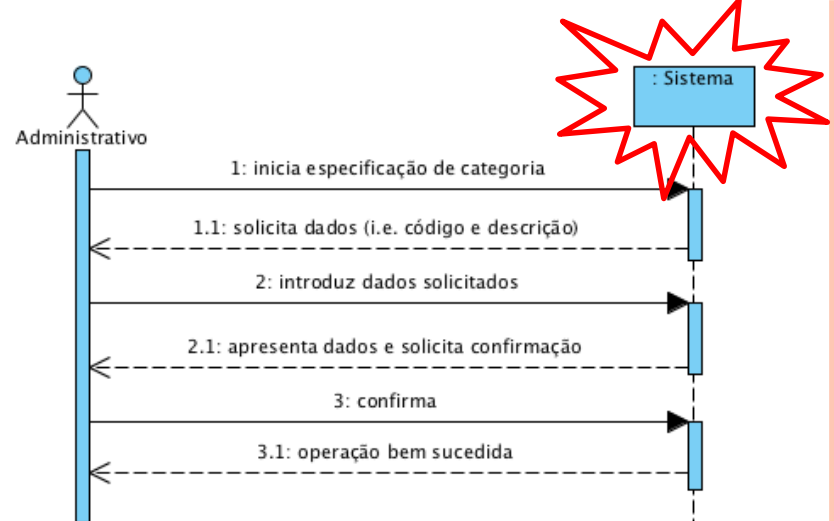
Aplicar padrões

- Abstract Factory
- Factory Method
- Builder

# CONTROLLER - EXEMPLO (DE OUTRO PROJETO)

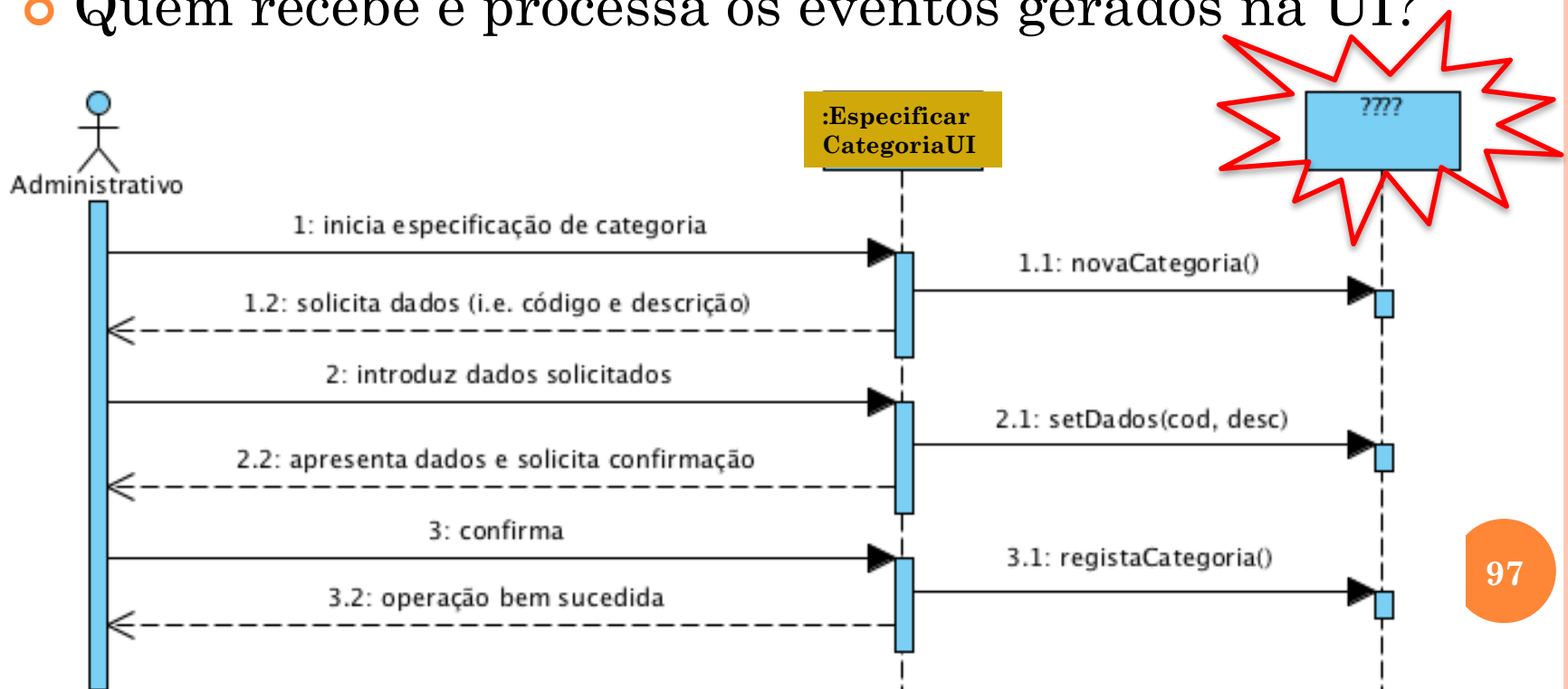
❖ Em que consiste o Sistema?

- ❖ User Interface (UI)
- ❖ ?



# CONTROLLER - EXEMPLO (DE OUTRO PROJETO)

- UI gera eventos
  - UI linha de comandos
  - UI gráfico
- Quem recebe e processa os eventos gerados na UI?

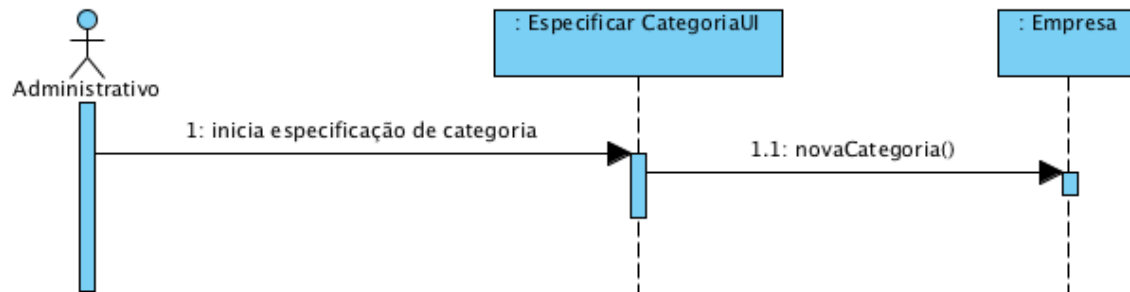


# CONTROLLER

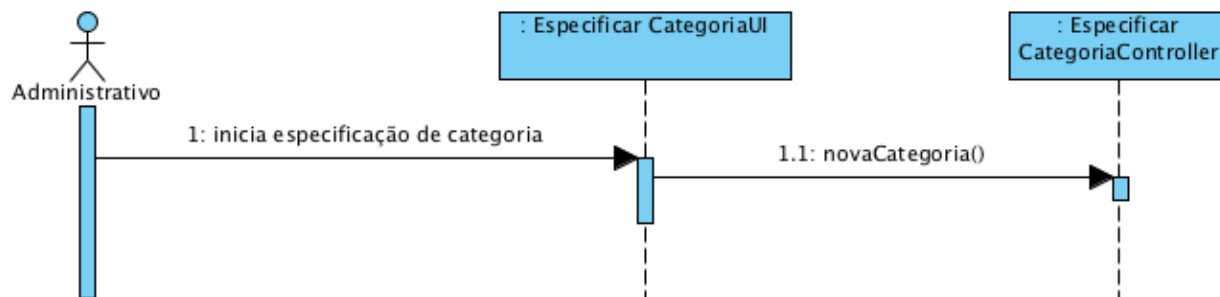
- ❖ **Problema:** Quem deve ser responsável por responder a um evento de entrada no sistema gerado na User Interface (UI)?
  - ❖ **Solução:** Atribuir a responsabilidade a uma das seguintes classes:
    - ❖ Aquela que representa globalmente o sistema, um dispositivo ou um sub-sistema (***facade controller***)
    - ❖ Aquela que representa um caso de uso no qual o evento ocorre:
      - ❖ *<UseCaseName>Handler*
      - ❖ *<UseCaseName>Session*
      - ❖ *<UseCaseName>Controller*
- ➔ **1 Controller por UC**

# CONTROLLER: FAÇADE VS. <UC>CONTROLLER

- *Empresa*: representa o sistema ou dispositivo



- *EspecificarCategoriaController*: representa um caso de uso no qual o evento ocorre



**Qual destas soluções possíveis devemos adotar?**



# CONTROLLER: FAÇADE VS. <UC>CONTROLLER

## ○ Como escolher?

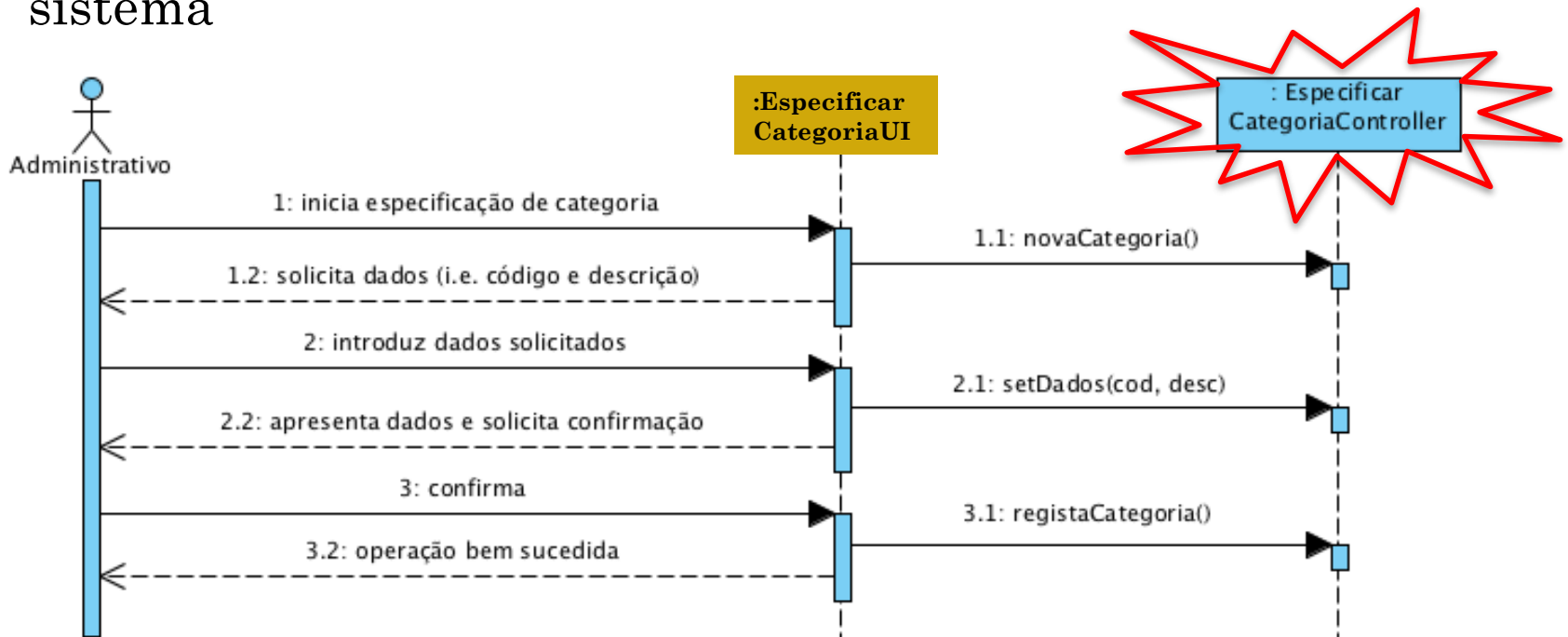
- Facade controller: quando existem poucos eventos de sistema
- Use-case controller: quando há muitos eventos e o facade controller ficaria muito extenso, com muitas responsabilidades

## ○ Vantagens:

- Aumenta a possibilidade de reutilização
- A camada de domínio pode ser usada com interfaces diferentes
- Controlo da sequência dos eventos, manutenção do estado de uma sessão

# CONTROLLER

- **Controller** é o primeiro objeto depois da camada UI que é responsável por receber ou tratar uma operação do sistema

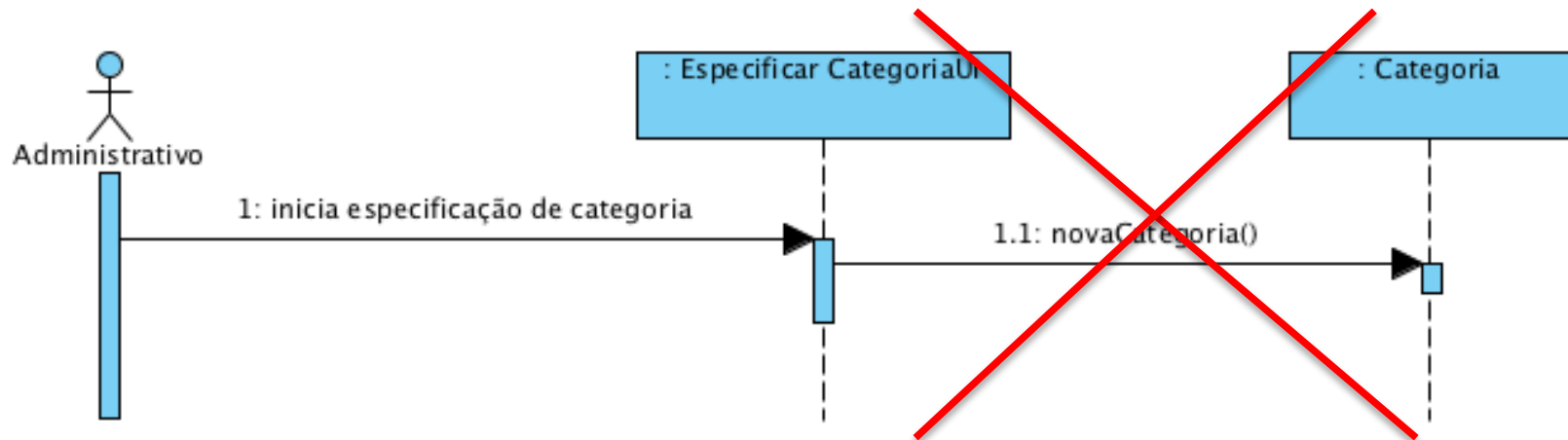


# CONTROLLER

- Fornece orientação sobre qual o objecto que deve tratar os eventos externos (e.g. UI)
- Se todos os eventos de um caso de uso são tratados na mesma classe, então:
  - É possível manter informação sobre o estado do caso de uso
  - É possível identificar erros na sequência de eventos
  - Pode/deve-se usar **Diagramas de Estado** para descrever estes eventos
- Normalmente um Controller deve
  - Coordenar/Controlar a atividade dum UC
  - Não deve ser o Controller a fazer o processamento →
  - Delegar (processamento) noutros objetos
- Operações do sistema devem ser tratadas na camada de domínio pelos controladores e não na camada UI por objetos GUI

# CONTROLLER

- A ligação directa da UI às classes do domínio deve ser evitada



# PURE FABRICATION

- **Problema:** Que objeto deve ter a responsabilidade, quando se quer cumprir a aplicação dos padrões *High Cohesion* e *Low Coupling*, mas as soluções propostas por outros padrões (e.g. *Expert*) não são apropriadas?

A atribuição de responsabilidades exclusivamente a objetos do domínio pode conduzir a uma baixa coesão e acoplamento.

- **Solução:** atribuir um conjunto coerente de responsabilidades a uma classe “artificial”, de conveniência – *Pure Fabrication* - que não representa um conceito do domínio.

# PURE FABRICATION

PersistentStorage
insert(Object) update(Object) ...

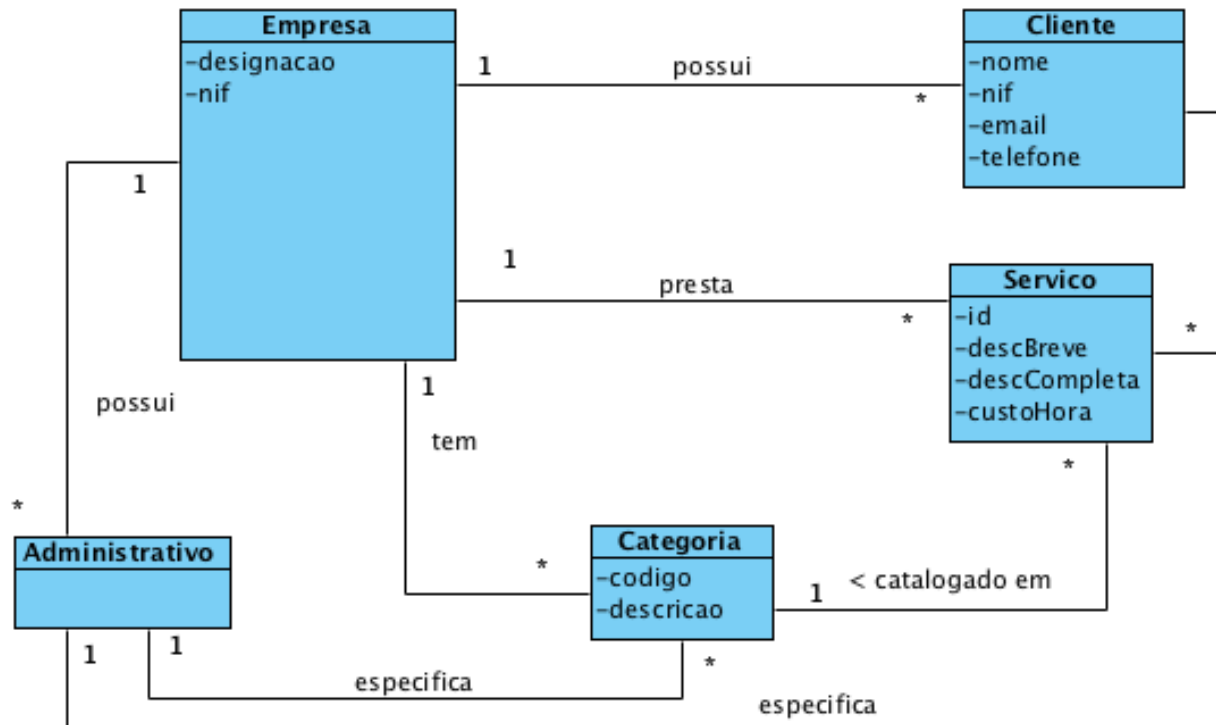
- Quem é o responsável pela gravação da informação numa *Categoria* na base de dados?
- Pela aplicação do padrão *Information Expert* poderia ser atribuída a responsabilidade à classe *Categoria*.  
*Porquê?*
- Esta atribuição conduzia a uma diminuição de coesão da classe *Categoria*, a um aumento de acoplamento entre classes e a uma diminuição da possibilidade de reutilização. *Porquê?*
- Uma possível solução é criar uma classe com apenas a responsabilidade de gravar objetos *PersistentStorage*

## EXEMPLO DE OUTRO PROJETO

Design – Use Case Realization

UC Especificar Serviço

# MODELO DE DOMINIO APLICAVEL





# RACIONAL (1/2)

Fluxo Principal	Questão: que classe	Resposta	Justificação
1. O administrativo inicia a especificação de um novo serviço.	... interage com o utilizador?	EspecificarServico oUI	PureFabrication
	...coordena o UC?	EspecificarServico oController	Controller
	..cria/instancia Servico?	Empresa	Creator (Regra 1)
2. O sistema solicita os dados necessários (i.e. identificador único, descrição breve e completa e o custo/hora).			
3. O administrativo introduz os dados solicitados.	... guarda os dados introduzidos?	Servico	Information Expert (IE) - instância criada no passo 1
4. O sistema mostra a lista de categorias existentes para que seja selecionada uma.	...conhece as categorias existentes a listar?	Empresa	IE: Empresa tem/agrega todas as Categoria

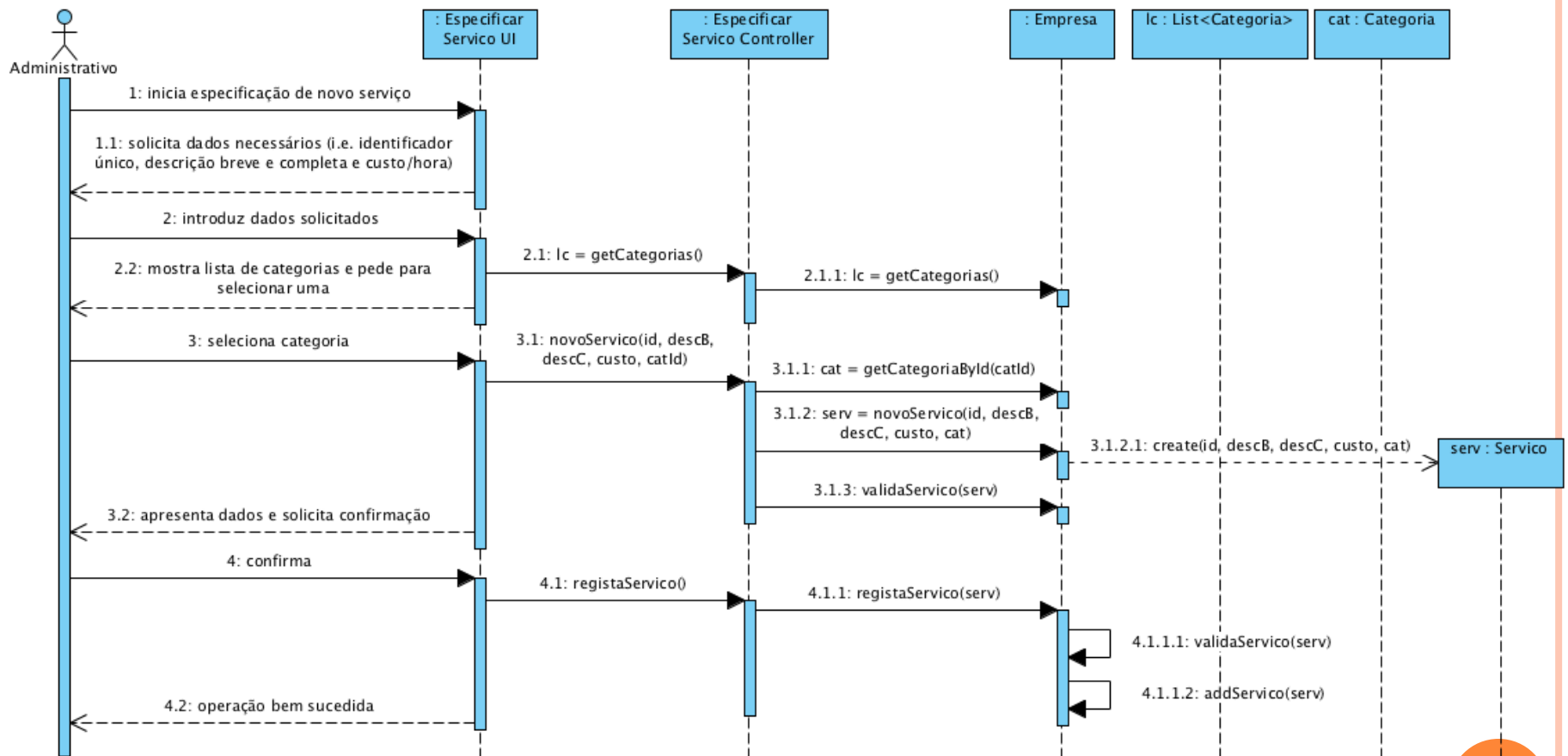
## RACIONAL (2/2)

Fluxo Principal	Questão: que classe	Resposta	Justificação
5. O administrativo seleciona a categoria em que pretende catalogar o serviço.	.. guarda a categoria selecionada?	Servico	IE: Servico catalogado numa Categoria - instância criada no passo 1
6. O sistema valida e apresenta os dados ao administrativo, pedindo que os confirme.	...valida os dados do Serviço (validação local)?	Servico	IE: Servico possui os seus próprios dados
	...valida os dados do Serviço (validação global)?	Empresa	IE: A Empresa contém/agrega Serviços
7. O administrativo confirma.			
8. O sistema regista os dados e informa o administrativo do sucesso da operação.	...guarda o Servico especificado/criado?	Empresa	IE. No MD a Empresa contém/agrega Servicos
	... notifica o utilizador?	EspecificarServicoUI	

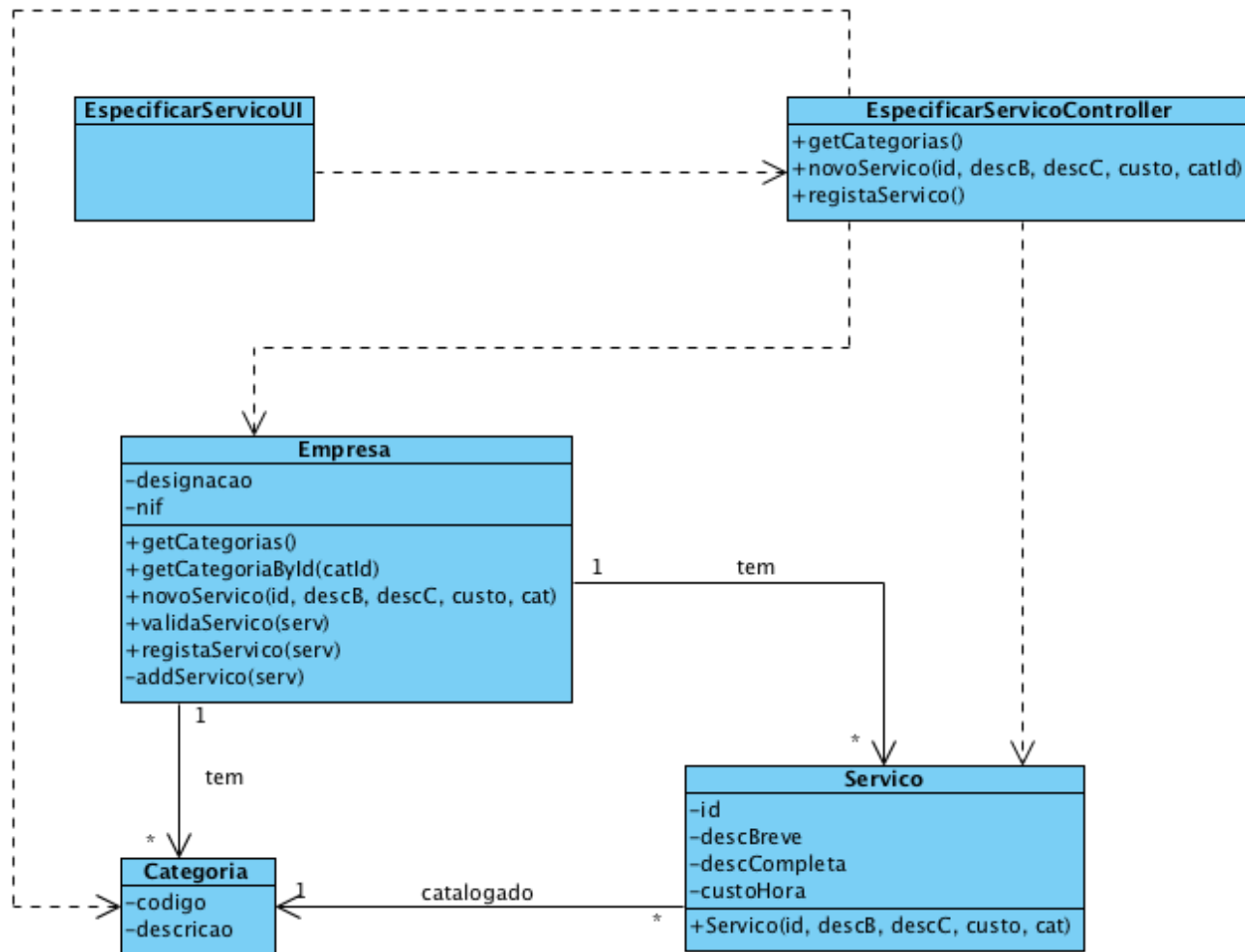
# SISTEMATIZAÇÃO DO RACIONAL

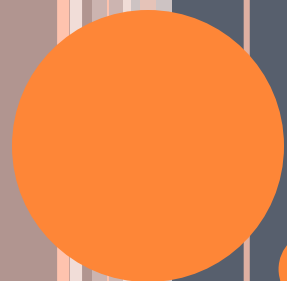
- Do racional resulta que as classes conceituais promovidas a classes de software são:
  - Empresa
  - Servico
  - Categoria
- Outras classes de software (i.e. Pure Fabrication) identificadas:
  - EspecificarServicoUI
  - EspecificarServicoController

# DIAGRAMA DE SEQUÊNCIA



# DIAGRAMA DE CLASSES





# SUMÁRIO DESIGN OO



# SUMÁRIO

- No contexto de Design OO
- Atribuição de responsabilidades deve obedecer a princípios/padrões consolidados
- GRASP é um guia de atribuição de responsabilidades OO, que promove:
  - Modularidade
  - Reutilização
  - Manutenção
- Princípios/Padrões GRASP são combinados entre si
- Muitos outros padrões baseiam-se nos GRASP

# REFERÊNCIAS E BIBLIOGRAFIA

- <http://www.dcs.bbk.ac.uk/~niki/SoftwareEngineering.htm>
- <http://www.cse.lehigh.edu/~glennb/oose/oose.htm>
- Rational Unified Process: Best Practices for Software Development Teams; Rational Software White Paper; TP026B, Ver 11/01.
- Rational Unified Process:  
<http://www.ts.mah.se/RUP/RationalUnifiedProcess/index.htm>
- Applying UML and Patterns; Craig Larman; (2nd ed.); 2002.