Introdução aos Sistemas de Informação

Sumário:

- Definição de Sistema de Informação.
- Problemas no desenvolvimento de Sistemas de Informação.
- Soluções:
  - Tecnologia;
  - Métodos de desenvolvimento.
  - Linguagens de modelação.

O que é um Sistema de Informação?

Sistema. *s. m.* Reunião de partes ligadas entre si, formando uma estrutura complexa. Conjunto de meios, processos destinados a produzir um resultado = Método.

Informação. *s. f.* Conjunto de conhecimentos reunidos sobre um determinado assunto; documentação.

Dicionário de Língua Portuguesa Contemporânea. Academia de Ciências de Lisboa.

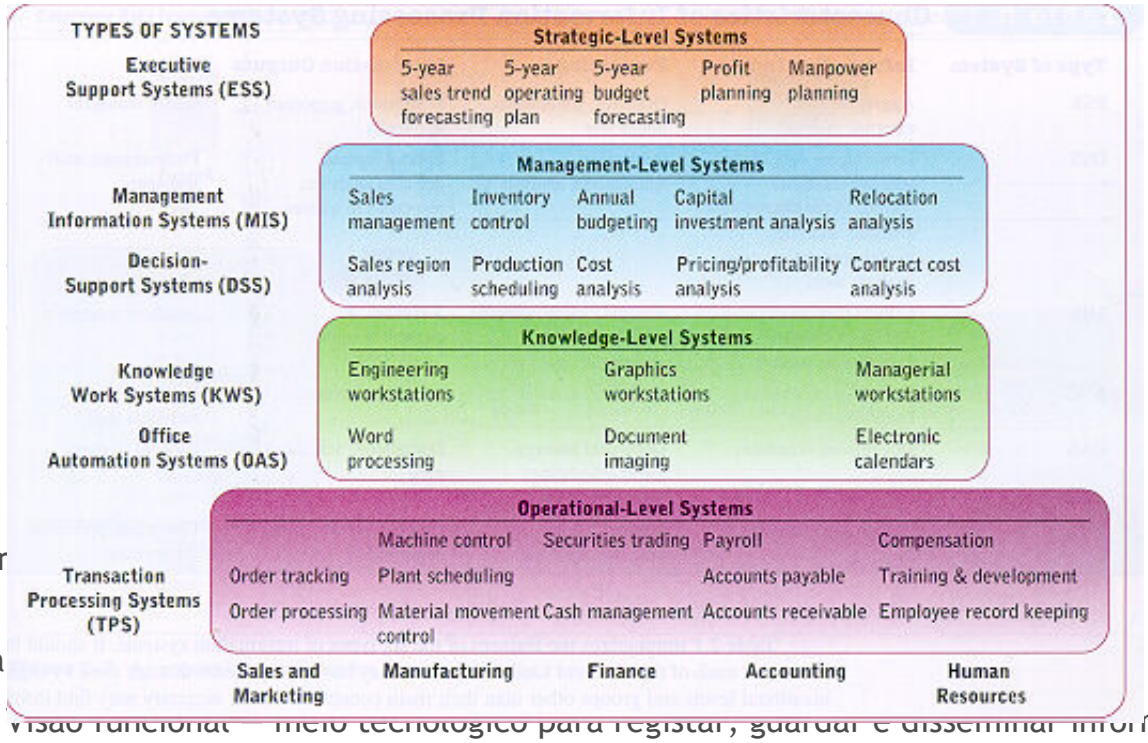
Sistema de Informação:

Fornece um meio para reunir informação de uma dada organização, fornecendo procedimentos para registo e disponibilização dessa informação, com o objectivo de auxiliar a organização nas suas actividades.

- Não é necessariamente um sistema software;
- Existem vários tipos de sistemas de informação...

Uma taxonomia para Sistemas de Informação

Diferentes tipos de Sistemas de Informação (SI):



Ater

Esta

- Visão funcional — meio tecnológico para registar, guardar e disseminar informação.
- Visão estrutural — colecção de pessoas, processos, dados, modelos tecnologia e linguagens.

Uma taxonomia para Sistemas de Informação (II)

Sistemas de Processamento de Transacções

- Recolhem e armazenam informação acerca de transacções (eventos de interesse);
- Controlam alguns aspectos relativos às transacções.

Sistemas de Automação de Escritórios

- Fornecem meios para o processamento de dados ao nível do indivíduo;
- Incluem o processamento de texto, folhas de cálculo, etc.

Sistemas de Trabalhos Baseados em Conhecimento

- Auxiliam na criação e integração de novos conhecimentos na organização.

Uma taxonomia para Sistemas de Informação (II)

Sistemas de Suporte à Decisão

- Auxiliam a tomada de decisões fornecendo informação, modelos ou ferramentas de análise.

Sistemas de Informação para a Gestão

- Analisam a informação produzida pelos Sistemas de Informação Transaccionais;
- Convertem informação sobre transacções em informação de controlo de desempenho e gestão da organização.

Sistemas de Informação para Executivos

- Evolução dos Sistemas de Informação para a Gestão vocacionada para executivos;
- Permitem análise da informação de forma simples e interactiva e a diferentes níveis de detalhe.

Uma taxonomia para Sistemas de Informação (IV)

Sejam de que tipo forem, os SI:

- existem para auxiliar a organização
  - ➨ mais um meio a juntar a tantos outros;
- devem ser concebidos em função das necessidades da organização
  - ➨ vão ser utilizados pela organização, não por quem os concebeu.

É necessário perceber a organização para conceber um bom SI.

- Quais as actividades da organização a suportar.
- Qual a informação relevante que flui na organização.
- Quais as tarefas das pessoas da organização.

Entender o problema antes de desenvolver a solução!

O que é um bom Sistema (de Informação)

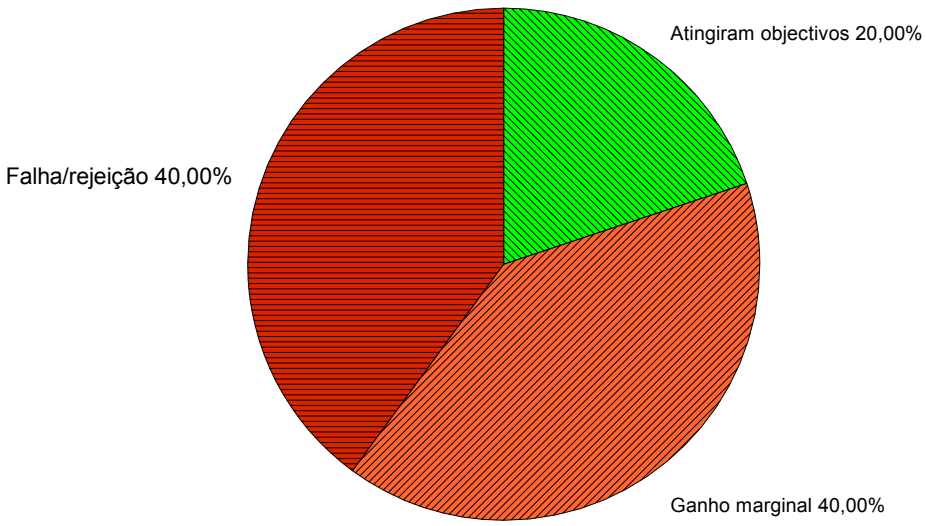
Aquele que satisfaz as necessidades dos seus utilizadores:

- **útil e utilizável** — bom software facilita a vida dos utilizadores - deve responder às necessidades reais dos utilizadores (*usabilidade!*);
- **confiável** — sem *bugs*!
- **flexível** — as necessidades dos utilizadores mudam, os *bugs* têm que ser corrigidos ✗ *bug* do ano 2k veio mostrar a falta de flexibilidade de muitos sistemas;
- **acessível** (financeiramente) — quer na compra quer na manutenção - fácil e rápido de desenvolver;
- **disponível** — se não está disponível nada mais interessa! - está disponível a tempo e horas? está disponível na plataforma tecnológica pretendida?

Como vai o desenvolvimento de software?

Estatísticas sobre desenvolvimento de Software

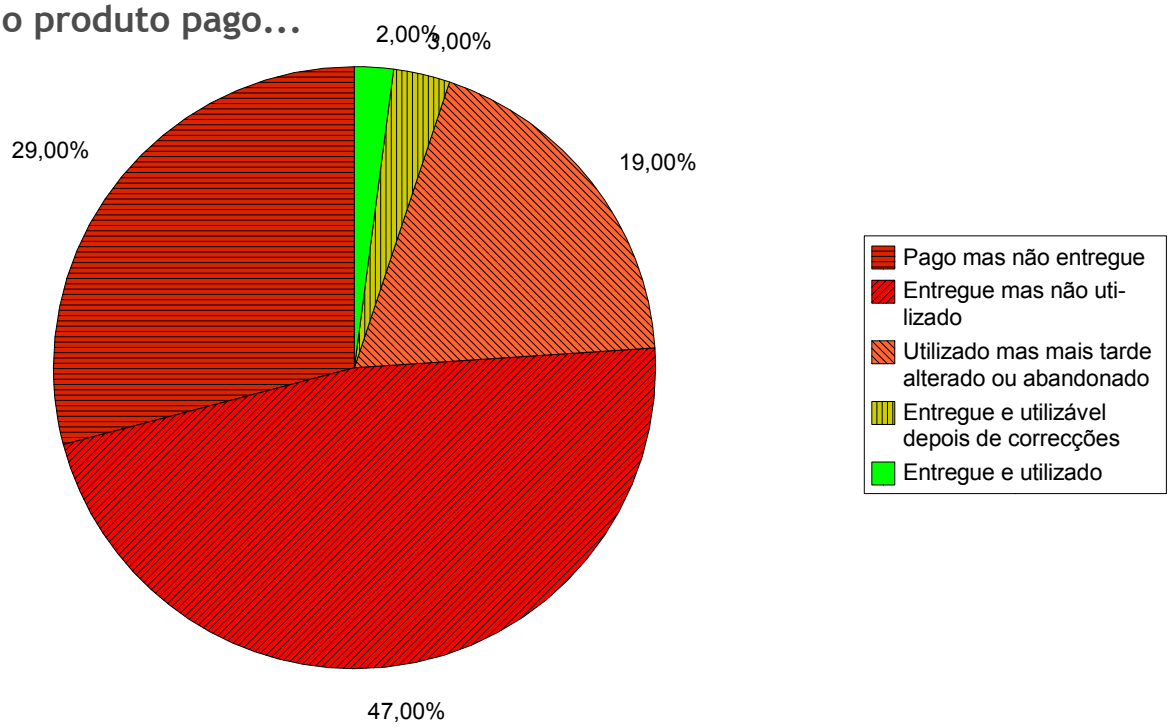
Sobre o produto entregue...



Fonte: Eason, 1988

Estatísticas sobre desenvolvimento de Software (II)

Sobre o produto pago...



- Mais de 75% do software pago não chegou a ser utilizado!
- Apenas 5% do software pago foi utilizado continuadamente (deste, 3% necessitou de correcções).

Fonte: GAO, 1992

Estatísticas sobre desenvolvimento de Software (III)

Sobre o processo...

Inquérito realizado em 1994 a 352 companhias (Standish Group):

- 56% de todos os *bugs* pode ser atribuídos a erros cometidos durante a fase de análise (i.e., não se esteve a construir o sistema certo!)
- 31% de todos os processos de desenvolvimento de software são cancelados antes de estarem terminados.
- 53% dos projectos custam 189% do estimado.
- 9% dos projectos de grandes companhias respeitam os prazos e o orçamento.
- 16% dos projectos de pequenas companhias respeitam os prazos e o orçamento.

Mais alguns dados sobre grandes projectos (>50,000 linhas de código):

- produtividade média está abaixo das 10 linhas de código por dia;
- em média, encontram-se 60 erros por cada 10,000 linhas de código;
- custo de manter o software ultrapassa o dobro do custo de desenvolvimento.

Estatísticas sobre desenvolvimento de Software (IV)

Tipos de erro:

- Erro devido a causas físicas — o sistema físico de suporte falha;
- Erro de software — os nossos conhecidos *bugs*;
- Erro humano — o operador do sistema utiliza-o de forma errada.

Resultados de alguns estudos:

- 60%-90% de todas as falhas são atribuíveis a erro humano (Hollnagel, 1993) .
- 92% das fatalidades consideradas num estudo entre 1979 e 1992 podiam ser atribuídas a problemas na interacção humano-computador, apenas 4% a causas físicas e 3% a erros de software (MacKenzie, 1994).

Mas...

Erro de qual humano?!

Do humano que utiliza o sistema, ou do humano que o desenhou/implementou?

Estatísticas sobre desenvolvimento de Software (V)

Exemplos

Alguns exemplos de sistemas com problemas atribuíveis ao software:

- Sonda Mariner I, Julho de 1962

  Deveria ter voado até Vénus. Apenas quatro minutos após o lançamento despenhou-se no mar. Descobriu-se depois que um operador de negação lógica tinha sido omitido por acidente no código do programa responsável por controlar os foguetes...

- Therac-25, finais dos anos 80

  Máquina de Raios-X totalmente controlado por software. Diversos problemas provocaram a administração de radiação excessiva a vários doentes.

- Aeroporto Internacional de Denver, início dos anos 90

  Sistema de tratamento de bagagem envolvendo mais de 300 computadores. O projecto excedeu os prazos de tal forma que obrigou ao adiamento da abertura do aeroporto (16 meses). Foi necessário mais 50% do orçamento inicial para o pôr a funcionar.

- Ariane 5, Junho de 1996

  Explodiu no voo inaugural devido a uma série de falhas no software de navegação Em circunstâncias específicas era efectuada uma divisão por zero. O sistema de segurança consistia em ter redundância: em caso de erro os dados eram processados por outro programa (uma cópia do primeiro! — abordagem adequada para hardware, mas não para software).

Estatísticas sobre desenvolvimento de Software (VI)

Em conclusão:

Problemas com o desenvolvimento de software:

- Atrasos na entrega.
- Incumprimento dos orçamentos.
- Falha na identificação e satisfação das necessidades dos clientes.
- Produtos entregues com falhas.

Algumas causas para o problema:

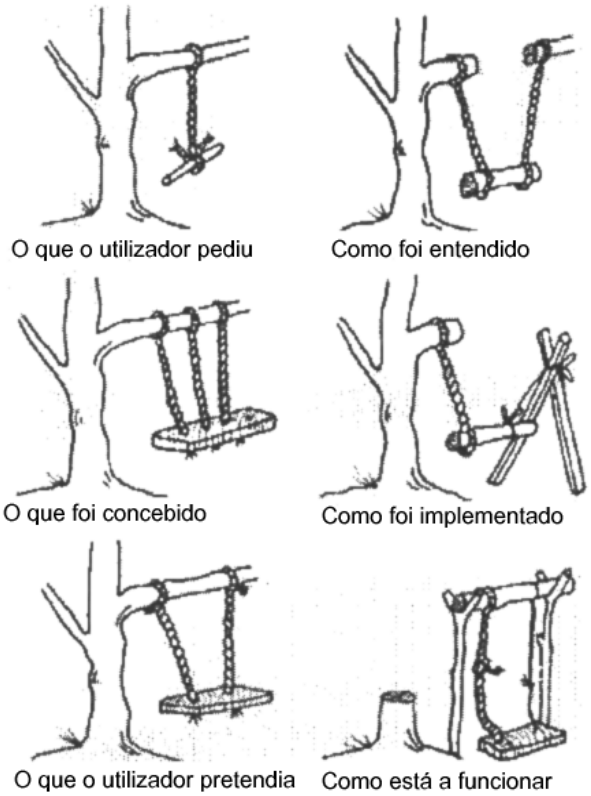
- Complexidade dos sistemas tem vindo a aumentar

  Quanto mais o hardware evolui, mais somos *tentados a atacar* problemas cada vez mais complexos.
- Condições de utilização do software são cada vez menos controladas

  Cada vez se torna mais difícil prever como/onde o software será utilizado.

Desenvolvimento de Software (Riscos)

Desenvolver um bom sistema não é tarefa trivial



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

Desenvolvimento de Software (Riscos) (II)

Riscos associados aos requisitos

É necessário comunicar com os peritos da organização para:

- compreender que tarefas o sistema deve suportar;
- compreender como o sistema *encaixa* nas actividades da organização.

Um dos maiores desafios é construir o sistema *certo*.

Riscos Tecnológicos

- Qual a tecnologia mais apropriada?
- Como controlar a complexidade?

É necessário validar as soluções tecnológicas o mais cedo possível.

Riscos de Competência

- É necessário saber-se o que se está a fazer (obviamente?).
- Exemplo de OO: fácil de aprender/difícil de dominar.

Riscos Políticos

- Por muito bom que seja o SI só terá sucesso se tiver o apoio das *pessoas certas*.

Respostas I - Tecnologia

- Primeiras abordagens à *Crise do Software* preocupavam-se mais com a produtividade do que com a qualidade.
- As tecnologias de programação têm vindo a tornar-se cada vez mais sofisticadas (tanto aos níveis dos paradigmas como das ferramentas).

Paradigmas de Programação

O modo como estruturamos o código tem vindo a evoluir como resposta ao aumento da complexidade do software:

- Programação estruturada (70's)
  Estruturar o código para controlar complexidade.
- Programação modular
  Estruturar o código, mas também os dados.
- Programação orientada aos objectos (80's)
  Aumenta o poder expressivo na estruturação de dados/código.
- Programação orientada aos aspectos (90's)
  Estruturação passa a ser feita, não em termos de objectos, mas em termos de *aspectos de interesse*.

Respostas I – Tecnologia (II)

Orientação aos Objectos

- *Mundo* visto como sendo composto por objectos com identidade própria e capacidade de interacção uns com os outros
- Vantagens reclamadas pela orientação aos objectos:
  - uma formal *natural* e compreensível de pensar/programar;
  - uma forma robusta e estável de programar;
  - facilita a manutenção dos sistemas e aumenta a produtividade
- Quatro aspectos fundamentais da orientação aos objectos:
  - Identidade (dos objectos);
  - Classificação (objecto como instância da sua classe);
  - Polimorfismo (operação+classe=método);
  - Herança. (super/sub-classes)

Respostas I – Tecnologia (III)

Identidade

- cada objecto tem identidade própria;
- podem existir objectos iguais; continuam, no entanto, a serem distintos (cf. gémeos).

Classificação

- os objectos agrupam-se em classes;
- todos os objectos de uma mesma classe possuem o mesmo conjunto de propriedades (atributos), o mesmo comportamento (operações) e as mesmas relações com outros objectos.

Polimorfismo

- a mesma operação pode ter comportamentos diferentes em objectos de classes diferentes;
- é uma forma de abstracção.

Herança (Especialização/Generalização)

- classes são organizadas de acordo com as suas similitude e diferenças;
- facilita a reutilização.

Respostas I – Tecnologia (IV)

Abstracção

- prestar atenção aos aspectos essenciais, ignorando os detalhes irrelevantes;
- nível de abstracção? demasiada abstracção, crie-se uma sub-classe (especializa-se); demasiado detalhe, crie-se uma super-classe (generaliza-se);
- O desenvolvimento passa a "*middle-out*".

Encapsulamento

- dados e operações correspondentes são agrupados numa única entidade.

Information hiding (ocultação de informação)

- torna-se possível esconder o estado (dados) dos objectos;
- acesso aos dados passa a ter que ser efectuado (de forma mais controlada) através das operações.

Representation hiding (ocultação da representação)

- ao ocultarmos os dados, passa também a ficar oculta a forma como estes estão representados.

Respostas I – Tecnologia (V)

Ferramentas

Ambientes de Desenvolvimento Integrado (IDEs) cada vez mais sofisticados procuram facilitar a tarefa de programação (e também de análise).

<u>Ferramentas para o UML</u>

- Rational Rose — inicialmente da Rational, agora da IBM;
- Together — inicialmente da TogetherSoft, agora da Borland;
- Poseidon — www.gentleware.com (versão *community edition*);
- Visual Paradigm — www.visual-paradigm.com (versão *community edition*);
- ArgoUML — argouml.tigris.org (open source /BSD)
- plugins para NetBeans, Eclipse, Jbuilder, etc.
- etc., etc., etc.

Respostas II – Métodos de Desenvolvimento

Mas...

- A tecnologia só por si não resolve os problemas (e estes têm vindo a aumentar!)
- A tecnologia é apenas uma ferramenta, é necessário saber como utilizá-la
- Hoje em dia a *Crise do Software* tem muito a ver com a qualidade do produto final.
- São necessários métodos de desenvolvimento que garantam produtividade e qualidade.

método

do Lat. methodu < Gr. méthodos, caminho para chegar a um fim

s. m., processo racional que se segue para chegar a um fim; modo ordenado de proceder; processo; ordem; conjunto de procedimentos técnicos e científicos;

(http://www.priberam.pt/dlpo/)

Respostas II – Métodos de Desenvolvimento (II)

Processo de desenvolvimento

Um processo define *quem* está a fazer *o quê*, *quando* e como tendo em vista atingir um dado objectivo (Jacobson et al. 99)

Um processo:

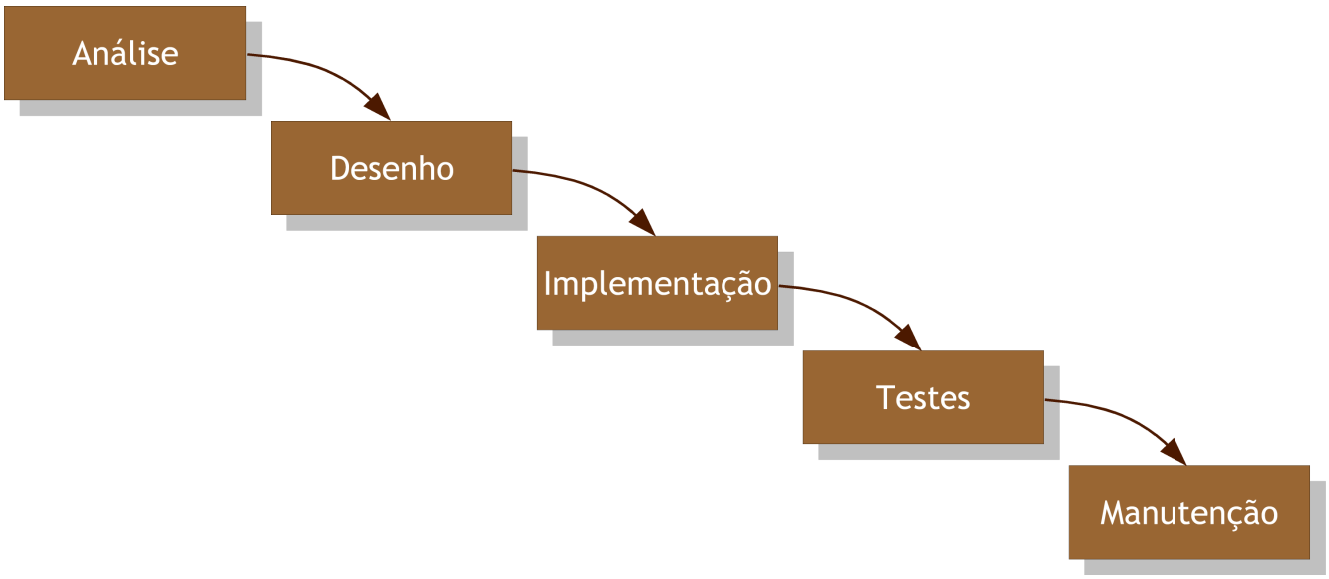
- Identifica um conjunto de regras que definem como o desenvolvimento de um sistema deve ser efectuado.
- Inclui, normalmente, uma descrição dos documentos que devem ser produzidos e em que ordem.
- Pode incluir indicação da linguagem (de modelação) a ser utilizada para a produção dos documentos.

Método de desenvolvimento:

Processo de desenvolvimento + Linguagem de modelação

Respostas II - Métodos de Desenvolvimento (III)

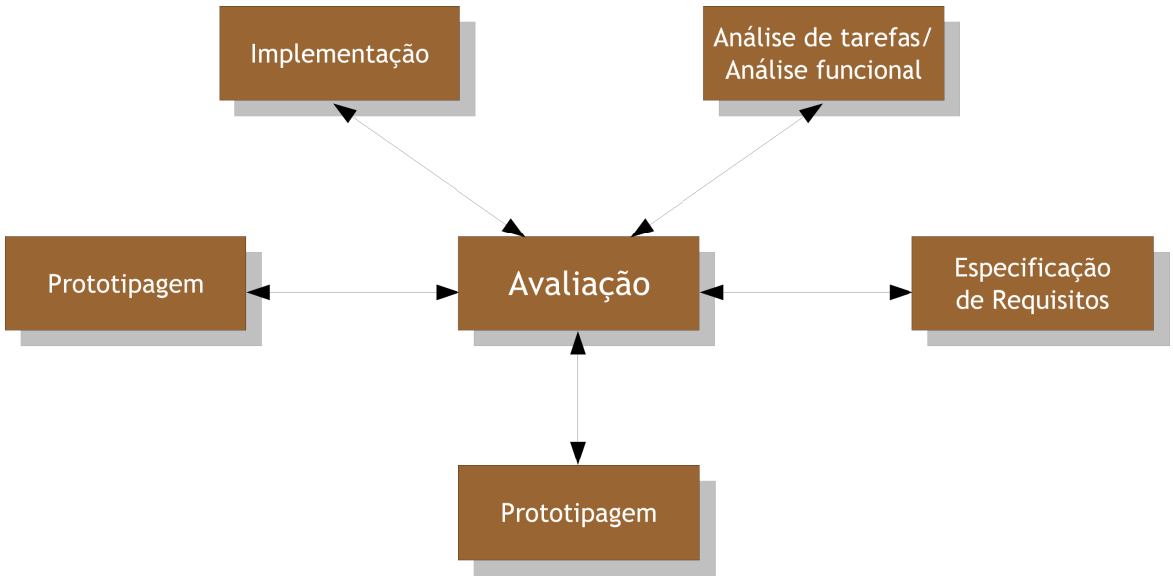
Waterfall



- Este tipo de processo define uma série de etapas executadas sequencialmente.
- Assume que é sempre possível tomar as decisões mais correctas — irrealista!

Respostas II - Métodos de Desenvolvimento (IV)

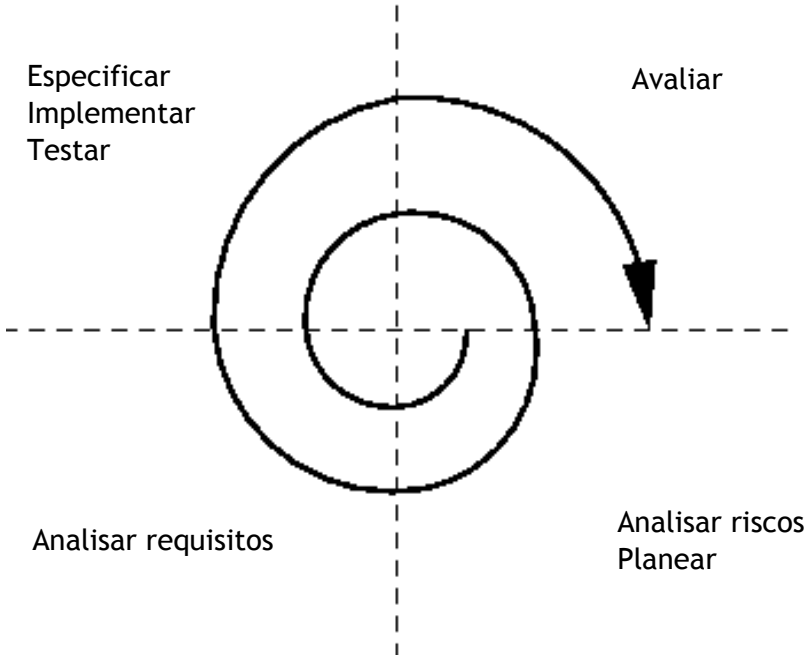
Estrela



- Neste tipo de processo coloca-se em destaque a necessidade de avaliar o sistema em todas as fases de desenvolvimento.

Respostas II – Métodos de Desenvolvimento (V)

Espiral



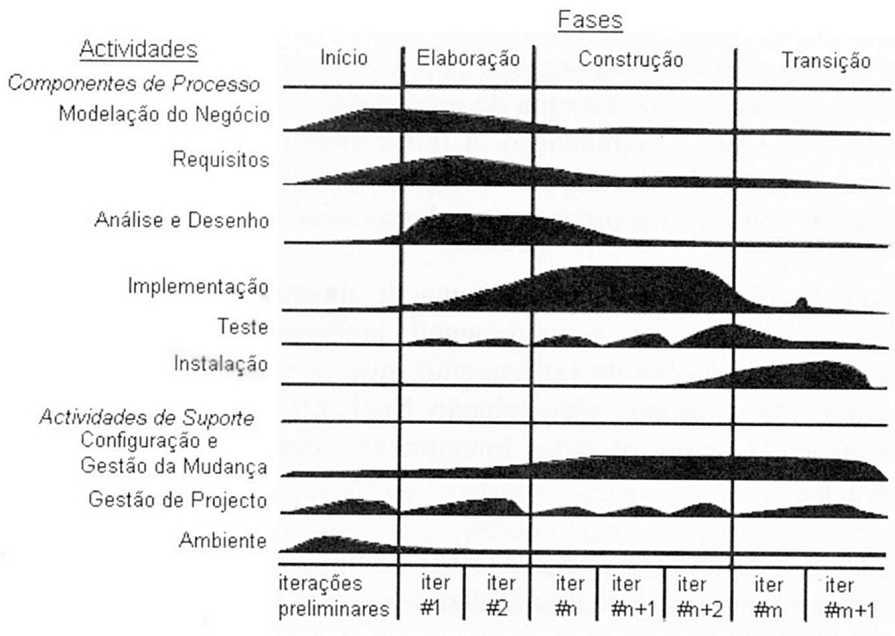
- Neste tipo de processo reconhece-se a necessidade de iterar para controlar riscos.

Respostas II – Métodos de Desenvolvimento (VI)

(Rational) Unified (software development) Process

Um processo:

- guiado por casos de uso (use cases);
- centrado na arquitectura do sistema a desenvolver;
- iterativo e incremental:
  - Início;
  - Elaboração;
  - Construção;
  - Transição.

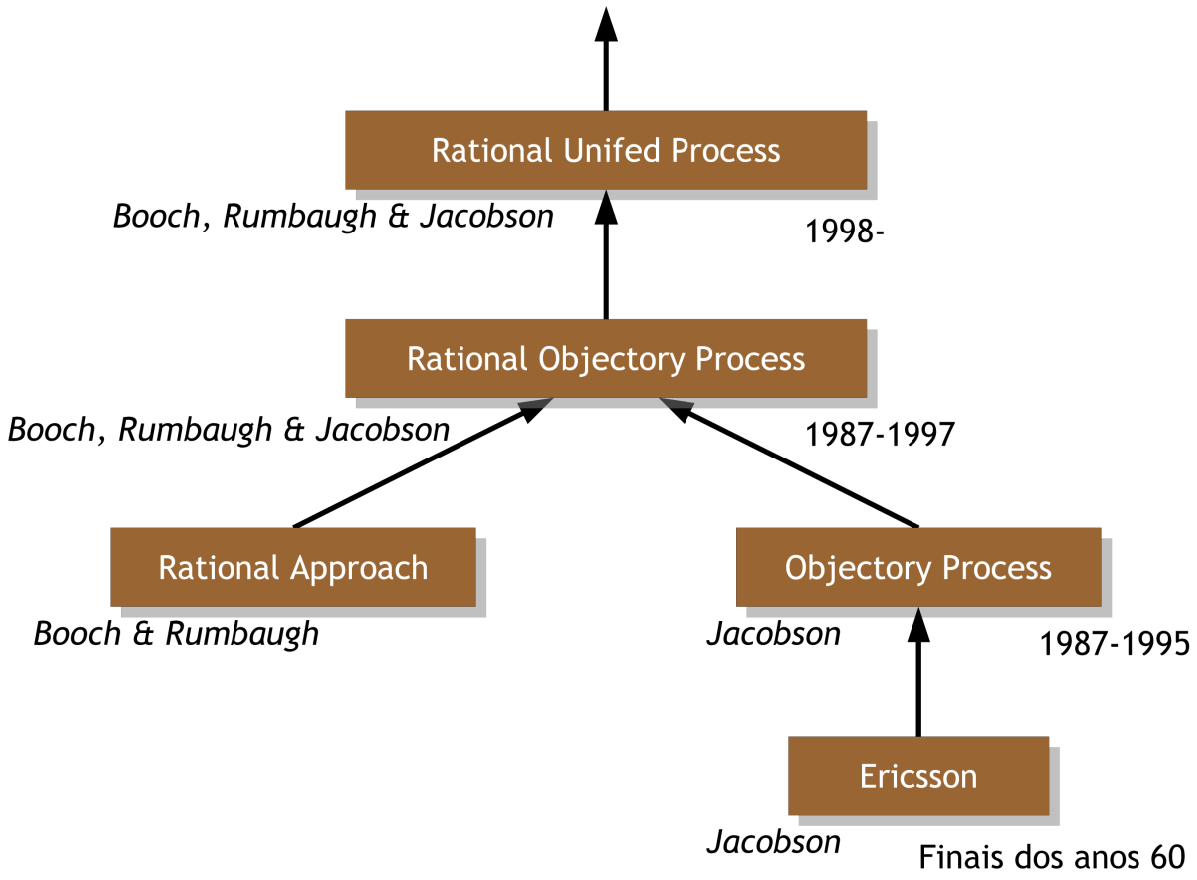


Unified Software Development Process

Sumário:

- Breve história do Unified Process
- O Unified Process
- O ciclo de vida do Unified Process
- O RUP (Rational Unifed Process)

Breve História do Unified Process



O Unified Process

O Unified Process é:

- Uma *framework* para o processo de desenvolvimento, genérica e adaptável.

O Unifed Process fomenta:

- O desenvolvimento baseado em componentes.

O Unified Process utiliza:

- O UML como ferramenta de modelação durante todas as fases do processo de desenvolvimento.

Três ideias chave

- Desenvolvimento guiado por *Use Cases* (Casos de Uso).
- Desenvolvimento centrado na arquitectura.
- Desenvolvimento iterativo e incremental.

Desenvolvimento guidado por Use Cases

- Um *use case* representa uma interacção entre o sistema e um humano ou outro sistema;
- O modelo de *use cases* é utilizado para:
  - guiar a concepção do sistema (captura de requisitos funcionais);
  - guiar a implementação do sistema (implementação de um sistema que satisfaça os requisitos);
  - guiar o processo de testes (testar que os requisitos são satisfeitos).

Desenvolvimento centrado na arquitectura

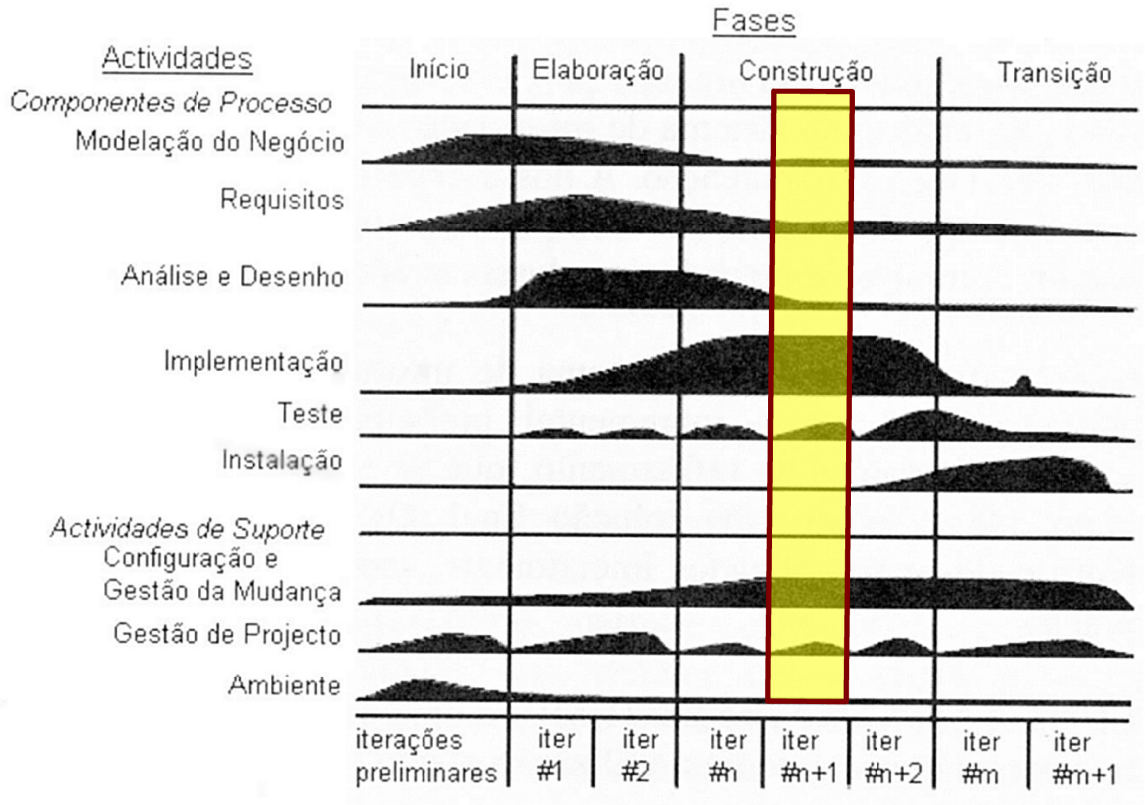
- O modelo de *use cases* descreve a função do sistema, o modelo da arquitectura descreve a forma;
- O modelo arquitectural permite tomar decisões sobre:
  - O estilo de arquitectura a adoptar;
  - Quais os componentes do sistema e quais as suas interfaces;
  - A composição de elementos estruturais e comportamentais.

Desenvolvimento de Sistemas de Informação - LESI/LMCC
José Creissac Campos

Desenvolvimento iterativo e incremental

- Permite dividir o desenvolvimento em “pedaços geríveis";
- Em cada iteração:
  - identificar e especificar *use cases* relevantes;
  - criar uma arquitectura que os suporte;
  - implementar a arquitectura utilizando componentes;
  - verificar que os *use cases* são satisfeitos.
- Selecção de uma iteração:
  - grupo de *use cases* que extendam a funcionalidade;
  - aspectos de maior risco.

Ciclo de vida do Unified Process



Fases do Unified Process (I)

Início

- Identificar o problema.
- Definir âmbito e natureza do projecto.
- Fazer estudo de viabilidade.

Resultado da fase: decisão de avançar com o projecto.

Elaboração (Análise / Concepção Lógica)

- Identificar o que vai ser construído (quais os requisitos?).
- Identificar como vai ser construído (qual a arquitectura?).
- Definir tecnologia a utilizar.

Resultado da fase: uma arquitectura geral (conceptual) do sistema.

Fases do Unifed Process (II)

Construção (Concepção Física/Implementação)

- Processo iterativo e incremental.
- Em cada iteração tratar um (conjunto de) *Use Case*:
  análise / especificação / codificação / teste / integração

Resultado da fase: um sistema de informação!

Transição

- Realização dos acertos finais na instalação do sistema.
- Optimização, formação.

Resultado da fase: um sistema instalado e 100% funcional (espera-se!).

O RUP (Rational Unifed Process)

- O RUP é uma concretização do Unifid Process.
- O RUP fornece:
  - ferramentas de gestão do processo de desenvolvimento segundo a *framework* definida pelo Unified Process (ver página 51);
  - ferramentas para a modelação e desenvolvimento baseadas no UML;
  - uma base de conhecimento (knowledge base).

- Na verdade as coisas passaram-se um pouco ao contrário...

Apresentação do UML

Sumário:

- UML
- Vantagens da utilização de modelos
- Problemas com a utilização de modelos
- Breve história do UML
- Diagramas do UML

UML – Unifiel Modelling Language

(Booch, Jacobson & Rumbaugh)

- o UML foi pensado para o **desenvolvimento de sistemas orientados aos objectos**, mas é independente das linguagens de programação a utilizar
  - permite explorar o paradigma OO (cf. riscos tecnológicos)
- o UML possibilita o **trabalho a diferentes níveis de abstracção**
  - facilita comunicação e análise (cf. riscos de requisitos)
- o UML não é uma linguagem, mas **uma família de linguagens gráficas** para modelar e construir sistemas software
  - inclui modelos para as diferentes fases do desenvolvimento
- o UML **não é um processo de desenvolvimento** de software, mas pode ser utilizado com diferentes processos
- O UML é um ***standard*** mantido pelo OMG (Object Management Group)
- O UML é suportado por **ferramentas**
  - *Rational Rose (IBM), Together (Borland), Visual Paradigm, Poseidon, etc., etc.*

Vantagens da utilização de modelos

- Auxiliam a **compreender** a realidade.
  - Sendo **abstracções** da realidade, os modelos permitem descrever o que é considerado essencial num dado contexto, escondendo detalhes desnecessários/ irrelevantes nesse contexto.
- Ajudam a **comunicar** ideias de forma simplificada.
  - Sendo simplificações da realidade, permitem comunicar apenas os aspectos pretendidos.
- Ajudam a **documentar** as decisões tomadas durante o desenvolvimento.
  - Os modelos desenvolvidos constituem uma base documental para a descrição do processo de desenvolvimento – "*Thinking made public*".

- Tipos de Modelos:
  - Preditivos / Normativos / Descritivos

Problemas com a utilização de modelos

- Mais uma "linguagem" a aprender.
  Isso acarreta **custos**, quer monetários (para as organizações), quer cognitivos (para os indivíduos).
- Modelos apresentam uma **visão idealizada da realidade**.
  Existe o risco de durante o processo de modelação nos esquecermos que os modelos são representações da realidade e não a realidade.
  É necessário considerar se estamos a utilizar abstracções adequadas e a modelar todos os aspectos relevantes.
- A fase de modelação **atrasa a produção de código** (pseudo-problema!)
  Espera-se, no entanto, que o código produzido seja de melhor qualidade (assim como o próprio sistema desenvolvido).
  Uma abordagem iterativa e incremental soluciona este *problema*. Por outro lado, é já possível passar, de forma semi-automática, dos modelos para o código.
  Regra dos 5/6 – 1/6 (análise e concepção vs. Codificação)

Atenção à "*analysis paralysis*"!

Breve história do UML

- Anos 60
  - Simula 67 é a primeira linguagem orientada aos objectos;
- Anos 70
  - Aparece o Smalltalk;
- Anos 80
  - as linguagens orientadas aos objectos (OO) tornam-se utilizáveis: Smalltalk estabiliza; surgem o Objective C, C++, Eiffel, CLOS, etc.
  - Finais dos anos 80 – surgem as primeiras metodologias de modelação OO
- Anos 90
  - existem dezenas de metodologias de modelação OO: Shlaer/Mellor, Coad/Yourdon, Booch, OMT (Rumbaugh), OOSE (Jacobson), etc.
  - meados dos anos 90 – começam as tentativas de unificação dos métodos
  - 1994 – Rumbaugh junta-se a Booch na Rational Software Corporation
  - 1995 – Booch e Rumbaugh apresentam a versão 0.8 do Unified Method (viria depois a mudar de nome para Unified Modelling Language); Jacobson junta-se a Booch e Rumbaugh
  - 1996 – o OMG (Object Management Group) pede propostas para um standard de modelação OO
  - Setembro, 1997 – a Rational, em conjunto com outras empresas (HP, IBM, Oracle, SAP, Unisys, ...), submete o UML 1.0 ao OMG como proposta de standard (existiram outras propostas)
  - Novembro, 1997 – o UML é aprovado como standard OO pelo OMG; o OMG assume a responsabilidade do desenvolvimento do UML
- Anos 00
  - 2005 – o UML 1.4.2 é aceite como norma ISO (ISO/IEC 19501)
  - 2006 – o UML 2.0 está em desenvolvimento

www.omg.org

Diagramas do UML

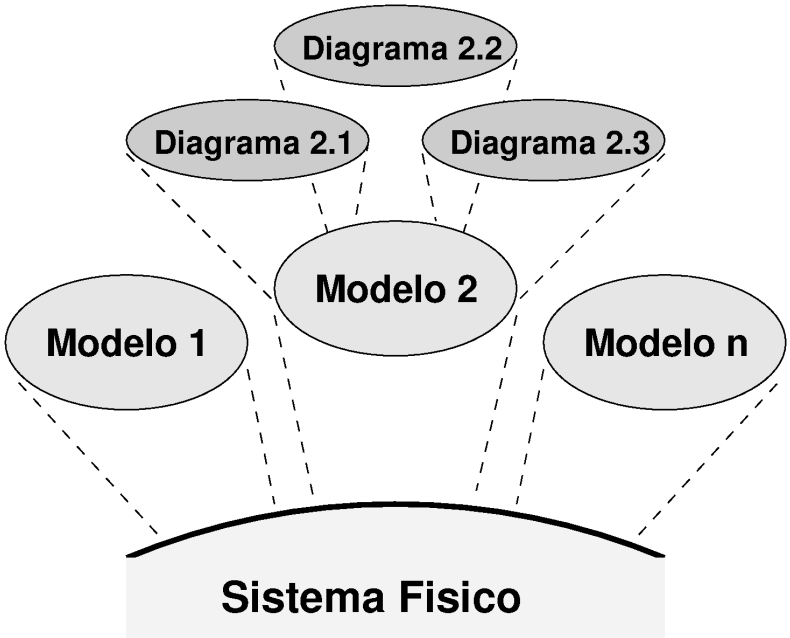
O UML define os seguintes oito diagramas base:

- **Diagramas de *Use Case*.**
- **Diagramas de Classe**
- Diagramas Comportamentais
  - Diagramas de Interacção
    - **Diagramas de Sequência**
    - **Diagramas de Comunicação** (aka Colaboração)
  - **Diagramas de máquinas de estado** (aka Statecharts)
  - **Diagramas de Actividade**
- Diagramas de implementação
  - **Diagramas de componentes**
  - **Diagramas de Instalação** (*Deployment*)

Desenvolvimento de Sistemas de Informação - LESI/LMCC
José Creissac Campos

- A **escolha dos modelos** a utilizar **tem** uma grande **impacto** na forma como um dado problema é abordado e, consequentemente, **na solução que se irá atingir**.
- A **Abstracção** (prestar atenção aos detalhes importantes e ignorar ou irrelevantes) **é** um factor **fundamental**.
- Assim:
  - Todo o sistema complexo deve ser abordado através de um **pequeno conjunto de** vistas/**modelos** tão independentes quanto possível;
  - Cada modelo pode ser expresso a diferentes níveis de detalhe;
  - Os melhores modelos são aqueles que têm **relação directa com a realidade**.
- Os oito tipos de diagramas identificados anteriormente procuram cobrir todas as necessidades de modelação que ocorram durante o desenvolvimento de software.

Os diagramas não são os modelos! / Os modelos não são o sistema!



Diagramas de *Use Case*

Sumário:

- Definição de requisitos
- Diagramas de Use Case I – conceitos base
- Diagramas de Use Case II – conceitos avançados
- Resumo
- Exercícios

Definição de requisitos

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 (II)

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.

Definição de *Use Case*

- Uma unidade coerente de funcionalidade – um serviço
- define um comportamento do sistema sem revelar a estrutura interna – apenas mostra a comunicação entre sistema e actores
- o conjunto de todos os *use case* define a funcionalidade do sistema
- deve incluir o comportamento normal, bem como variações (erros, etc.)
  - vamos definir o comportamento com texto estruturado;
  - 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 invocaçãoo 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).

O *use case* para fazer um telefonema:

Use Case: Fazer Telefonema
Pré-condição: Telefone ligado e em descanço
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-consição: Telefone ligado e em descanço

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 os Use Cases (serviços) necessários à correcta disponibilização da funcionalidade requerida pelo mesmo.

Identificação de *Use Cases (II)*

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

1. Enviar sumários via web
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!

*Durante o semestre o Prof. Faísca (1.) **foi enviando os sumários** com breves resumos da matéria leccionada, **via email**, para o sistema Fly2. Após o fim das aulas, o Prof. Faísca (2.) **utilizou a interface web** do sistema **para actualizar cada um dos sumários** com descrições mais completas das matérias leccionadas. Finda essa actualização, (3.) **imprimiu os sumários** e (4.) enviou-os à Secretaria.*

Diagramas de *Use Cases* – conceitos base

- 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*.

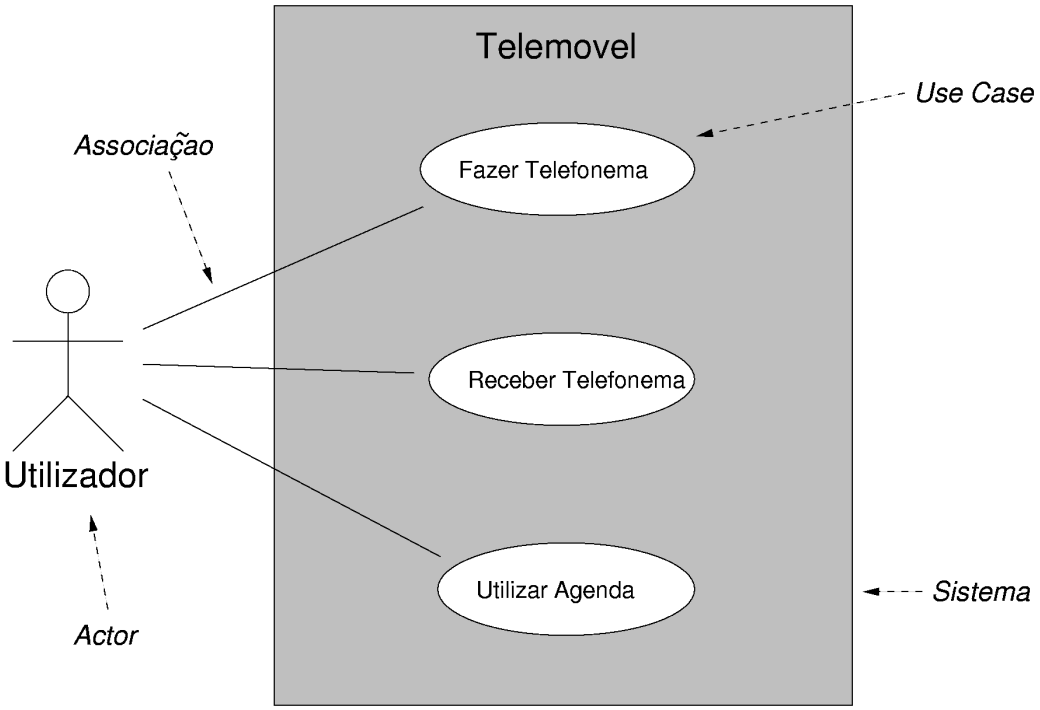
Black-box testing

- 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.

Exemplo de diagrama de *Use Cases*



Sistema

- define as fronteiras do sistema

Use Case (novamente)

- Uma unidade coerente de funcionalidade – um serviço
- define um comportamento do sistema sem revelar a estrutura interna – apenas mostra a comunicação entre sistema e actores
- o conjunto de todos os *use case* define a funcionalidade do sistema
- deve incluir o comportamento normal, bem como variações (erros, etc.)
  - vamos definir o comportamento com texto estruturado;
  - vamos também de□nir as pré-condições e pós-condiçõoes de cada use case (cf. *design by contract*).

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 define um conjunto de papeis que utilizadores do sistema podem 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*

Novamente a gestão de sumários

Sistema de gestão de sumários e presenças.

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

1. Identificar actores
2. Identi□car *use cases*
3. Identificar associações

Identificar actores

- Quem vai utilizar o sistema?
- Neste caso: Docente, Secretaria e Aluno

Identificar use cases

- Objectivos dos utilizadores/actores?
- Resposta a estimulos externos. não por se tratarem de requisitos do sistema.

Novamente a gestão de sumários (II)

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.

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 abragente.

Apenas as associações em que é o sistema externo o interessado

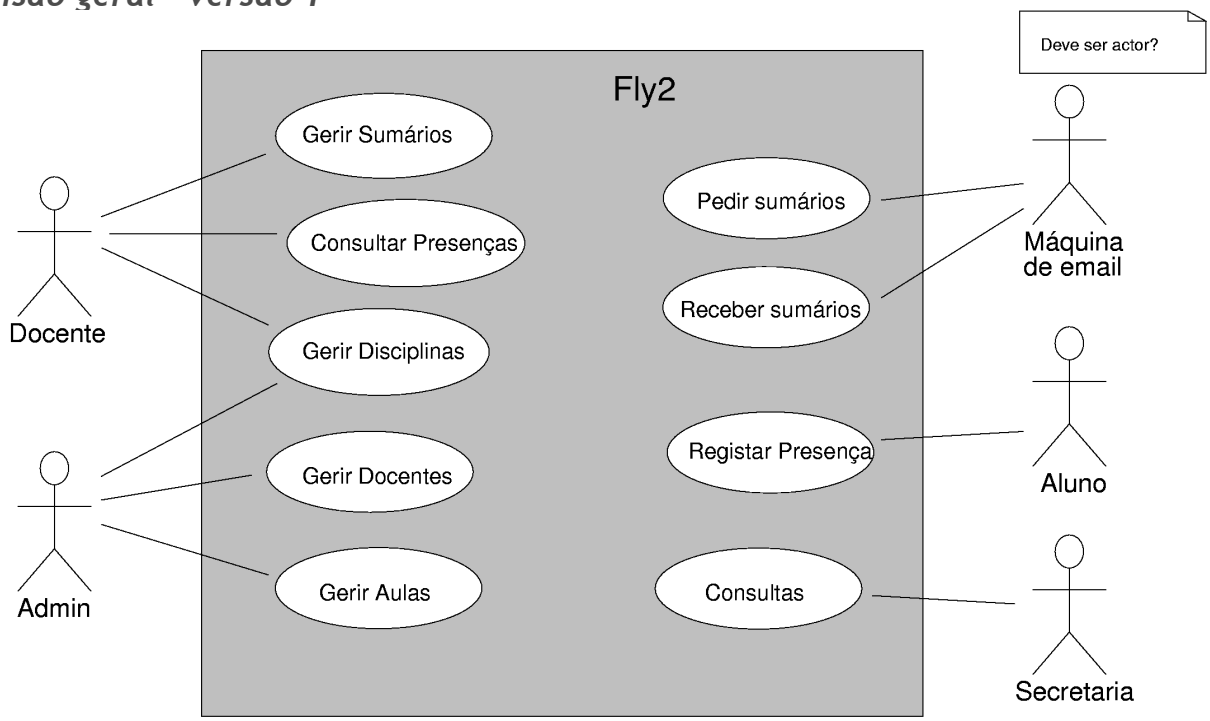
- Neste caso só são apresentados como actores os sistemas externos que necessitam de funcionalidade fornecida pelo sistema em análise.
- Usalmente esta é uma solução equilibrada.

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.

Novamente a gestão de sumários (III)

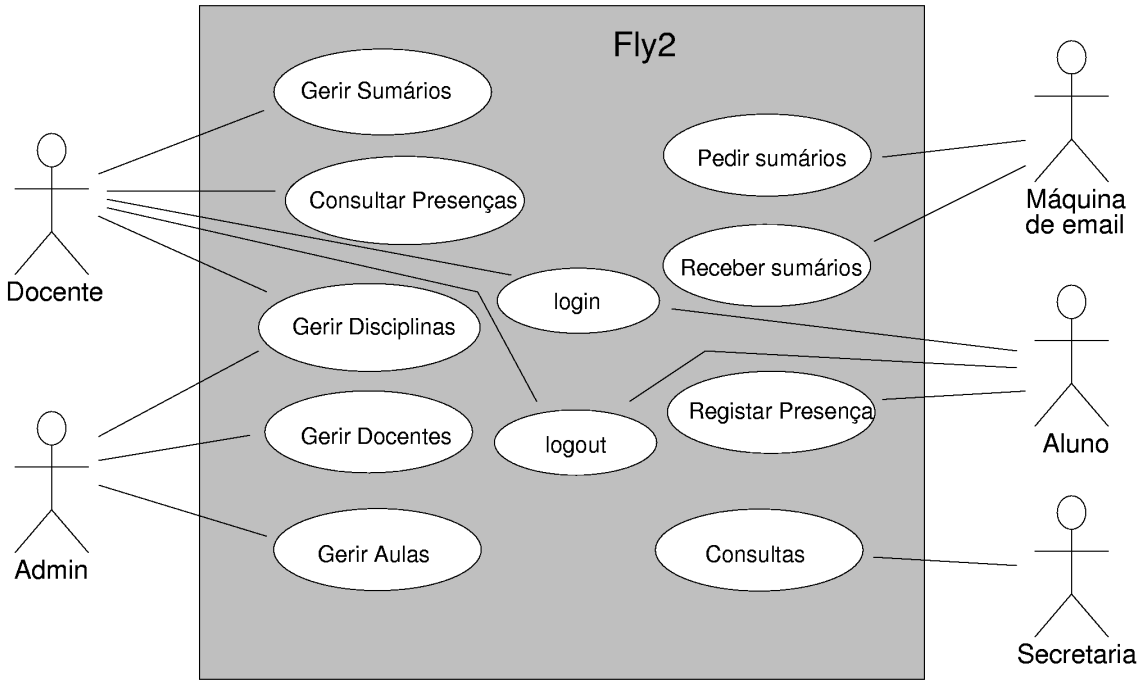
Visão geral - versão 1



- Falta mecanismo de autenticação!

Novamente a gestão de sumários (IV)

Visão geral - versão 2



- são adicionadas pré-condições nos *use case* Gerir Sumários, Gerir Presenças, Gerir Disciplinas e Registar Presença a exigir que tenha sido feito login.