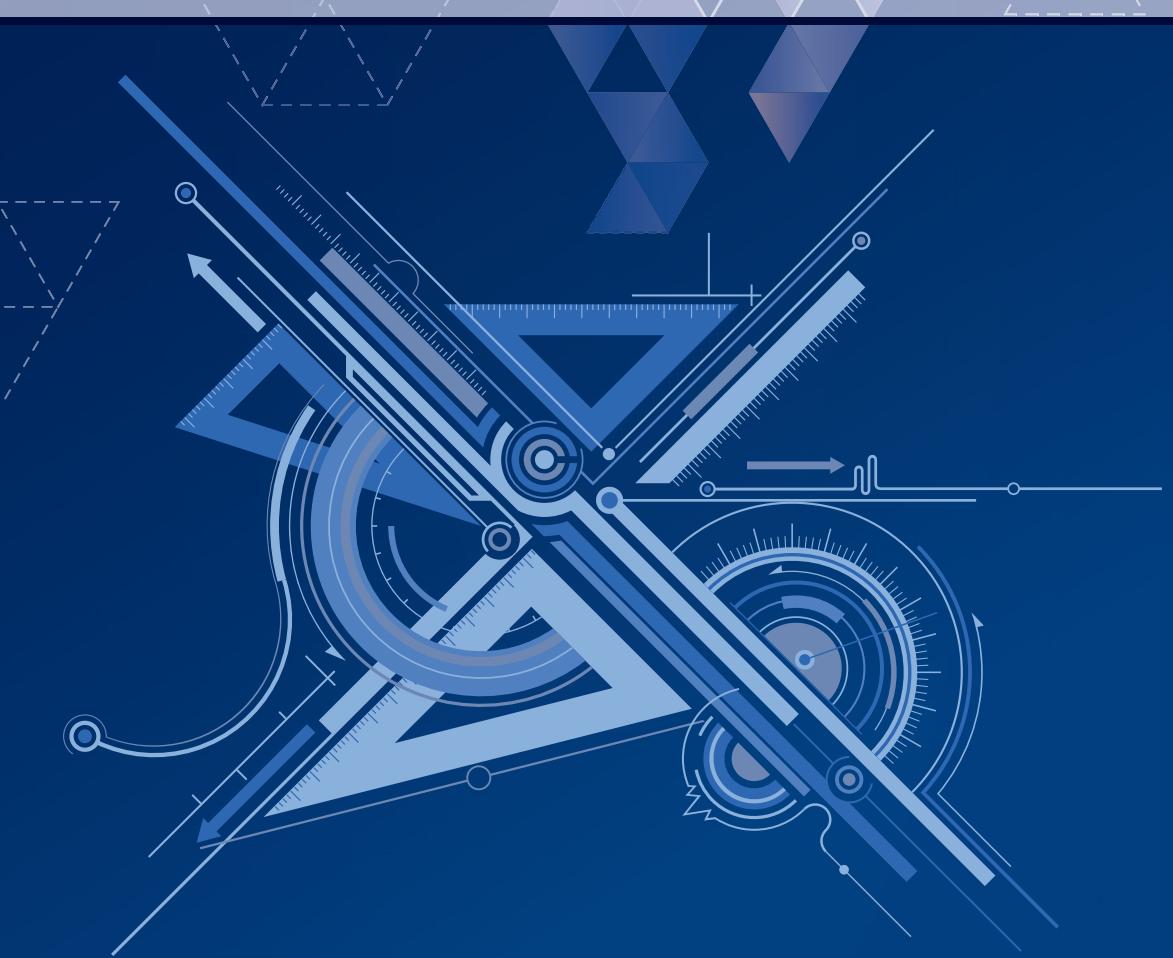




# PROCESSOS DE SOFTWARE



# **Processos de software**



# Processos de software

Polyanna Pacheco Gomes Fabris  
Luis Cláudio Perini

© 2014 by Editora e Distribuidora Educacional S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

*Diretor editorial e de conteúdo:* Roger Trimer

*Gerente de produção editorial:* Kelly Tavares

*Supervisora de produção editorial:* Silvana Afonso

*Coordenador de produção editorial:* Sérgio Nascimento

*Editor:* Casa de Ideias

*Editor assistente:* Marcos Guimarães

*Revisão:* Luciane Comide e Marina Sousa

*Capa:* Katia Megumi Higashi, Fernanda Caroline de Queiroz Costa

e Mariana Batista de Souza

*Diagramação:* Casa de Ideias

---

#### **Dados Internacionais de Catalogação na Publicação (CIP)**

---

Fabris, Polyanna Pacheco Gomes

F128p Processos de software / Polyanna Pacheco Gomes

Fabris, Luis Cláudio Perini. – Londrina: Editora e

Distribuidora Educacional S.A., 2014.

192 p.

ISBN 978-85-68075-89-0

1. Engenharia. 2. Negocio. 3. Gerenciamento. I. Perini ,  
Luis Cláudio. II. Título.

---

CDD 005

---

# Sumário

|  |           |
|--|-----------|
| <b>Unidade 1 — Processos de negócio .....</b>  | <b>1</b>  |
| <b>Seção 1 Visão geral sobre as áreas de negócio .....</b>   | <b>3</b>  |
| 1.1 O que é negócio?.....  | 3         |
| 1.2 E processos de negócio?.....   | 3         |
| <b>Seção 2 Noções sobre modelagem de processos de negócio.....</b>   | <b>6</b>  |
| 2.1 Introdução à modelagem de processos .....  | 6         |
| 2.2 O que são os modelos de processo?.....   | 7         |
| 2.3 Quais são os conteúdos de um modelo de processo?.....  | 7         |
| 2.4 Níveis de representação de processo.....   | 7         |
| 2.5 Componentes de processo e ferramentas .....  | 8         |
| 2.6 Arquitetura de processos e arquitetura de negócio .....  | 9         |
| 2.7 Notações de modelagem de processos.....  | 9         |
| <b>Seção 3 Introdução à modelagem organizacional e ao método Enterprise Knowledge Development (EKD).....</b> | <b>31</b> |
| 3.1 Modelos de EKD .....   | 33        |
| <b>Unidade 2 — Engenharia de software .....</b>  | <b>43</b> |
| <b>Seção 1 Natureza do software e da engenharia de software .....</b>  | <b>44</b> |
| 1.1 Evolução do software.....  | 50        |
| 1.2 Software.....  | 53        |
| 1.3 Engenharia de software .....   | 57        |
| 1.4 Crise do software .....  | 61        |
| 1.5 Causas da crise de software .....  | 61        |
| 1.6 Consequências da crise de software.....  | 62        |
| 1.7 Causas dos problemas associados à crise de software .....  | 63        |
| 1.8 Mitos relativos ao software.....   | 63        |

|  |     |
|--|-----|
| <b>Seção 2 Modelos de processos de software .....</b>                                    | 67  |
| 2.1 Modelos ágeis.....   | 69  |
| 2.2 Modelo evolucionário.....  | 77  |
| <b>Unidade 3 — Engenharia de requisitos e gerenciamento de projeto de software .....</b> | 93  |
| <b>    Seção 1 Requisitos de software .....</b>  | 95  |
| 1.1 Requisitos de software .....   | 95  |
| <b>    Seção 2 Engenharia de requisitos .....</b>  | 108 |
| 2.1 Engenharia de requisitos .....   | 108 |
| 2.2 Estudos de viabilidade .....   | 109 |
| 2.3 Elicitação e análise de requisitos .....   | 111 |
| 2.4 Validação de requisitos .....  | 117 |
| 2.5 Gerenciamento de requisitos .....  | 119 |
| <b>    Seção 3 Gerenciamento de projetos de software .....</b>                           | 123 |
| 3.1 Gerenciamento de projetos.....   | 123 |
| 3.2 Atividades de gerenciamento .....  | 124 |
| 3.3 Planejamento de projeto .....  | 125 |
| <b>Unidade 4 — Noções sobre modelagem de sistemas .</b>                                  | 143 |
| <b>    Seção 1 Introdução à modelagem estruturada de sistemas .....</b>                  | 145 |
| 1.1 Introdução .....   | 145 |
| 1.2 Introdução à modelagem estruturada de sistemas .....                                 | 145 |
| <b>    Seção 2 Introdução à modelagem orientada a objeto e para web .....</b>            | 149 |
| 2.1 Introdução .....   | 149 |
| 2.2 Introdução à modelagem orientada a objeto e para web.....                            | 149 |

# Processos de negócio

*Polyanna Pacheco Gomes Fabris*

**Objetivos de aprendizagem:** Nesta unidade, você será levado a compreender conceitos importantes sobre as áreas de negócio, noções sobre modelagem de processos de negócio, modelagem organizacional o método Enterprise Knowledge Development (EKD), bem como seus principais modelos.

## » Seção 1: Visão geral sobre as áreas de negócio

Caro aluno, nesta seção, vamos apresentar uma visão geral sobre as áreas de negócio, a definição do que é negócio e processos de negócio. E, para reforçar o conteúdo estudado, você terá atividades de aprendizagem.

## » Seção 2: Noções sobre modelagem de processos de negócio

Esta seção apresenta os conceitos presentes na modelagem de processos de negócio e alguns métodos que auxiliam nesta atividade. E, para reforçar o conteúdo estudado, você terá atividades de aprendizagem.

## » Seção 3: Introdução à modelagem organizacional e ao método Enterprise Knowledge Development (EKD)

Caro aluno, nesta seção, abordaremos um conteúdo sobre a modelagem organizacional e o método Enterprise Knowledge Development (EKD), que enfoca a importância de uma organização trabalhar de forma integrada e não setorizada, ou seja, todos vi-

sando ao crescimento organizacional. E assim como foi realizado nas seções anteriores teremos, ao final desta, atividades de aprendizagem para reforçar o conteúdo estudado.

---

# Introdução ao estudo

Caro aluno, esta unidade apresentará uma visão geral sobre as áreas de negócio, noções sobre modelagem de processos de negócio e como esses modelos podem contribuir para o entendimento dos negócios organizacionais, proporcionando um entendimento comum dentro da organização, possibilitando medição de possíveis problemas e também contribuindo para tomada de decisões.

Com base nesses processos definidos, os colaboradores de uma organização conhecem claramente seus papéis, bem como as atividades a serem desenvolvidas. A execução e a revisão desses processos é que possibilita o crescimento organizacional, em que a organização passa a depender de processos, e não de pessoas.

Ainda nesta unidade faremos uma introdução à modelagem organizacional e ao método Enterprise Knowledge Development (EKD) e seus modelos.

---

## Seção 1 Visão geral sobre as áreas de negócio

Caro aluno, em meio às frequentes variações de mercado, as organizações buscam soluções de integração dos seus processos para melhorar a interação com seus clientes internos ou externos e oferecer qualidade aos seus serviços prestados, resultando em padronização e eficiência na execução de seus processos de negócio. Esta seção apresenta uma visão detalhada sobre os processos de negócio.

### 1.1 O que é negócio?

Segundo ABPMP (2013), o termo “negócio” corresponde à interação de pessoas para executar um conjunto de atividades de entrega de valor para os clientes e gerar retorno às partes interessadas. “Negócio” compreende todos os tipos de organizações com ou sem fins lucrativos, públicas ou privadas, de qualquer porte e segmento de negócio.

### 1.2 E processos de negócio?

Segundo Moreira (2014), um processo é a introdução de insumos de entrada em um ambiente formado por procedimento, normas e regras, que, ao processarem tais insumos, resultam em produtos de saída aos clientes do processo. Além dos insumos de entrada e os produtos de saída, um processo é composto por mais dois elementos essenciais para sua execução: os recursos e o controle. Desta forma, pode-se afirmar que um processo típico é formado por quatro elementos:

- **Entradas:** são materiais ou informações que contribuirão para que o processo seja executado.
- **Saídas:** representa o resultado obtido com a execução do processo, podem ser produtos ou serviços.

— **Controle:** responsável pelo monitoramento e verificação do cumprimento dos métodos, diretrizes, objetivos, procedimentos e padrões na execução do processo.

— **Recurso:** representa a provisão de elementos que se fazem necessárias para a execução do processo, tais como: humanos, materiais, financeiros entre outros.

Portanto, o processo de negócio é uma associação de atividades e procedimentos executados por humanos ou máquinas para entregar valor para clientes ou apoiar/gerenciar outros processos.

Os processos de negócio podem ser classificados, segundo ABPMP (2013); Capote (2014):

- a) **Processos primários:** são os processos que ultrapassam qualquer fronteira funcional corporativa, representando as atividades essenciais que uma organização executa para cumprir sua missão. Suas características são: entrega de valor ao cliente, visão ponta a ponta e interfuncional, criação e percepção de valor, podem percorrer organizações funcionais, setores, departamentos ou outras organizações, e tendem a traduzir a cadeia de valor (agrupamento corporativo estruturado entre atividades primárias e atividades de suporte).
- b) **Processos secundários:** são os processos definidos formalmente na organização e que visam a dar suporte aos processos primários. Possuem como características a ausência de relacionamento direto com os clientes, e também, o vínculo à visão funcional tradicional, mas processos de suporte também podem ser interfuncionais. Possuem impacto direto na capacidade de realização e entrega dos processos primários.
- c) **Processos de gestão:** são processos estabelecidos formalmente e com o intuito de coordenar as atividades dos processos de apoio e dos processos primários. Deve buscar garantir que os processos atinjam suas metas operacionais, financeiras, regulatórias e legais.



#### *Para saber mais*

Nesta seção, foram abordados os conceitos de processos de negócio e as suas possíveis classificações. E agora, você sabe descrever o que é um processo de negócio? Para mais informações acesse: <<http://www.abpmp-br.org/>>.



## Atividades de aprendizagem

1. Assinale V para verdadeiro e F para falso.
  - ( ) Entradas representam o resultado obtido com a execução do processo, podem ser produtos ou serviços.
  - ( ) Saídas são materiais ou informações que contribuirão para que o processo seja executado.
  - ( ) Processos secundários são os processos definidos formalmente na organização e que visam dar suporte aos processos primários.
  - ( ) Processos primários são os processos que ultrapassam qualquer fronteira funcional corporativa, representando as atividades essenciais que uma organização executa para cumprir sua missão.
  - ( ) Controle é responsável pelo monitoramento e verificação do cumprimento dos métodos, diretrizes, objetivos, procedimentos e padrões na execução do processo e são definidos como saídas.
2. O que é processo de negócio e quais são suas classificações?



## Questões para reflexão

Com base no conteúdo estudado, pense em uma organização e defina para ela os quatro elementos típicos de um processo.

## Seção 2

# Noções sobre modelagem de processos de negócio

A modelagem de processos de negócio é vista como uma ferramenta fundamental, a modelagem de processos tornou-se uma das principais estratégias modernas de gestão. Organizações pequenas, médias e grandes utilizam esse mecanismo com resultados cada vez mais satisfatórios (CAMPOS, 2014a).

A modelagem de processos fornece mecanismos para constituir e aperfeiçoar a operação e controle das organizações, e também, para viabilizar o atingimento dos objetivos estratégicos dessas organizações (CAMPOS, 2014b).

## 2.1 Introdução à modelagem de processos

A modelagem de processos pode ser definida como um conjunto de atividades envolvidas na criação de representações de um processo de negócio existente ou proposto. Ela expressa uma perspectiva ponta a ponta de processos primários, de suporte e gerenciamento de uma organização (MPF, 2014). Requer um conjunto de habilidades e técnicas para permitir que os componentes de processos sejam compreendidos, comunicados e gerenciados.

O objetivo da modelagem é gerar uma representação do processo de maneira completa e precisa sobre seu funcionamento. Desta forma, o nível de detalhamento e o tipo específico de modelo são baseados no que é esperado da iniciativa de modelagem (ABMP, 2013).

A aplicação da modelagem de processos promove a identificação de todas as áreas envolvidas no negócio, de todos os passos necessários para a execução de um processo e documentação utilizada.

Veja, a seguir, alguns dos seus principais objetivos:

- └ Entendimento da estrutura e a dinâmica das áreas da organização.
- └ Entendimento dos problemas atuais da organização e identificar potenciais melhorias.
- └ Garantir que os usuários e engenheiros de software tenham entendimento comum da organização.
- └ Auxílio na identificação de competências (DEVMEDIA, 2014).
- └ Dentre os motivos para uma organização modelar seus processos estão (ELOGROUP, 2014):
  - └ Documentar claramente um processo existente.
  - └ Auxílio nos treinamentos por meio da definição das atribuições de cada ator presente no processo.
  - └ Capacidade de avaliação das normas e cumprimentos das obrigações.
  - └ Compreensão do comportamento do processo.
  - └ Base para análise na identificação de oportunidades de melhoria.

- └ Suporte ao design de um novo ou nova abordagem para um processo existente.
- └ Prover uma base para a comunicação e discussão organizacionais.
- └ Descrição de requisitos para uma nova operação da empresa.

## 2.2 O que são os modelos de processo?

Um modelo de processo é uma representação simplificada de uma coisa, um conceito ou uma atividade que suporta os seus estudos e desenhos. Os modelos podem ser matemáticos, gráficos, físicos ou narrativos na sua forma ou em alguma combinação de elementos. Possuem ampla série de aplicações, que incluem: organização (estruturação), heurística (descoberta, aprendizado), previsões (predições), medição (quantificação), explanação (ensino, demonstração), verificação (experimentação, validação) e controle (restrições, objetivos).

Podem conter um ou mais diagramas, informação sobre objetos no diagrama, informação sobre relacionamento entre objetos, informação sobre como objetos representados se comportam ou desempenham. Assim, podem ser expressos por meio de vários níveis de detalhe, desde uma visão contextual abstrata até uma visão detalhada, representando diversas perspectivas e propósitos (ABPMP, 2013; MPF, 2014).

## 2.3 Quais são os conteúdos de um modelo de processo?

Um modelo de processo possui ícones que representam as atividades, os eventos, as decisões, condições e outros elementos. Pode conter ilustrações sobre:

- └ Ícone (representa os elementos do processo).
- └ Relacionamentos entre os ícones.
- └ Relacionamentos entre os ícones e ambiente.
- └ Como os ícones se comportam ou o que executam (ABPMP, 2013).

## 2.4 Níveis de representação de processo

Diagrama, mapa e modelo são três níveis de “representação” de processos, e muitas vezes são utilizados de forma intercambiável ou como sinônimos. Contudo, eles se diferem em níveis de abstração, informação, utilidade, precisão, complexidade, padronização de elementos do fluxo, evolução e amadurecimento do desenho proposto (ABPMP, 2013; IPROCESS, 2014).



*Para saber mais*

Vamos ver a diferença entre diagrama, mapa e modelo de processos?

Acesse: <<http://blog.iprocess.com.br/2014/02/modelagem-de-processos-de-negocio-diferencias-entre-diagrama-mapa-e-modelo-de-processos/>> (IPROCESS, 2014).

### 2.4.1 Resumo da comparação entre diagrama, mapa e modelo de processo

Para facilitar seu entendimento dos níveis de representação do processo, observe a Tabela 1.1 a seguir.

**Tabela 1.1 Comparação entre níveis de representação de processos**

| Diagrama ou mapa de processo  | Modelo de processo  |
|---|---|
| Notação ambígua.  | Convenção padronizada da notação.   |
| Baixa precisão.   | Maior precisão e detalhamento, conforme necessidade.  |
| Menos detalhado.  | Mais detalhado.   |
| Ícones (representação de componentes do processo) vagamente definidos e sem padronização. | Ícones definidos e padronizados.  |
| Relacionamentos dos ícones retratados.  | Relacionamentos dos ícones definidos e explicados em notações, glossários do modelo de processos e especificação detalhada do processo. |
| Limitado a uma representação simples ou um contexto de alto nível.                        | Representa a complexidade adequada.   |
| Limitado a retratar um momento específico da realidade.                                   | Possibilidade de crescimento, evolução e amadurecimento.  |
| Gerado com ferramentas simples de diagramação.  | Gerado com ferramenta adequada ao objetivo.   |
| Simples de utilizar, mas não permite a navegação em um nível mais detalhado.              | Fornece a possibilidade de uma simulação manual ou automatizada do processo.  |
| Dificuldade de conexão com outros modelos existentes.                                     | Ligações verticais e horizontais, mostrando os relacionamentos entre processo e diferentes níveis de processo.                          |
| Utiliza estruturas comuns de gerenciamento de arquivos.                                   | Utiliza repositório de modelos.   |
| Adequado para certas capturas rápidas de ideias.  | Apropriado para qualquer nível de captura de processos, análise e desenho.  |
| Não é adequado para automatização de processos.   | Adequado para automatização de processos.   |

Fonte: ABPMP (2013).

## 2.5 Componentes de processo e ferramentas

Os componentes de processo especificam as propriedades, os comportamentos, o objetivo e outros elementos do processo. Algumas ferramentas de modelagem podem ser usadas com a finalidade de capturar e catalogar os componentes de processo e informações associadas de forma que sejam organizadas, analisadas e o portfólio de processos seja gerenciado.

Dentre os componentes de processo e informações que os modelos de processo podem capturar, podemos citar: entradas e saídas, padrões de chegada e distribuições,

eventos e resultados, custos, valor agregado, regras de entrada, papéis e organizações, regras de saída, dados e informação, regras de decisão, regras de junção, enfileiramento, tempo de trabalho e manuseio, tempo de transmissão, agrupamento, tempo de espera e número de pessoas com disponibilidade para executar tarefas (ABPMP, 2013).

## 2.6 Arquitetura de processos e arquitetura de negócio

Durante a modelagem de processo, as pessoas envolvidas podem se questionar sobre o que diferencia a arquitetura de processos da arquitetura de negócio.

A Tabela 1.2, a seguir, apresenta um comparativo entre essas duas arquiteturas (ABPMP, 2013).

**Tabela 1.2 Arquitetura de processos e arquitetura de negócios**

| Arquitetura de processos   | Arquitetura de negócios   |
|--|---|
| Especificação da estrutura geral de um sistema de processos. Pode ser um conceito aplicável a diversos campos tais como informática, gestão de processos de negócio, gestão estratégica etc. (WIKIPÉDIA, 2014). A arquitetura de processos lida com o COMO do negócio e definem com um entregável. | É modelo conceitual e lida com O QUE no negócio.                      |
| Foco nas atividades físicas e seu gerenciamento.   | Relaciona as atividades necessárias e está preocupada com a eficácia. |

Fonte: Do autor (2014).

Contudo, podemos afirmar que essas disciplinas se complementam e visam a entregar melhoria de negócio e ambas podem fazê-lo. Não são similares e não há concorrência entre elas.

## 2.7 Notações de modelagem de processos

Segundo o ABPMP (2013), “notação” pode ser definida como um conjunto padronizado de símbolos e regras que determinam o significado desses símbolos. Existem muitos padrões de notação de modelagem, porém, a escolha de um padrão que siga normas e convenções oferece vantagens, tais como:

- └ Conjunto de símbolos, linguagem e técnicas comuns, possibilitando um “idioma” comum entre os envolvidos.
- └ Consistência em forma e significado dos modelos de processos.
- └ Possibilidade de importar e exportar os modelos entre ferramentas.
- └ Gerar aplicações a partir dos modelos.

Dentre as notações de modelagem de processos usualmente encontrada, podemos citar BPMN (Business Process Model and Notation), fluxograma, ARIS, UML (Unified Modeling Language), IDEF (Integrated Definition Language) entre outras.

## 2.7.1 BPMN (Business Process Model and Notation)

Segundo BPMN (2014) e TUDO SOBRE BPMN (2013), a meta primária da notação BPMN é fornecer uma notação que possa ser entendida por todos os usuários do negócio (analistas de processos/negócios que criam a versão inicial do processo), desenvolvedores responsáveis por implementar a tecnologia que auxiliará os processos e, finalmente, as pessoas que irão gerenciar e monitorar os processos.

Esta notação cria uma ponte padrão para a análise de aderência entre o desenho do processo de negócio e a implementação do processo. Outra meta é garantir que a linguagem XML concebida para a execução de processos de negócio possa ser visualizada com uma notação orientada ao negócio.

A intenção do BPMN é padronizar a notação para modelagem de processos de negócio das diferentes notações de modelagem e pontos de vista, proporcionando um meio simples de comunicação de informações a outros utilizadores de negócio, implementadores de processo, clientes e fornecedores.

A interoperação dos processos de negócio, em um nível humano, pode ser resolvida com a padronização, e é essa notação da modelagem que fornece um diagrama o qual é desenhado para o uso das pessoas que implementam e gerenciam os processos de negócio. Também fornece um mapeamento para uma linguagem de execução para sistemas, resultando na geração de um mecanismo de visualização padrão para processos de negócio, definido em uma linguagem de execução otimizada de processos de negócio.

Essa notação proverá às organizações uma capacidade de compreensão de seus procedimentos internos do negócio em uma notação gráfica e fornecerá para as organizações a habilidade de comunicar esses procedimentos em um padrão uniforme (BPMN, 2014; TUDO SOBRE BPMN, 2014).

Apomando os conceitos de modelagem, a notação BPMN exclui de seu escopo outros tipos de modelagens feitas por organizações sem o foco no negócio, tais como: estruturas organizacionais e recursos, modelos de funções de departamentos, modelos de dados e informações, modelos de estratégia empresarial, modelos regras de negócios (ELOGROUP, 2014).

A notação visa a cobrir muitos tipos de modelagem e a permitir a criação de processos de negócio fim a fim. Existem três tipos de modelos básicos para a modelagem BPMN; veja a seguir.

### 2.7.1.1 *Processos de negócio privado (internos)*

Específicos de uma organização e são os tipos de processos que geralmente são chamados de *workflow* ou processos BPM (Figura 1.1). Para sua modelagem, serão usadas raias de executores ou suporte. O fluxo de atividades está contido apenas em uma única raia mas, tratando-se de fluxo de informação, permite a travessia entre raias, de maneira que demonstre as interações existentes entre processos privados de negócio diferentes (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

**Figura 1.1 Exemplo de processo de negócio privado**

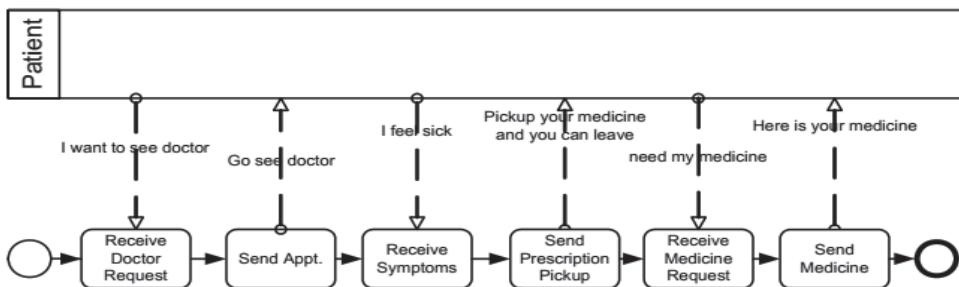


Fonte: BPMN (2014).

### 2.7.1.2 Processos abstratos (públicos)

Representam a interação entre um processo de negócio privado e outro processo ou participante. Apenas as atividades que são utilizadas para a comunicação externa ao processo de negócio privado, mas o mecanismo adequado de fluxo de controle, estão incluídos nos processos abstratos. Demais processos internos não são apresentados nos processos abstratos. Desta forma, este tipo mostra ao mundo exterior a sequência de mensagens que são necessárias na interação com outro processo ou entidade externa. A Figura 1.2, a seguir, apresenta um exemplo de processo abstrato.

**Figura 1.2 Exemplo de processo de negócio abstrato**

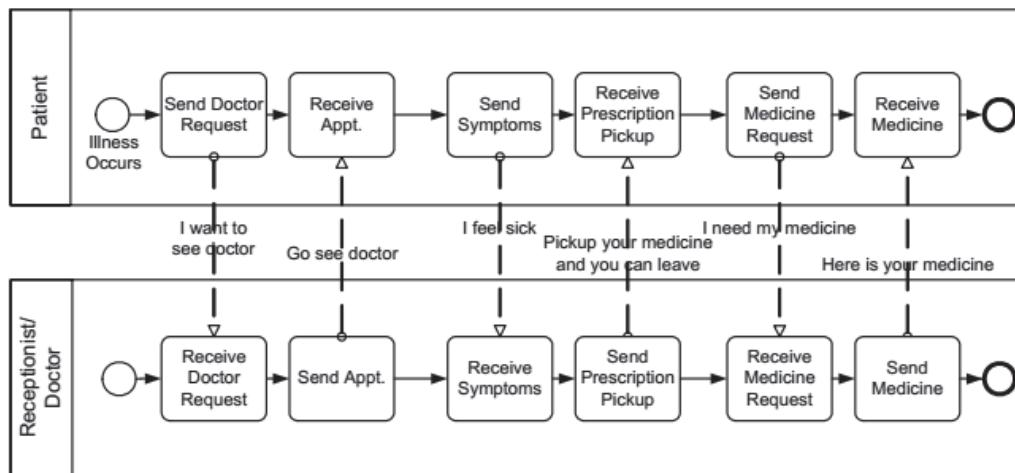


Fonte: BPMN (2014).

### 2.7.1.3 Processos de colaboração (global)

Representa as interações entre duas ou mais entidades que são definidas como uma sequência de atividades que mostram a troca de mensagens entre os padrões das entidades envolvidas. Esse modelo pode ser realizado em um ou mais processo, apontando os locais de comunicação entre eles. Caso seja um único modelo, cada raia de executor ou suporte retratará cada um dos participantes. Se for tratar de um processo privado e que esteja sendo modelado no mesmo diagrama BPMN, a associação das atividades que são compartilhadas por ambos os processos deverá ser feita. Confira na Figura 1.3 um exemplo de processo de colaboração.

Figura 1.3 Exemplo de processo de colaboração



Fonte: BPMN (2014).

Dentro e entre esses três modelos BPMN, diversos tipos de diagramas podem ser criados. A seguir, listamos os tipos de processos de negócio que podem ser modelos com o BPMN:

- └ Alto nível das atividades dos processos privados.
- └ Processos de negócio privado detalhados (AS IS ou processos de negócio抗igos e TO BE ou processos de negócio novos).
- └ Processo de negócios privados detalhados com interações para um ou mais entidades externas.
- └ Dois ou mais processos privados detalhados interagindo entre si.
- └ Relação detalhada do processo de negócio privado para o processo abstrato.
- └ Relação detalhada do processo de negócio privado para o processo colaborativo.
- └ Dois ou mais processos abstratos.
- └ Relação do processo abstrato para o processo colaborativo.
- └ Processo colaborativo.
- └ Dois ou mais processos de negócio privado detalhados interagindo por meio do seu processo abstrato.
- └ Dois ou mais processos de negócio privado detalhados interagindo por meio do seu processo colaborativo.
- └ Dois ou mais processos de negócio privado detalhados interagindo por meio do seu processo abstrato e processo colaborativo.

É importante ressaltar que a notação BPMN tem como direcionador a criação de um simples mecanismo para gerar modelos de processo de negócio e que também

é apto a lidar com a complexidade intrínseca aos processos de negócio. A abordagem usada para tratamento desses dois requisitos conflitantes foi a organização dos aspectos gráficos em categorias específicas, fornecendo um pequeno conjunto de categorias para que, ao consultar, possamos reconhecer rapidamente os tipos básicos dos elementos e compreender o diagrama.

Na categoria dos elementos básicos, podem ser adicionadas variações e informações complementares para suportar os requisitos de complexidade sem mudar o visual e sentimentos básicos do diagrama (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

As categorias básicas de elementos são:

a) **Objetos de fluxo**

Representam os principais elementos gráficos para definir o comportamento dos processos de negócio.

A entrada de sequência de fluxo pode ser vinculada para qualquer local em um objeto de fluxo (esquerda, direita, em cima ou em baixo). Do mesmo modo ocorre com uma saída de sequência de fluxo, que pode ser conectada de qualquer lugar em um objeto de fluxo (esquerda, direita, em cima ou em baixo).

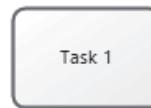


### *Para saber mais*

**Para saber mais sobre os 3 fluxos desta categoria (Evento, Atividade e Gateway):**



Evento



Atividade



Gateway

Aprofunde seus conhecimentos em BPMN acessando os links:

- └ <[http://www.cin.ufpe.br/~processos/TAES3/slides-2012.2/Introducao\\_BPMN.pdf](http://www.cin.ufpe.br/~processos/TAES3/slides-2012.2/Introducao_BPMN.pdf)>;
- └ <<http://www.omg.org/spec/BPMN/2.0/PDF/>>;
- └ <<http://www.tudosobrebpn.com.br/p/bpmn-business-process-modeling-notation.html>>;
- └ <<http://www.teclogica.com.br/web/consultoria/bpm/vantagens>>;
- └ <<http://planningit.wordpress.com/2012/10/24/bpmn-tecnicas-de-modelagem>>;
- └ <[http://www.informazione4.com.br/cms/opencms/desafio21/artigos/gestao/organizando\\_0017.html](http://www.informazione4.com.br/cms/opencms/desafio21/artigos/gestao/organizando_0017.html)>.

b) **Dados**

Os dados são representados pelos seguintes elementos:

- └ Objetos de dados.
- └ Entrada de dados.

- \_| Saída de dados.
- \_| Armazenamento de dados.

Na Tabela 1.3 a seguir, visualize os elementos da categoria dados (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

**Tabela 1.3 Dados**

| Objetos de dados | Tipos de dados                        |
|------------------|---------------------------------------|
|                  | Representação singular de dados       |
|                  | Representação de uma coleção de dados |
|                  | Dados de Entrada                      |
|                  | Dados de Entrada                      |
|                  | Armazenamento de dados                |
|                  |                                       |
|                  | Mensagem                              |

Fonte: BPMN (2014).

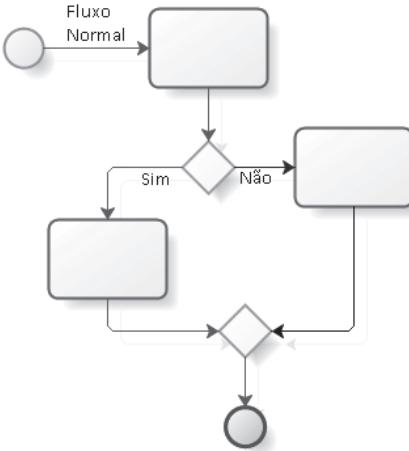
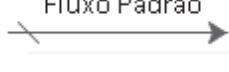
### c) Objetos de conexão

Para conectar os fluxos dos objetos entre si ou a outra informação, existem três objetos de conexão:

- \_| Fluxo de sequência.
- \_| Fluxo de mensagem.
- \_| Associação.

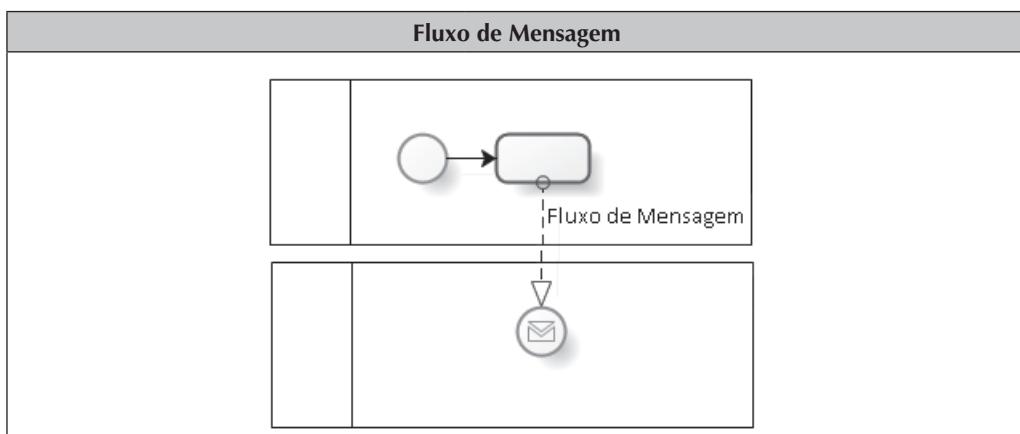
A seguir, temos a Tabela 1.4 contendo mais detalhes sobre esses objetos (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

Tabela 1.4 Objetos de conexão

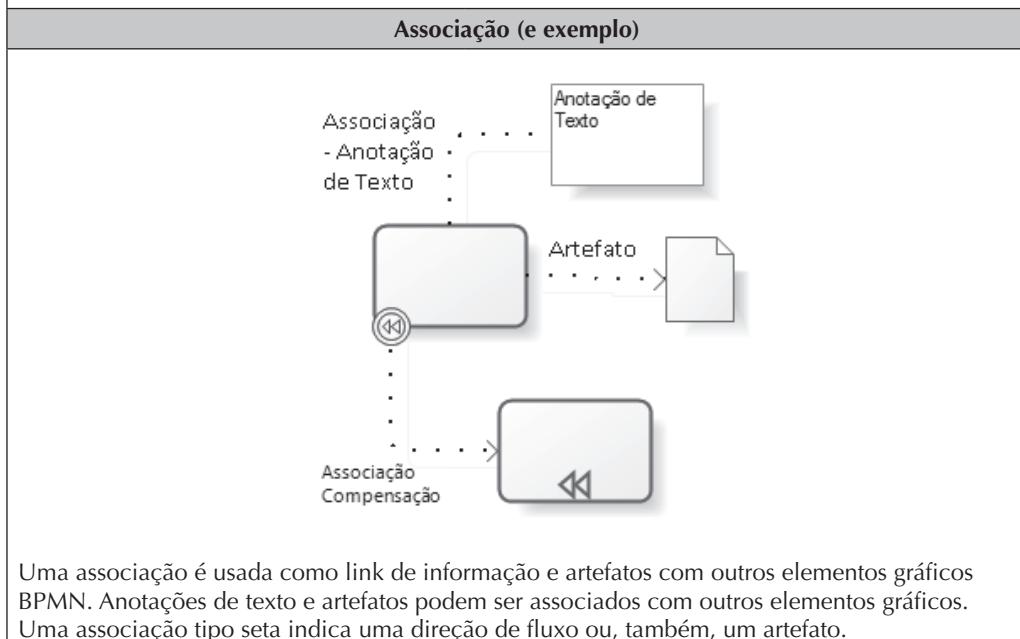
| Fluxo de sequência  | Tipos de fluxo de sequência (descrição e exemplos)  |
|---|---|
| <p><b>Fluxos de sequência são utilizados para mostrar a ordem em que as atividades devem ser executadas em um processo.</b></p> |  <p>O fluxo de sequência normal refere-se ao fluxo originado pelo evento de início e continua nas atividades seguintes via caminhos alternativos e paralelos até o seu término no evento fim.</p>           |
|   |  <p>Um fluxo descontrolado representa um fluxo que não é afetado por nenhuma condição ou não passa em nenhum gateway. Como exemplo, citamos um único fluxo de sequência conectando duas atividades.</p>    |
|   |  <p>O fluxo condicional pode ter expressões de condição que são avaliados em tempo para determinar se o fluxo será ou não utilizado.</p>  |
|   |  <p>Para as decisões exclusivas ou decisões inclusivas será utilizado este tipo de fluxo. Será utilizado somente se todos os fluxos condicionais de saída não forem verdadeiros no tempo de execução.</p> |

continua

continuação



Um fluxo de mensagem é usado para mostrar mensagens entre entidades que são preparadas para emitir-las e receber-las. Em BPMN, duas piscinas separadas no diagrama representam as duas entidades.



Uma associação é usada como link de informação e artefatos com outros elementos gráficos BPMN. Anotações de texto e artefatos podem ser associados com outros elementos gráficos. Uma associação tipo seta indica uma direção de fluxo ou, também, um artefato.

Fonte: BPMN (2014).

A Figura 1.4 a seguir demonstra quando os elementos podem ser conectados com o fluxo de sequência. O símbolo ↗ indica que o objeto listado na linha (From — De) pode ser conectar com o objeto listado na coluna (To — Para).

**Figura 1.4 Conexão entre objetos com fluxo de sequência**

| From\To     | ○ | Name<br>[+] | Name | ◇ | ○ | O |
|-------------|---|-------------|------|---|---|---|
| ○           | ↗ | ↗           | ↗    | ↗ | ↗ | ↗ |
| Name<br>[+] | ↗ | ↗           | ↗    | ↗ | ↗ | ↗ |
| Name        | ↗ | ↗           | ↗    | ↗ | ↗ | ↗ |
| ◇           | ↗ | ↗           | ↗    | ↗ | ↗ | ↗ |
| ○           | ↗ | ↗           | ↗    | ↗ | ↗ | ↗ |
| O           |   |             |      |   |   |   |

Fonte: BPMN (2014).

#### d) *Swinlanes*

Há o agrupamento dos elementos primários de modelagem através das *Swinlanes* (Tabela 1.5). São mecanismos de organização das atividades em categorias visuais separadas (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

- └ Piscinas (*pools*).
- └ Raias (*lanes*).

**Tabela 1.5 Swinlanes**

| Piscinas |   |
|----------|---|
|          |  |
| Raias    |   |
|          |  |

Representa um participante em um processo, atuando como um container gráfico para dividir um conjunto de atividades de outros pools.

Representa uma subpartição pertencente. São utilizadas para organizar e categorizar atividades.

Fonte: Do autor (2014).

### e) Artefatos

Usados para fornecer informação adicional sobre o processo. Existem artefatos padrão, porém, durante a modelagem, podem ser adicionados outros artefatos complementares. O conjunto de artefatos padrão inclui:

- └ Grupo.
- └ Anotação.

Confira na Tabela 1.6, a seguir, o detalhe de cada conjunto de artefatos (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

**Tabela 1.6 Artefatos**

| Grupo   |
|---|
|  <p>“Caixa” em volta de um conjunto de objetos em uma mesma categoria. Esse agrupamento não afeta a sequência do fluxo ou as suas atividades. O nome da categoria aparece como o rótulo do grupo. Essas categorias podem ser usadas para a documentação ou análise de efeitos. Os grupos são compostos com o objetivo de destacar visualmente as categorias de objetos no fluxo.</p> |

| Anotação  |
|---|
|  <p>É um mecanismo utilizado pelo modelador do processo com a finalidade de fornecer informações adicionais ao leitor do diagrama em BPMN.</p> |

Fonte: Do autor (2014).

Por todas as características apresentadas anteriormente, podemos concluir que a notação BPMN está construída sobre três pilares de sucesso. O primeiro representa o fato de ser uma linguagem visual de fácil compreensão para todos os participantes do processo. O segundo representa a série de recursos que torna possível uma modelagem de processos simples até os mais complexos. Por último, citamos a base sólida em fundamentação matemática, construída no conceito de *Pi-Calculus*, uma derivação do cálculo de processos para representação de processos de forma dinâmica e móvel (ELOGROUP, 2014; BPMN, 2014; TUDO SOBRE BPMN, 2014).

## 2.7.2 Fluxogramas

Segundo Ticontrol (2014), os fluxogramas têm como propósito fornecer uma representação gráfica dos elementos, componentes ou tarefas associadas com um processo. Auxiliam na documentação de objetivos, padronização de símbolos, promovem o entendimento comum de passos de um processo e os relacionamentos/dependências entre eles. Eles podem ser modelados em alto nível para leitores/usuários que não estão familiarizados com a terminologia de especificação de processos. Também, podem ser modelados em nível detalhado para uma organização que possua leitores/usuários com conhecimento em processo (ELOGROUP, 2014).

Um fluxograma típico pode ter tipos de símbolos descritos a seguir:

- └ Símbolos de início e fim representados por retângulos com bordas arredondadas.
- └ Normalmente, contêm a palavra “Início” ou “Fim” ou uma frase simbolizando o começo ou término de um processo.
- └ Setas que indicam que o controle passa de um símbolo para outro.
- └ Passos de processamento representados por retângulos.
- └ Dados de entrada e saídas representadas por paralelogramos.
- └ Condição ou decisão representada por losango.



### Para saber mais

Sobre as características do fluxograma, quando usar, como usar e suas vantagens e desvantagens, acesse os links:

- └ <<http://www.informazione4.com.br/cms/opencms/desafio21/artigos/gestao/organizando/0017.html>>;
- └ <<http://office.microsoft.com/pt-br/visio-help/criar-um-fluxograma-basico-HA010357088.aspx>>;
- └ <<http://producaoindustrialequality.blogspot.com.br/2011/02/fluxograma.html>>.

## 2.7.3 IDEF

IDEF (Integration Definition) é um método de modelagem de processos para um desenvolvimento seguro e sustentado, que de forma gráfica descreve todo o ciclo de vida de desenvolvimento de um sistema. É uma orientação através de padrões e critérios de análise (MODELAGEM DE PROCESSOS IDEF, 2014).

A notação aplica um conjunto simples de símbolos, que consistem em caixas de processos com setas composta por entradas, saídas, controles e mecanismos. Embora cada nível do modelo deva ser lido da esquerda para a direita e de cima para baixo, o sistema de numeração utilizado para os passos é representado de maneira que possibilite a associação entre níveis de pais e filhos no processo.

Apresenta uma estratégia abrangente, permitindo a representação empresarial e organizacional de maneira integrada e tem se transformado em um dos principais padrões (ELOGROUP, 2014; MODELAGEM DE PROCESSOS IDEF, 2014).

A proposta dos métodos e descrições é auxiliar na tomada de decisão. Cada tipo de método ou descrição está focado em um conjunto estreito de relacionamento e características de sistemas, comprehende um ponto de vista ou perspectiva global do sistema. Buscam um balanceamento favorável entre os métodos especiais de aplicações efetivas limitadas a tipos de problemas específicos e modelos que incluem tudo que é necessário. Provê mecanismos explícitos para integração de resultados de aplicações de métodos individuais.

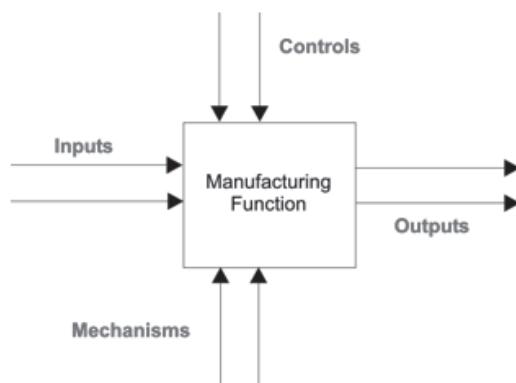
O foco das descrições a seguir será no método IDEF0 e IDEF3 (MODELAGEM DE PROCESSOS IDEF, 2014).

### 2.7.3.1 IDEF0 — Modelagem funcional

Representa o primeiro conjunto de padrões do IDEF, utilizando métodos estruturados para compreensão e melhoria dos processos de um sistema. Esse padrão de modelagem é útil para estabelecer o escopo de análise, especialmente em análise funcionais. Sendo uma ferramenta de comunicação, ajuda o modelador de processos a identificar quais as funções são realizadas, quais são as necessidades para a realização de determinadas funções e identifica em que o sistema atual está acertando e em que está errando.

O diagrama contém um nível de decomposição de um processo. No processamento de uma coleção de atividades e outras ações está presente um conjunto chamado de ICOMs (Inputs Control Output Mechanisms), setas e caixas (ELOGROUP, 2014; IDEF, 2014). Observe a Figura 1.5.

**Figura 1.5 IDEF0**



Fonte: IDEF0 (2014).

- **Input** — recebe o dado a ser convertido pela atividade.
- **Control** — responsabilidade de como e quando a entrada deve ser processada e executada.
- **Output** — resultado do processamento da entrada.

- ❑ **Mechanism** — representa quem deve executar esta atividade (pessoa, equipamento, máquina ou outras organizações).

Dentre os principais objetivos do IDEF0, estão:

- ❑ Fornecimento de recursos de maneira completa para a modelagem de funções (atividades, processos, operações e ações) requeridas.
- ❑ Técnica de modelagem independente de métodos ou ferramentas, mas que possam trabalhar em conjunto.
- ❑ Possibilidade de análise de sistemas com diversas finalidades, escopos ou complexidades.
- ❑ Facilidade de compreensão, comunicação, consenso e validação.
- ❑ Representação dos requisitos das funções físicas ou implementações organizacionais.
- ❑ Flexibilidade para suportar as várias fases do ciclo de vida de um projeto (ELOGROUP, 2014; IDEF, 2014a).



### *Para saber mais*

Veja algumas vantagens e desvantagens sobre IDEF0.

Acesse os links:

- ❑ <<http://www.idef.com/IDEF0.htm>>;
- ❑ <<http://www.idef.com/pdf/idef0.pdf>>;
- ❑ <[http://graco.unb.br/alvares/pub/idef0/idef0\\_cefet.pdf](http://graco.unb.br/alvares/pub/idef0/idef0_cefet.pdf)>.

#### **2.7.3.2 IDEF3—Modelagem para a descrição e captura de processos**

Este método fornece um mecanismo para coleta e documentação dos processos. A captura de precedência e causalidade das relações entre situações e eventos é feita uma forma natural de especialistas de domínio, fornecendo um método estruturado para expressar o conhecimento sobre como um sistema, processo ou organização executa suas atividades. O IDEF3 permite:

- ❑ Gravação dos dados brutos resultantes de atividades de análise em sistemas de averiguação.
- ❑ Determinação de impacto dos recursos de informação de uma organização sobre os principais cenários de operação da organização.
- ❑ Documentação dos processos de decisão que afetam os estados e do ciclo de vida de dados críticos compartilhados.
- ❑ Gerenciamento de configuração de dados e definição de políticas de controle.
- ❑ Geração de modelos de simulação.
- ❑ Gerar a concepção de um sistema e o design de análise de *trade-offs*.

Tem a capacidade de compreender os aspectos comportamentais de um sistema existente ou sugerido e captura todas as informações temporais, inclusive de precedência e causalidade associadas aos processos empresariais.

As descrições resultantes fornecem uma base de conhecimento estruturada para a construção de modelos de análise e design.

Existem dois modos para a descrição do IDEF3: fluxo de processos, transição da rede do estado do objeto. A descrição de um fluxo de processo captura o conhecimento de como as coisas funcionam.

O segundo modo permite a transição completa de um objeto para um determinado processo. A descrição do fluxo de processo captura a descrição de um processo e a rede de relações existente entre os processos dentro de um contexto.

A finalidade é apresentar como as coisas estão funcionando em determinada organização, quando são visualizadas como parte de um problema específico, resolvendo ou demonstrando uma situação recorrente. O desenvolvimento de um processo deste tipo consiste na demonstração de fatos, capturados a partir do domínio de um especialista (IDEF3, 2014).



### *Para saber mais*

Veja algumas vantagens e desvantagens sobre IDEF3:

Acesse o link: <<http://www.idef.com/IDEF3.htm>>.

Para conhecer os elementos do IDEF3:

Acesse os links: <<http://www.idef.com/IDEF3.htm>>;

<<http://paginas.fe.up.pt/~ee95027/idef3.pdf>>;

<<http://juno.unifei.edu.br/bim/0037603.pdf>> — tabela IDEF3 [28].

## 2.7.4 ARIS

A metodologia ARIS foi desenvolvida com o objetivo de integrar os processos de negócio de uma organização. É suportada pela ferramenta Aris Toolset.

Como primeiro passo para a criação dessa arquitetura, foi gerado um modelo composto por características para descrição de um processo. Assim, a divisão em vistas foi criada. Essas “vistas” são inter-relacionadas de maneira que permita que os modelos sejam analisados holisticamente e sem ambiguidades (ELOGROUP, 2014; TICONTRROLE, 2014). As visões do ARIS são:

- „ **Visão funcional:** permite a construção de modelos que definem de maneira hierárquica todas as funções da empresa, iniciando pelas macro e, após, decompondo-as até o nível de detalhe que especifique as funções de programação dentro do sistema.

- └ **Visão dos dados:** usada para definir os modelos de dados, iniciando pelas definições de informação mais complexas, passando pelo modelo de dados e seus relacionamentos, esquemas, até a definição da própria base de dados.
- └ **Visão organizacional:** permite a especificação e detalhamento da estrutura organizacional da empresa (divisões e unidades de negócios, estrutura de cargos e seus ocupantes, estrutura física com equipamentos).
- └ **Visão de controle:** permite o relacionamento entre as três visões anteriores. Nesta visão, existem métodos de modelagem específicos para a definição de relacionamentos entre as funções e dados, funções e organização, organização e dados e, principalmente, a integração das três funções.

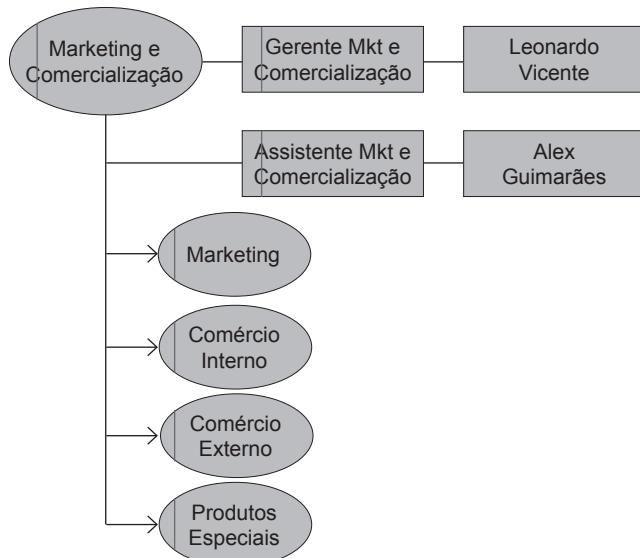
Esta metodologia tem como principais modelos: Cadeia de Valor Agregado (VAC), Diagrama de Objetivos (DO), Organograma (ORG), Diagrama de Alocação de Função (FAD) e Cadeia de Processos Orientada por Eventos (EPC) (ELOGROUP, 2014; TICONTRÔLE, 2014).

Nos próximos tópicos, apresentaremos maiores detalhes sobre cada um desses modelos citados.

#### 2.7.4.1 *Organograma*

Descreve as unidades organizacionais, representadas por retângulos, e a hierarquia, por linhas. Fornece a descrição da empresa que será convertida em objetos do modelo e conhecimento de organização e responsabilidades (TICONTRÔLE, 2014). Como exemplo de organograma, podemos observar a Figura 1.6 e a Tabela 1.7 a seguir.

**Figura 1.6 Exemplo de organograma**



Fonte: Ticontrol (2014).

**Tabela 1.7 Elementos de um organograma**

| Elemento               | Descrição   | Representação gráfica |
|------------------------|---|-----------------------|
| Unidade organizacional | Pode representar departamentos, setores, seções, diretorias etc.  |                       |
| Cargo                  | Corresponde aos postos de trabalho existentes na organização.   |                       |
| Pessoa interna         | Corresponde aos funcionários da organização.  |                       |
| Pessoa externa         | Representa pessoa ou organização externa.   |                       |
| Grupo                  | Corresponde ao grupo de funcionários que trabalham em conjunto com o objetivo de realizar atividades relacionadas entre si. |                       |

Fonte: Do autor (2014).

#### **2.7.4.2 Cadeia de Valor Agregado — VAC**

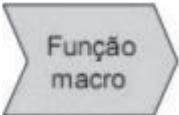
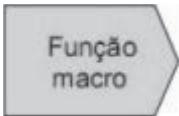
O modelo VAC tem como objetivo identificar funções dentro da organização que são envolvidas diretamente nos valores estratégicos da instituição. Estas funções podem ser inter-relacionadas pela criação de uma sequência de funções (TICON-TROLE, 2014).

Os processos podem ser relacionados aos seus respectivos objetivos e indicadores. Um modelo VAC pode ter um detalhamento em outros macroprocessos do mesmo tipo. No seu diagrama, as funções podem ser dispostas de uma forma hierárquica ou similar a uma árvore de função. A orientação do processo subordinado ou superior é sempre apresentada (DBD, 2014).

Este modelo é composto pelos seguintes objetos: unidade organizacional, função e *cluster*. O objeto de unidade organizacional tem a mesma definição que apresentamos anteriormente em outra notação. O objeto função tem diferenciação entre processos com predecessoras e processos iniciais. Já o *cluster* pode ser definido como um conjunto de dados e informações (ELOGROUP, 2014).

Confira na Tabela 1.8 a representação gráfica destes objetos.

Tabela 1.8 Objetos de um modelo VAC

| Representação gráfica   | Descrição  |
|---|--|
|  | Representa processos com predecessoras   |
|  | Representa processos iniciais (sem predecessoras)  |
|  | <i>Cluster</i> — conjunto de dados ou informações utilizadas para execução de atividades/processos (entradas — <i>inputs</i> ) e os itens resultantes deles (saídas — <i>outputs</i> ) |

Fonte: Afonso (2004).

#### 2.7.4.3 Diagrama de Objetivos (DO)

Antes de iniciar a modelagem, análise ou otimização dos processos de negócio, é necessário definir quais são os objetivos que a organização deseja alcançar com a modelagem.

Um objetivo define quais são objetivos de negócios futuros que devem ser atingidos durante a implementação dos novos processos. Os fatores que podem ser críticos para alcançar os objetivos podem ser especificados, organizados em uma hierarquia e relacionados às suas metas. Os fatores de sucesso que são necessários ao considerar como determinado objetivo de negócio será atingido estarão presentes no diagrama de objetivo (ARIS, 2014).

Os principais objetos utilizados neste modelo estão listados na Tabela 1.9; observe que os relacionamentos são limitados e não podem ser customizados.

Tabela 1.9 Objetos de um diagrama de objetivos

| Representação gráfica   | Objeto   | Descrição   |
|---|----------|---|
|  | Objetivo | Corresponde às metas a serem alcançadas pela organização. Pode ser decomposto em subobjetivos e deve estar relacionado a um processo (ELOGROUP, DBD, 2014). |

continua

continuação

|  |                    |  |
|--|--------------------|--|
|  | Processo           | Um processo pode conter um ou mais objetivos relacionados.   |
|  | Fator crítico      | Define os aspectos que devem ser considerados para que um objetivo específico seja alcançado (ELOGROUP, 2014). |
|  | Produto ou serviço | Define os elementos que servem de insumo ou são resultados do processamento de uma função.                     |

Fonte: Do autor (2014).

#### 2.7.4.4 Cadeia de Processos Dirigida por Eventos (EPC)

Este método é utilizado para modelagem de processo e tem grande aceitação no mundo (TICONTROLE, 2014). É o modelo central para toda modelagem de negócio, sendo um modelo dinâmico que traz os recursos estáticos do negócio (sistemas, organizações e dados). Possui características similares ao modelo gerado pelo uso da notação BPMN (DBD, 2014).

Representa a integração das várias visões do ARIS, tais como: função, dados, organização e saídas. É uma modelagem que contém detalhes do processo da organização e seus diagramas são encadeados em atividades sequenciais, sendo que seus fluxos, normalmente, são evento-função-evento e baseados em regras (ELOGROUP, 2014).

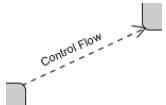
Este modelo pode conter diagramas bem simples e até os de complexidade muito alta (ABPMP, 2013). A Tabela 1.10, a seguir, mostra a lista de alguns dos possíveis objetos em um diagrama EPC.

Tabela 1.10 Objetos EPC — VISUAL

| Objeto | Descrição  |
|--------|--|
|        | Evento — descreve em quais circunstâncias uma função ou processo são executados ou quais são os resultados deles.  |
|        | Função — descreve as transformações de um estado inicial a um estado resultante.   |
|        | Operador E (And) — descreve que todas as saídas previstas, obrigatoriamente, geradas de um dado de entradas ou que todas as entradas são necessárias para que uma determinada saída seja gerada. |
|        | Operador Ou (Or) — corresponde à ativação de um ou mais caminhos entre os fluxos de controle.  |
|        | Operador Ou Exclusivo (XOr) — corresponde à tomada de decisão de qual o caminho escolher entre vários fluxos de controle.  |
|        | Unidade Organizacional — determina uma pessoa ou organização que é responsável por uma função específica dentro da estrutura de uma empresa.   |

continua

continuação

|   |  |
|---|--|
|  | Um fluxo de controle conecta eventos com a função, caminhos do processo ou operadores de sequência cronológica e a interdependência lógica entre eles. |
|  | Caminha — mostra a ligação para outros processos.  |

Fonte: Do autor (2014).



### Para saber mais

Sobre EPC, acesse os links:

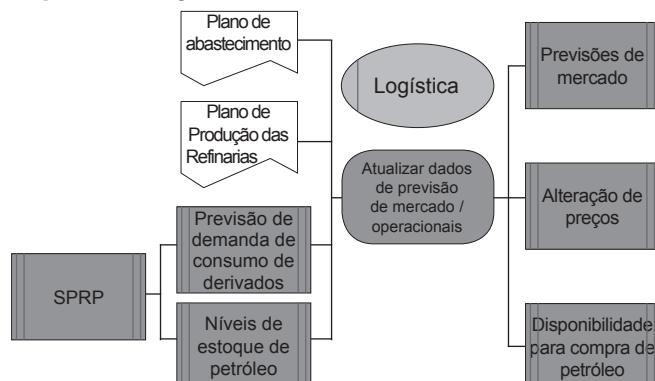
- ↳ <<http://bpmquotes.wordpress.com/2012/01/21/02-estrategia-e-processos-cadeia-de-valor/>>;
- ↳ <<http://www.visual-paradigm.com/VPGallery/bpmmodeling/epc.html#event>>.

#### 2.7.4.5 Diagrama de Alocação de Funções

O modelo FAD (Function Allocation Diagram) representa a transformação dos dados de entrada em dados de saída, identifica os responsáveis pela execução da atividade (papéis/unidade organizacional), e também os sistemas e recursos usados na execução das atividades (TICONTRROLE, 2014). Possui as informações sobre uma atividade do ponto de vista operacional, reduzindo a complexidade dos processos com a representação de elementos, tais como, as funções, áreas, sistemas de apoio, entradas e saídas, artefatos e riscos inerentes às atividades (DBD, 2014).

Uma das diretrizes para a modelagem é torná-la de fácil leitura, evitando carregar o diagrama com muita informação. Uma opção, neste caso, é o desmembramento dos modelos em unidades menores (ELOGROUP, 2013). Os objetos deste tipo de diagrama são os mesmos do EPC. O exemplo mostrado na Figura 1.7 ilustra o diagrama de alocação de funções.

Figura 1.7 Diagrama de alocação de funções



Fonte: Afonso (2004).



### Para saber mais

Veja algumas vantagens e desvantagens sobre ARIS.

Acesse o link: <<http://www.ariscommunity.com/>>.

### 2.7.5 UML (Linguagem de Modelagem Unificada)

A notação utilizada pela UML é padronizada pela OMG (Object Management Group) e facilita a compreensão de cada parte do sistema que está sendo modelado por qualquer pessoa que tenha conhecimento sobre a linguagem (CONNALEN, 2003).

As principais características são:

- \_| Utilizar atores e casos de uso para mostrar estruturas e fronteiras de um sistema e suas principais funções e funcionalidades.
- \_| Utilizar diagrama de classes, no qual demonstra atributos, operações e relacionamentos para representar a estrutura estática.
- \_| Representar a estrutura dinâmica com os diagramas de estado, sequência, colaboração e atividades.
- \_| Revelar a arquitetura de implementação física (hardware ou pacote software) com os diagramas de componentes e implementação.
- \_| Estender sua função por meio de estereótipos.

Assim é a linguagem de modelagem indicada para especificar, visualizar, construir e documentar um sistema.

Com todas essas características, é possível a representação de partes do sistema com os vários diagramas e a união destes diagramas representa o sistema como um todo.

Diagramas são recursos gráficos para a visualização de um sistema de diferentes perspectivas e geralmente por itens e relacionamentos. Esses diagramas podem ser divididos em três categorias (MACORATTI, 2014).

- \_| **Diagrama de estrutura:** representam a construção de blocos de recursos de sistemas que não mudam com o tempo. Dentre eles, estão: classe, objetos, estrutura composta, implantação, componentes e pacotes.
- \_| **Diagramas comportamentais:** mostram como o processo responde às alterações ou como evolui ao longo do tempo; entre eles estão: atividades, casos de uso e diagrama de máquina.
- \_| **Diagramas de interação:** descrevem as trocas de mensagem em processos colaborativos, sendo eles: diagrama “overview”, sequência, comunicação e tempo.



### *Para saber mais*

Para aprofundar seus conhecimentos em UML e ficar ligado nas atualizações, acesse os links:

- <<http://www.uml.org/>>;
- <[http://www.macoratti.net/vb\\_uml2.htm](http://www.macoratti.net/vb_uml2.htm)>;
- <[http://www.wthreex.com/rup/portugues/process/modguide/md\\_seqdm.htm](http://www.wthreex.com/rup/portugues/process/modguide/md_seqdm.htm)>;
- <<http://msdn.microsoft.com/pt-br/library/dd409360.aspx>>;
- <[http://www.wthreex.com/rup/portugues/process/modguide/md\\_bactd.htm](http://www.wthreex.com/rup/portugues/process/modguide/md_bactd.htm)>.

Veja também algumas bibliografias:

GUEDES, G.T. **UML 2**: Uma abordagem prática. 2. ed. São Paulo: Novatec, 2011.

FOWLER, Martin; Scott, Kendal. **UML essencial**. Porto Alegre: Bookman, 2000.

FURLAN, José Davi. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.



### *Questões para reflexão*

Você conhece os 3 amigos da UML, ou seja, os principais responsáveis por sua existência?



### *Atividades de aprendizagem*

1. Dentre as notações de modelagem de processos, apresentamos o BPMN (Business Process Model and Notation), cujo intuito é padronizar a comunicação entre os usuários do negócio. E, para apoiar essa notação, são apresentados 3 tipos de modelos básicos.

Com base nesta afirmativa, analise os itens a seguir e marque a alternativa **CORRETA**.

- a) Processos de negócio privado (global), processos abstratos (internos) e processos de colaboração (privado).
- b) Processos de negócio privado (internos), processos abstratos (públicos) e processos de colaboração (global).
- c) Processos de negócio internos, processos concretos (internos) e processos de colaboração (global).

- d) Processos de negócio públicos (global), processos concretos (internos) e processos de colaboração (privado).
  - e) Processos de negócio colaborativo (global), processos abstratos (internos) e processos de colaboração (privado)
2. O IDEF0 representa o primeiro conjunto de padrões do IDEF, utilizando métodos estruturados para compreensão e melhoria dos processos de um sistema. Esse padrão de modelagem é útil para estabelecer o escopo de análise, especialmente em análise funcionais.

Dentre os principais objetivos do IDEF0, podemos destacar:

- I. Fornecimento de recursos de maneira completa para a modelagem de funções (atividades, processos, operações e ações) requeridas.
- II. Técnica de modelagem independente de métodos ou ferramentas, mas que possam trabalhar em conjunto.
- III. Flexibilidade para suportar as várias fases do ciclo de vida Interativo de um projeto orientado a objetos e estruturado.

Com base nas afirmativas, assinale a alternativa CORRETA.

- a) Somente o item I.
- b) Somente o item II.
- c) Somente o item III.
- d) Somente os itens I e II.
- e) Somente os itens II e III.

### Seção 3

## Introdução à modelagem organizacional e ao método Enterprise Knowledge Development (EKD)

Caro aluno, segundo Pádua (2000), EKD é uma técnica de modelagem organizacional que facilita a compreensão do ambiente empresarial e é conhecida como uma atividade valiosa para a engenharia de requisitos, e que tem como objetivo “estreitar os laços” entre as áreas de negócio e de tecnologia da informação dentro de uma organização.

A necessidade de flexibilidade nas organizações decorrente de um mercado cada vez mais competitivo torna imprescindível o uso da tecnologia de informação. As constantes mudanças provocadas por essas tecnologias forçam as empresas a fazerem parte do processo de informatização para que continuem no negócio, uma vez que a informação tornou-se fundamental no contexto mercadológico. Entretanto, possuir informações não é o bastante. Uma empresa deve possuir métodos ágeis capazes de analisar essas informações, buscando a confecção de uma base de conhecimento que lhe servirá para reconhecer as mudanças e possibilitando sua antecipação aos eventos. Isso lhe garantirá sustentabilidade e competitividade em relação aos concorrentes.

Como tratar essas informações? Como tirar proveito dessas mudanças? Os processos de modelagem organizacional são os mecanismos capazes de contribuir com tais tarefas. Eles apresentam a finalidade de descrever os processos de negócios visando à otimização das informações que circulam entre os setores de uma empresa.

Dessa forma, a modelagem tem a função de representar como uma organização realmente funciona, não somente apresentando os elementos da empresa de forma separada, mas também a interação existente entre os diversos setores para construção de um todo.

Como exemplo de metodologia utilizada para a modelagem organizacional, podemos citar a EKD (Enterprise Knowledge Development). A modelagem em si é desenvolvida para garantir uma forma estruturada de representar o conhecimento organizacional, criando um ambiente tecnológico capaz de buscar aspectos relacionados à funcionalidade dos processos de negócios, descrever os elementos envolvidos nos processos e apresentar de maneira detalhada as etapas que constituem tais processos. Um ponto importante a se destacar em relação a modelagem EKD é que ela ainda apresenta a característica de agregar informações como objetivos da organização, restrições e regras de negócio a serem seguidas.

Podemos também caracterizar o objetivo do EKD como o fornecimento de uma descrição eficaz e não ambígua de como a organização trabalha, quais as condições e as razões para mudanças, quais as alternativas podem ser tomadas e quais critérios são utilizados para avaliação dessas alternativas (EKD\_MoMRN). Dessa forma, o EKD auxilia na especificação de requisitos e na construção de modelo organizacional que remete à organização e seus requisitos. Verificando-se como cada requisito é

relacionado aos objetivos, pessoas e atividades é possível a criação de um processo de conhecimento organizacional, que fornece diversas visões da organização.

A organização deve criar e manter organizado todo o conhecimento adquirido, recorrendo a ele quando uma necessidade de mudança é reconhecida. De acordo com Kirikova (EKD\_MO-Burbenko), a modelagem EKD pode ser usada em diferentes situações com diversas finalidades, como pode ser observado a seguir:

- \_| Na engenharia de requisitos para a criação e especificação de requisitos.
- \_| Na detecção de problemas usando a análise do negócio.
- \_| Na criação de novos sistemas de negócio por meio da reengenharia de processos do negócio.
- \_| No gerenciamento de conhecimento organizacional ou aprendizagem organizacional para formar a base de propagação e ampliação de conhecimento.

Todos esses aspectos resultaram em um modelo que poderá ser usado na definição de novas diretrizes a serem tomadas pela organização.

Na modelagem EKD, as tarefas são desenvolvidas por equipes, divididas por atividades, por usuários, engenheiros de requisitos e *stakeholders*. A denominação *stakeholder* é dada a todo membro que está envolvido no projeto, seja direta ou indiretamente. Também são aqueles que mesmo não possuindo poder de decisão ou que não conhecem profundamente o projeto, têm interesse em que esse seja desenvolvido. Exemplos de *stakeholders* diretos são os gerente e usuários finais e os indiretos são os fornecedores, alguns clientes, a concorrência e até mesmo a sociedade (EKD\_Helc-bubenko).

Como é de se esperar, a modelagem EKD permeia por todos os níveis de uma organização, isto é, passa pelo setor responsável pela estratégia, pelos gerentes e funcionários responsáveis pelo nível operacional até chegar ao facilitador, cuja função é liderar e advertir durante as reuniões de modelagem. De acordo com EKD\_Silvia todos contribuem com o cumprimento das seguintes tarefas:

- \_| Diagnóstico: modela a situação atual da organização e os requisitos de mudanças.
- \_| Entendimento ou compreensão: proposição de mudanças e discussão sobre a viabilidade de cada uma delas.
- \_| Projeto: apresenta e modela as situações alternativas e os possíveis cenários.

Assim sendo, o modelo EKD proporciona que essas três tarefas sejam discutidas por diferentes pontos de vista, envolvendo participantes de todos os setores da organização, cada qual fornecendo experiências e habilidades diferentes. As seções destinadas à modelagem devem ser abertas, construtivas e requerer a participação de todos. Dessa forma, o conhecimento organização se apresenta dependente dos participantes e não somente dos facilitadores.

Segundo Pádua (2000), o EKD traz os seguintes benefícios para as organizações:

- \_| Entender melhor o negócio.
- \_| Facilitar a aprendizagem e comunicação organizacional sobre questões essenciais.
- \_| Ajudar atender e promover as capacidades e processos da organização.

- ❑ Melhorar a comunicação e o entendimento dos envolvidos sob o enfoque de sistemas e de tecnologias.
- ❑ Descrever os objetivos da organização, de seus processos, o papel dos atores (que são as pessoas envolvidas nos processos de tarefas) e dos conceitos que a organização adota.
- ❑ Chegar a um documento chamado repositório de conhecimento, utilizado para um raciocínio sobre o negócio; discutir mudanças em situações futuras e traçar a cadeia de componentes das decisões formalizadas e adotadas.



### *Para saber mais*

Para aprofundar seus conhecimentos sobre EKD, acesse os links:

- ❑ <<http://crinfo.univ-paris1.fr/EKD-CMMRoadMap/index.html>>;
- ❑ <[http://people.dsv.su.se/~js/ekd\\_user\\_guide.html](http://people.dsv.su.se/~js/ekd_user_guide.html)>;
- ❑ <<http://www.prod.org.br/doi/10.1590/S0103-65132008000200005>>.

O modelo organizacional resultante da modelagem proposta pelo EKD é usado pelas pessoas responsáveis por tomar decisões sobre as futuras estratégias da empresa, táticas e objetivos. Tal modelo é formado por submodelos que representam alguns aspectos do organização.

## 3.1 Modelos de EKD

### 3.1.1 Modelo de Objetivos (MO)

Destaca a importância da descrição dos objetivos da organização. Nesse submodelo são descritos o que a empresa e seus empregados desejam alcançar ou evitar. Também tem a finalidade de esclarecer alguns pontos importantes da gerência da organização como por exemplo:

- ❑ Qual a prioridade dos objetivos e quais são os mais importantes?
- ❑ Qual a relação entre os objetivos?
- ❑ Existe algum impedimento para a realização de um objetivo?

### 3.1.2 Modelo de Regra de Negócio (MRN)

Este submodelo destina-se a afirmar e manter as regras de negócios estabelecidas pelo modelo e que estejam de acordo com o Modelo de Objetivos. É também função desse submodelo responder a perguntas como:

- ❑ Quais regras prejudicam os objetivos da organização?
- ❑ Quais são as políticas utilizadas?
- ❑ Como é a relação de cada regra com os objetivos?
- ❑ Como cada objetivo será adotado pelas regras?

### 3.1.3 Modelo de Conceitos (MC)

Seu uso restrigido define que esse submodelo atua somente definindo fenômenos e coisas relacionadas a outros submodelos. É responsável por representar conceitos, atributos e relações. Os conceitos são usados para definir estritamente expressões do Modelo de Objetivos, assim como o conteúdo do conjunto de informações do Modelo de Processos do negócio. Cabe ao Modelo de Conceitos:

- \_| Demonstrar as entidades ou conceitos que são identificados na organização levando-se em conta o relacionamento entre os objetivos, atividades e processos e atores;
- \_| Como as entidades são estabelecidas;
- \_| Definição de quais regras de negócios e restrições serão atribuídas aos objetivos e conceitos.

### 3.1.4 Modelo de Processo de Negócio (MPN)

Este modelo tem como objetivo definir os processos organizacionais, mostrando a forma de interação e manuseio das informações e materiais. Outra finalidade é apresentar quais são as entradas necessárias em termos de informação ou material. A elaboração do modelo de processo de negócios permite esclarecer [43]:

- \_| Quais são as atividades e processos reconhecidos na organização em concordância com as metas?
- \_| De que forma os processos deveriam ser realizados e quais as informações necessárias [43] [44]?

### 3.1.5 Modelo de atores e recursos

Este modelo define os tipos de atores e recursos envolvidos na atividade da empresa. Descreve como diferentes atores e recursos se relacionam uns com os outros e como ele se relacionam com os componentes do modelo de objetivos. Atores e recursos podem ser:

- \_| Indivíduos: representando uma pessoa na empresa;
- \_| Unidade organizacional: representa toda a estrutura organizacional da empresa;
- \_| Recursos materiais;
- \_| Funções: atribuições para indivíduos e unidades organizacionais [43] [44].

### 3.1.6 Modelo de requisitos e componentes técnicos

Define a estrutura e propriedades do sistema de informação para apoiar as atividades de negócio. Os modelos de Objetivos, Regras de Negócio, Processo de Negócios e Atores e Recursos representam uma descrição inicial para a empresa, entretanto, para desenvolver um sistema de informação que suporte esses processos, existe a necessidade de direcionamento da atenção para o sistema técnico que é necessário para apoiar os objetivos, processos e atores da organização [43] [44].



## Atividades de aprendizagem

1. A técnica de Modelagem Organizacional EKD facilita a compreensão do ambiente empresarial e é conhecida como uma atividade valiosa para a engenharia de requisitos, sendo organizada com seis submodelos, sendo estes:
    - a) Objetivos; Regras de Negócio; Conceitos; Processos de Negócio; Atores e Recursos; Modelo de Requisitos e Componentes Técnicos.
    - b) Objetivos; Regras de Negócio; Teste; Processos de Negócio; Atores e Recursos; Modelo de Requisitos e Componentes Técnicos.
    - c) Objetivos; Regras de Negócio; Concepção; Processos de Negócio; Atores e Recursos; Modelo de Requisitos e Componentes Técnicos.
    - d) Objetivos; Regras de Negócio; Transição; Processos de Negócio; Atores e Recursos; Modelo de Requisitos e Componentes Técnicos.
    - e) Objetivos; Regras de Negócio; Conceitos; Transição de Negócio; Atores e Recursos; Modelo de Requisitos e Componentes Técnicos.
  2. Com base nos Benefícios de EKD, analise as afirmativas abaixo:
    - I. Entender melhor o negócio, para que a organização vise aos interesses setorizados e não ao todo;
    - II. Facilitar a aprendizagem e comunicação organizacional sobre questões essenciais;
    - III. Ajudar a atender e promover as capacidades e processos da organização;
    - IV. Melhorar a comunicação e o entendimento dos envolvidos sob o enfoque de sistemas e de tecnologias.
- Assinale a alternativa CORRETA:
- a) Somente I, II.
  - b) Somente I, III.
  - c) Somente II, III, IV.
  - d) Somente III.
  - e) Somente V.



## *Questões para reflexão*

Conforme estudamos, o objetivo do EKD é apresentar uma visão da organização como um todo. E por que não trabalhar de forma setorizada? Não seria mais rápido utilizar a modelagem do processo?



## *Fique ligado!*

Caro aluno, chegamos ao final de nossa unidade. Vamos lembrar, aqui, as principais questões abordadas? Foram elas:

- └ Uma visão geral sobre as áreas de negócio.
- └ Abordamos o conceito de negócio e processos de negócio, bem como sua classificação.
- └ Nos aprofundamos em modelagem de processos de negócio, onde abordarmos algumas notações que tem como objetivo criar uma forma de comunicação padronizada.
- └ Abordamos também a modelagem organizacional, EKD, que visa à organização como um todo.
- └ E, com o intuito de reforçar o conteúdo estudado, tivemos algumas atividades de aprendizagem.



## *Para concluir o estudo da unidade*

Parabéns, você chegou ao final da Unidade 1.

Alguns dos conteúdos estudados servirão de base para os demais capítulos e outros para os demais semestres. Como, por exemplo, a UML (Unified Modeling Language, ou no português, Linguagem de Modelagem Unificada), que será um conteúdo abordado nos próximos semestres, sendo de extrema importância para a padronização na modelagem de sistemas orientados a objetos. Esta linguagem é conhecida mundialmente, ou seja, uma vez que você tenha aprendido seus conceitos não existem barreiras para sua aplicação, bastando apenas a definição dos modelos a serem trabalhados dentro da equipe de desenvolvimento e também, é claro, muita disciplina.

Sobre a camada de negócio, através dos modelos apresentados, foi possível identificar a importância em ter soluções que visam integrar os processos organizações,

ou seja, perde-se o conceito do trabalho setorizado para dar lugar ao integrado, proporcionando um grande crescimento organizacional.

As organizações que optam por investir em seus processos passam a oferecer aos seus clientes mais qualidade nos produtos e serviços.

E, para todo processo definido, exige-se disciplina na execução e revisão, ou seja, é necessária a melhoria contínua, onde a organização depende de processos e não de pessoas.



## *Atividades de aprendizagem da unidade*

1. Com base nas técnicas de modelagem, apresentamos alguns objetos gerados por elas. Relacione os objetos de modelagem apresentados na coluna da esquerda com a coluna da direita.

| OBJETOS DE MODELAGEM                              | TÉCNICAS DE MODELAGEM |
|---|-----------------------|
| a)  Atividade     Evento     Gateway     Conector | I. BPMN               |
| b)  | II. UML               |
| c)  | III. IDFO             |

Assinale a alternativa **CORRETA**:

- a — III, b — II, c — I
- b — II, b — III, c — I
- a — I, b — II, c — III
- a — I, b — III, c — II
- a — III, b — I, c — II

2. A respeito de BPMN, podemos afirmar que:
- a) É uma linguagem de programação utilizada somente para software comercial.
  - b) É uma notação para criar diagrama da UML, representando apenas os diagramas de caso de uso e classe.
  - c) As empresas adotam BPMN para atender somente às metodologias ágeis.
  - d) É uma notação padrão para o desenho de fluxogramas em processos de negócios. Sendo um conjunto de regras e convenções que determinam como os fluxogramas devem ser desenhados.
  - e) É uma notação adotada para representar somente os diagramas orientados a objetos.
3. Modelagem de processos pode ser definida como um conjunto de atividades envolvidas. A respeito dos seus objetivos, analise as seguintes sentenças:
- I. Entendimento da estrutura e a dinâmica das áreas da organização.
  - II. Entendimento dos problemas atuais da organização e identificar potenciais melhorias.
  - III. Garantir que os usuários e engenheiros de software trabalhem de forma individualizada, visando ao bem comum apenas de sua equipe.
  - IV. Auxílio na identificação de competências.
- Agora, assinale a alternativa CORRETA.
- a) Somente o item I está correto.
  - b) Somente o item II está correto.
  - c) Somente o Item III está correto.
  - d) Somente os itens I, II, III estão corretos.
  - e) Somente os itens I, II, IV estão corretos.
4. Complete afirmativa: \_\_\_\_\_ é uma técnica de modelagem organizacional que facilita a compreensão do ambiente empresarial e é conhecida como uma atividade valiosa para a \_\_\_\_\_, e que tem como objetivo “estreitar os laços” entre as áreas de negócio e a de tecnologia da informação dentro de uma organização.
- a) EKD, Engenharia de Requisitos.
  - b) UML, Engenharia de Software.
  - c) EKD, Elicitação de Requisitos.
  - d) IDEF, Elicitação de Requisitos.
  - e) BPMN, Análise de Sistemas.

5. Assinale V para verdadeiro e F para falso.

- ( ) Saídas são materiais ou informações que contribuirão para que o processo seja executado;
- ( ) Entradas representam o resultado obtido com a execução do processo, podem ser produtos ou serviços.
- ( ) Controle é responsável pelo monitoramento e verificação do cumprimento dos métodos, diretrizes, objetivos, procedimentos e padrões na execução do processo;
- ( ) Recurso representa a provisão de elementos que se fazem necessários para a execução do processo, tais como: humanos, materiais, financeiros entre outros.

---

## Referências

ABPMP. BPM CBOK. **Association of Business Process Management.** Brasil, 2013.

AFONSO, Fernanda Chalhoub. **Avaliação do Supply-Chain Operations Reference-Model (SCOR) como um modelo de referência de processos voltado para o gerenciamento de cadeias de suprimentos:** aplicação em um estudo de caso no setor de óleo & gás. Rio de Janeiro, 2004.

ARIS. **ARIS Method.** Disponível em: <[http://documentation.softwareag.com/arist/platform\\_72sr2e/method\\_manual\\_aris\\_s.pdf](http://documentation.softwareag.com/arist/platform_72sr2e/method_manual_aris_s.pdf)>. Acesso em: 22 mar. 2014.

BPMN. **BPMN — Técnicas de Modelagem.** Disponível em: <<http://planningit.wordpress.com/2012/10/24/bpmn-tecnicas-de-modelagem/>>. Acesso em: 19 mar. 2014.

BPMN. **Business Process Model and Notation (BPMN).** Disponível em: <<http://www.omg.org/spec/BPMN/2.0/PDF/>>. Acesso em: 19 mar. 2014.

BPQUOTE. **02 Estratégia e Processos (Cadeia de Valor).** Disponível em: <<http://bpquote.wordpress.com/2012/01/21/02-estrategia-e-processos-cadeia-de-valor/>>. Acesso em: 21 mar. 2014.

CAMPOS, André. **Modelagem de Processos:** Notações. Disponível em: <<http://www.tiespecialistas.com.br/2013/01/modelagem-de-processos-notacoes/>>. Acesso em: 19 mar. 2014.

CAMPOS, André. **Modelagem de Processos:** Notações. Disponível em: <<http://www.tiespecialistas.com.br/2013/03/modelagem-de-processos-o-que-nao-devo-fazer/>>. Acesso em: 19 mar. 2014.

CAPOTE, Gart. **Conceitos Fundamentais de BPM | Tipos de Processos de Negócio.** Disponível em: <<http://www.mundobpm.com/2011/07/conceitos-fundamentais-de-bpm-tipos-de.html>>. Acesso em: 19 mar. 2014.

CONNALEN, J. **Desenvolvendo aplicações Web com UML.** 2. ed. Rio Janeiro: Campus, 2003.

DBD. **Marco Teórico.** Disponível em: <[http://www2.dbd.puc-rio.br/pergamum/tesesabertas/1012645\\_2012\\_cap\\_2.pdf](http://www2.dbd.puc-rio.br/pergamum/tesesabertas/1012645_2012_cap_2.pdf)>. Acesso em: 21 mar. 2014.

DEVMEDIA. **Modelagem de processos:** um caminho para o sucesso da organização. Disponível em: <<http://www.devmedia.com.br/modelagem-de-processos-um-caminho-para-o-sucesso-da-organizacao/4785>>. Acesso em: 19 mar. 2014.

ELOGROUP. **Modelagem.** Disponível em: <[http://www.elogroup.com.br/wikibpm-modelagemprocessos/wikibpm\\_modelagem\\_de\\_processos.pdf](http://www.elogroup.com.br/wikibpm-modelagemprocessos/wikibpm_modelagem_de_processos.pdf)>. Acesso em: 19 mar. 2014.

EPC. **EPC Diagram.** Disponível em: <<http://www.visual-paradigm.com/VPGallery/bpmodeling/epc.html#event>>. Acesso em: 22 mar. 2014.

FLUXOGRAMA/PRODUÇÃO INDUSTRIAL E QUALIDADE. **Fluxograma.** Disponível em: <<http://producaoindustrialequalidade.blogspot.com.br/2011/02/fluxograma.html>>. Acesso em: 20 mar. 2014.

**IDEF.** **IDEF.** Disponível em: <<http://sylverio.com.br/blog/2013/08/idef/>>. Acesso em: 20 mar. 2014.

**IDEF. Integration Definition for Function Modeling (IDEF0).** Disponível em: <<http://www.idef.com/pdf/idef0.pdf>>. Acesso em: 20 mar.. 2014.

**IDEF3. IDEF3 Process Description Capture Method.** Disponível em: <<http://www.idef.com/IDEF3.htm>>. Acesso em: 21 mar. 2014.

**IPROCESS. Modelagem de Processos de Negócio:** Diferenças entre diagrama, mapa e modelo de processos. Disponível em: <<http://blog.iprocess.com.br/2014/02/modelagem-de-processos-de-negocio-diferencias-entre-diagrama-mapa-e-modelo-de-processos/>>. Acesso em: 19 mar. 2014.

**IVNET. Fluxogramas.** Disponível em: <<http://www.ivnet.com.br/educacional/osm/fluxogramas.pdf>>. Acesso em: 20 mar. 2014.

**JUNO. Análise da aplicabilidade da técnica de modelagem IDEF-SIM nas etapas de um projeto de simulação a eventos discretos.** Disponível em: <<http://juno.unifei.edu.br/bim/0037603.pdf> — tabela idef3>. Acesso em: 21 mar. 2014.

**MACORATTI. UML Conceitos Básicos.** Disponível em: <[http://www.macoratti.net/vb\\_uml2.htm](http://www.macoratti.net/vb_uml2.htm)>. Acesso em: 22 mar. 2014.

**MELO, I. S. Administração de sistemas de informação.** São Paulo: Pioneira Thomson Learning, 2006.

**MODELAGEM DE PROCESSOS IDEF. Modelagem de Processos IDEF:** Modelo Descritivo da Cadeia Produtiva do Biodiesel. Disponível em: <<http://revistas.utfpr.edu.br/pg/index.php/revistagi/article/view/525/482>>. Acesso em: 20 mar. 2014.

**MOREIRA, Marcelo dos Santos. Processos de negócios otimizados pelas tecnologias ECM.** Disponível em: <<http://www.revistasapere.inf.br/download/terceira/PROCESSOS.pdf>>. Acesso em: 19 mar. 2014.

**MPF. Modelagem de Processos de Negócio.** Disponível em: <<http://www.modernizacao.mpf.mp.br/bpm/modelagem-de-processos/elementos-da-notacao-bpmn>>. Acesso em: 19 mar. 2014.

**MSDN. Diagramas de atividade UML:** referência. Disponível em: <<http://msdn.microsoft.com/pt-br/library/dd409360.aspx>>. Acesso em: 23 mar. 2014.

**OFFICE. Criar um fluxograma básico.** Disponível em: <<http://office.microsoft.com/pt-br/visio-help/criar-um-fluxograma-basico-HA010357088.aspx>>. Acesso em: 20 mar. 2014.

**PÁDUA, S. I. D. Investigação do processo de desenvolvimento de software a partir da modelagem organizacional, enfatizando regras do negócio.** São Carlos. 144p. Dissertação (Mestrado) — Escola de Engenharia de São Carlos, Universidade de São Paulo, Área: Engenharia de Produção, São Carlos, 2000. Cap. 4.

**PÁGINAS. IDEF3 — Process Description Capture Method.** Disponível em: <<http://paginas.fe.up.pt/~ee95027/idef3.pdf>>. Acesso em: 21 mar. 2014.

**REDE GESTÃO. Por que Usar BPMN.** Disponível em: <<http://www.informazione4.com.br/cms/opencms/desafio21/artigos/gestao/organizando/0017.html>>. Acesso em: 19 mar. 2014.

TECLOGICA. **Vantagens BPM.** Disponível em: <<http://www.teclogica.com.br/consultoria/web/bpm/vantagens>>. Acesso em: 19 mar. 2014.

TICONTROLE. **Padrões para Modelagem de Processos no TCU.** Disponível em: <[http://www.ticontroler.gov.br/portal/page/portal/TCU/comunidades/gestao\\_processos\\_trab/padrao\\_modelagem/padrao\\_para\\_modelagem\\_processos.pdf](http://www.ticontroler.gov.br/portal/page/portal/TCU/comunidades/gestao_processos_trab/padrao_modelagem/padrao_para_modelagem_processos.pdf)>. Acesso em: 21 mar. 2014.

TUDO SOBRE BPMN. **BPMN — Business Process Modeling Notation.** Disponível em: <<http://www.tudosobrebpn.com.br/p/bpmn-business-process-modeling-notation.html>>. Acesso em: 19 mar. 2014.

UNB. **IDEF0 — Método de Representação de Processos em Forma de Fluxo.** Disponível em: <[http://graco.unb.br/alvares/pub/idef0/idef0\\_cefet.pdf](http://graco.unb.br/alvares/pub/idef0/idef0_cefet.pdf)>. Acesso em: 20 mar. 2014.

VALENÇA, George. **BPMN (Business Process Modeling Notation).** Disponível em: <[http://www.cin.ufpe.br/~processos/TAES3/slides-2012.2/Introducao\\_BPMN.pdf](http://www.cin.ufpe.br/~processos/TAES3/slides-2012.2/Introducao_BPMN.pdf)>. Acesso em: 19 mar. 2014.

WIKIPÉDIA. **Arquitetura de processos.** Disponível em: <[http://pt.wikipedia.org/wiki/Arquitetura\\_de\\_processos](http://pt.wikipedia.org/wiki/Arquitetura_de_processos)>. Acesso em: 19 mar. 2014.

WTHREEEX. **Diagramas de atividade UML:** referência. Disponível em: <[http://www.wthreex.com/rup/portugues/process/modguide/md\\_bactd.htm](http://www.wthreex.com/rup/portugues/process/modguide/md_bactd.htm)>. Acesso em: 23 mar. 2014.

WTHREEEX. **Diretrizes:** Diagrama de Sequência. Disponível em: <[http://www.wthreex.com/rup/portugues/process/modguide/md\\_seqdm.htm](http://www.wthreex.com/rup/portugues/process/modguide/md_seqdm.htm)>. Acesso em: 23 mar. 2014.

# Engenharia de software

*Luis Cláudio Perini*

## Objetivos de aprendizagem:

- \_| Compreender a natureza e tipos de software.
- \_| Entender o que é engenharia de software e por que ela é importante.
- \_| Saber as respostas para as questões-chave da engenharia de software.
- \_| Entender as questões profissionais e éticas da engenharia de software.
- \_| Entender o que é um processo de software.
- \_| Compreender quais as atividades genéricas que estão presentes em todos os processos de software.
- \_| Compreender os conceitos e modelos de processos de software.
- \_| Identificar os modelos de processos prescritivos bem como seus pontos fortes e fracos.

### \_| Seção 1: **Natureza do software da engenharia de software**

Nesta seção, abordaremos uma discussão inicial sobre os conceitos iniciais de software e da engenharia de software. Abordaremos também sobre o que foi a crise de software e como delineou os objetivos da engenharia de software.

### \_| Seção 2: **Modelos de processo de software**

Nesta seção, abordaremos uma discussão sobre os conceitos de processos de software bem como os tipos de modelos existentes no mercado.

---

## Introdução ao estudo

Segundo Pressmann (2011), atualmente o software assume um duplo papel, em que primeiro ele é um produto e segundo, ao mesmo tempo torna-se um meio para distribuir um produto. Analisando o mesmo como sendo um produto, ele fornece o potencial computacional representado pelos dispositivos de hardware ou por uma rede de computadores independentemente de onde estiver instalado, seja em um celular ou dentro de um mainframe, software é um transformador de informações, isto é, produzindo, gerenciando, adquirindo, modificando, exibindo ou transmitindo informações que podem ser tão simples quanto um único bit ou tão complexas quanto uma apresentação multimídia derivada de dados obtidos de dezenas de fontes independentes.

Conforme Pressmann (2011), vendo o software como veículo de distribuição, ele atua como a base para o controle do computador (sistemas operacionais), a comunicação de informações (redes) e na criação e controle de outros programas (ferramentas de software e ambientes). O software distribui o produto mais importante de nossa era — a informação, ou seja, transformando dados pessoais ou corporativos (exemplo, transações financeiras de um indivíduo) de uma maneira que possam ser mais úteis em uma determinada situação ou contexto, gerenciando informações comerciais no intuito de aumentar a competitividade, fornecendo um portal para redes mundiais de informação (Internet) e os meios para obter informações sob todas as suas formas.

Sob esse prisma, o papel desempenhado pelo software tem passado por grandes mudanças ao longo das últimas décadas, com aperfeiçoamentos significativos no desempenho do hardware, mudanças profundas nas arquiteturas computacionais, grande aumento na capacidade de memória e armazenamento, e uma vasta variedade de opções de dispositivos de entrada e saída, tudo isso resultando em sistemas computacionais mais sofisticados e complexos.

---

### Seção 1 **Natureza do software e da engenharia de software**

Hoje o software é uma tecnologia única e importantíssima no cenário mundial. Ninguém 50 anos atrás poderia prever que o software se tornaria uma tecnologia indispensável tanto para os negócios, para a ciência e engenharia e que pudesse permitir a criação e extensão de novas tecnologias tais como as ligadas à engenharia genética, às telecomunicações entre outras e também a mudança radical nas tecnologias mais antigas (por exemplo, indústria gráfica), que se tornaria a força motriz da revolução do computador pessoal, que uma empresa de software fosse se tornar maior e mais influente que a maioria das empresas da era industrial, que uma enorme rede (internet) orientada por um software pudesse evoluir e modificar tudo desde pesquisas, a

forma de as pessoas efetuarem compras e até mesmo a forma de marcar encontros e de se relacionar com outras pessoas

Praticamente todos os países, hoje em dia, dependem de sistemas complexos baseados em computadores desde infraestruturas e serviços que contam com sistemas baseados em computadores, e a maioria dos produtos elétricos/eletônicos possui embutido um computador e um software de controle, a manufatura industrial está completamente automatizada bem como os sistemas financeiros. Dessa forma, produzir e manter o software dentro de custos adequados é essencial para o funcionamento da economia nacional e internacional.

Assim sendo a engenharia de software é um ramo da engenharia em que o foco é o desenvolvimento de sistemas de software dentro de custos, prazo e qualidade adequados. Uma vez que o software é abstrato e intangível, não sendo limitado por materiais ou controlado por leis da física ou por processos de manufatura. Dessa forma não existem limitações físicas no potencial de software, porém, a falta de restrições significa que o software pode facilmente se tornar extremamente complexo e, portanto, difícil de ser compreendido.

Seria quase impossível prever que o software estaria embutido em sistemas em varias áreas tais como transportes, médica, telecomunicações, militar, industrial, entretenimento etc. e também que milhões de programas de computador tivessem que ser corrigidos, adaptados e aperfeiçoados e que o ônus de prestar tal serviço (manutenção) absorveria mais pessoas e recursos que todo o trabalho de desenvolvimento de novos softwares.

Atualmente, com a web 2.0 e a computação pervasiva, que vem surgindo com força, estamos por ver uma geração completamente diferente de software. Ele será distribuído via internet e parecerá estar residente nos dispositivos do computador de cada usuário... Porém, estará residente em um servidor bem distante.

Conforme aumenta a importância do software, a comunidade da área tenta desenvolver tecnologias que tornem mais fácil, mais rápido e mais barato desenvolver e manter programas de computador de alta qualidade. Algumas dessas tecnologias são direcionadas a um campo de aplicação específico (por exemplo, projeto e implementação de sites); outras são focadas em um campo de tecnologia (por exemplo, sistemas orientados a objetos ou programação orientada a aspectos); e ainda outras são de bases amplas (por exemplo, sistemas operacionais, fanáticos por tecnologia Linux). Entretanto, nós ainda temos de desenvolver uma tecnologia de software que faça tudo isso, embora a probabilidade de surgir tal tecnologia no futuro seja pequena. Ainda assim, as pessoas apostam seus empregos, seu conforto, sua segurança, seu entretenimento, suas decisões e a própria vida em software.

Historicamente o conceito de *engenharia de software* foi de início proposto em 1968, no intuito de discutir a “crise de software”, que resultava diretamente da introdução de novo hardware de computador baseado em circuitos integrados, cujo poder fez das aplicações de computador, consideradas até então não realizáveis, propostas

viáveis e o software resultante era de ordem de grandeza maior e mais complexo que sistemas até então criados.

No início a construção desses sistemas mostrou que o desenvolvimento informal de software não era suficiente, pois alguns projetos importantes apresentavam, geralmente, anos de atraso e o custo do software superava as previsões, não era confiável, era difícil de manter e seu desempenho era insatisfatório. Assim sendo o desenvolvimento de software estava em crise, pois os custos de hardware estavam caindo, enquanto os custos de software aumentavam rapidamente. Para fazer frente a esses novos problemas, novas técnicas e métodos tornaram-se necessários para controlar a complexidade no desenvolvimento de grandes sistemas de software.

Essas técnicas tornaram-se parte da engenharia de software e são amplamente usadas hoje; contudo, da mesma forma que aumentou a habilidade de produzir software, cresceu também a necessidade por sistemas de software mais complexos. O surgimento de novas tecnologias resultantes da convergência de computadores e sistemas de comunicação, e as complexas interfaces com o usuário impuseram novos desafios aos engenheiros de software. Como muitas empresas ainda não aplicam as técnicas de engenharia de software de forma efetiva, muitos projetos de software são produzidos com baixa confiabilidade, em atraso e com custo além do orçamento.

Penso que fizemos enormes progressos desde 1968, entendemos melhor as atividades envolvidas no desenvolvimento de software. Criamos métodos eficazes de especificação, projeto e implementação de software e essas novas notações e ferramentas têm por objetivos reduzir o esforço necessário para produzir sistemas complexos e de grande porte. Agora sabemos que não existe uma abordagem única “ideal” para a engenharia de software, pois, com a diversidade de tipos de sistemas e organizações que usam esses sistemas, precisamos de uma gama de abordagens para o desenvolvimento de software, porém noções fundamentais de processo e de organização de sistemas constituem a base de todas essas técnicas que formam a essência da engenharia de software. Sem softwares complexos, não seria possível a exploração do espaço, a Internet e os modernos sistemas de telecomunicações não existiriam e todos os meios de viagem seriam mais perigosos e caros.

No Quadro 2.1, resumimos alguns questionamentos sobre software citados por Pressmann (2011, p. 29) e Sommerville (2007, p. 4-10).

**Quadro 2.1 Questionamentos sobre software**

| <b>Questionamentos</b>             | <b>Descrição</b>  |
|------------------------------------|---|
| O que é um software?               | <p>Software é o produto que profissionais desenvolvem e ao qual dão suporte no longo prazo e que abrange programas executáveis, seja em um computador de qualquer porte ou arquitetura, seus conteúdos, suas informações tanto na forma impressa como na virtual e abrangendo praticamente qualquer mídia eletrônica. Considerando a área de engenharia de software, o mesmo abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade. Já um sistema de software consiste em um conjunto de programas separados, de arquivos de configuração que são utilizados para configurar esses programas, a documentação do sistema que descreve a estrutura do sistema, a documentação do usuário que explica como usar o sistema e sites web onde os usuários obtêm informações sobre o produto.</p> <p>Existem dois tipos fundamentais de produtos de software:</p> <ul style="list-style-type: none"> <li>Produtos genéricos. São sistemas produzidos por uma organização de desenvolvimento e vendidos no mercado para qualquer cliente disposto a comprá-los.</li> <li>Produtos sob encomenda (<i>ou personalizados</i>). São os sistemas encoroadados por um determinado cliente, sendo o software desenvolvido especialmente para aquele cliente por uma empresa de software.</li> </ul> <p>A diferença entre esses tipos é que nos produtos genéricos a organização que desenvolve o software controla sua especificação; já nos produtos encomendados, a especificação é desenvolvida e controlada pela organização que compra o software e os desenvolvedores de software devem trabalhar de acordo com tal especificação.</p> |
| Quem produz um software?           | São engenheiros de software que criam e dão suporte a ele.  |
| Qual a importância de um software? | Ele é importante pois afeta a quase todos os aspectos de nossa vida e incorporou-se no comércio, na cultura, no entretenimento e em nossas atividades cotidianas. Já o software na engenharia de software é importante porque ela nos dá condições de desenvolver sistemas complexos dentro do prazo e com alta qualidade.  |
| Como se produz um software?        | Um software é criado da mesma forma que qualquer produto bem-sucedido: aplica-se um processo que conduza a um resultado de alta qualidade e que atenda às necessidades daqueles que o solicitarão, para tal aplicando uma abordagem de engenharia de software.  |
| O que é um artefato de software?   | Na visão de um engenheiro de software, é um conjunto de programas, conteúdo (dados) e outros dispositivos que compõem o ambiente de um software. Já na visão do usuário, consiste em informações que de alguma forma tornam a vida dele melhor.   |

continua

continuação

|   |   |
|---|---|
| O que é engenharia de software?   | A engenharia de software é uma disciplina de engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que este entrar em operação.   |
| Qual é a diferença entre engenharia de software e ciência da computação?  | Essencialmente, a ciência da computação diz respeito às teorias e aos métodos que constituem a base de computadores e sistemas de software, por outro lado a engenharia de software se dedica aos problemas práticos da produção de software.   |
| Qual é a diferença entre engenharia de software e engenharia de sistemas? | A engenharia de sistemas diz respeito a todos os aspectos do desenvolvimento e da evolução de sistemas complexos, nos quais o software desempenha um papel importante. Está relacionada ao desenvolvimento de hardware, projeto de políticas e de processos e implantação do sistema, assim como com a engenharia de software. Os engenheiros de sistemas se envolvem com a especificação do sistema, com a definição da arquitetura geral e com a integração de diferentes partes do sistema. Porém estão menos envolvidos com a engenharia dos componentes do sistema (como hardware, software etc.).   |
| O que é processo de software?   | É um framework e resultados associados que produzem um produto de software. Existem quatro atividades fundamentais de processo que são comuns a todos os processos de software. São elas: especificação de software; desenvolvimento de software; validação de software e evolução de software.   |
| O que é um modelo de processo de software?                                | Um modelo de processo de software é uma descrição simplificada que apresenta uma visão dele. Os modelos de processo incluem as atividades, que fazem parte do processo de software, os produtos de software e os papéis das pessoas envolvidas na engenharia de software. A maior parte dos modelos de processo se baseia em um dos três paradigmas de desenvolvimento a seguir: <b>modelo em cascata</b> que representa as atividades citadas acima em fases separadas de processo, como especificação de requisitos, projeto de software, implementação e teste; <b>desenvolvimento iterativo</b> intercala as atividades de especificação, desenvolvimento e validação e <b>engenharia de software baseada em componentes</b> (CBSE — Component Based Software Engineering) supõe que partes do sistema já existam e, assim sendo, o processo de desenvolvimento concentra-se mais na integração dessas partes do que no seu desenvolvimento a partir do início. |

continua

continuação

|  |  |
|--|--|
| Quais são os custos da engenharia de software? | <p>A distribuição do custo ao longo das diferentes atividades no processo de software depende do processo e do tipo de software que está sendo desenvolvido, pois na abordagem cascata os custos de especificação, projeto, implementação e integração são medidos separadamente. Na abordagem iterativa, não existe uma linha precisa entre especificação, projeto e desenvolvimento. Na engenharia de software baseada em componentes tem sido aplicada intensamente porém ainda não há valores precisos dos custos de diferentes atividades de desenvolvimento de software. Sabe-se que os custos de desenvolvimento são menores que os custos de integração e de teste.</p> <p>Além dos custos de desenvolvimento, os custos também ocorrem na manutenção no software após sua liberação para uso. Os custos de evolução variam muito, dependendo do tipo de sistema, pois para sistemas de software de vida longa, ou seja, em sistemas que podem ser usados por dez anos ou mais, esses custos provavelmente excedem de três a quatro vezes os custos de desenvolvimento.</p> <p>Essas distribuições de custo são válidas para software sob encomenda, conforme especificado pelo cliente e desenvolvido por um fornecedor. Esses produtos são geralmente desenvolvidos a partir de um esboço de especificação, usando uma abordagem evolucionária de desenvolvimento e, desta forma, portanto, os custos de especificação são relativamente baixos.</p> <p>Os custos de evolução para os produtos genéricos de software são particularmente difíceis de serem estimados, em vários casos, existe uma pequena evolução formal de um produto.</p> |
| O que são métodos de engenharia de software?   | <p>Um método de engenharia de software é uma abordagem estruturada para desenvolvimento de software, cujo objetivo é facilitar a produção de software de alta qualidade dentro de custos adequados. Métodos tais como Análise Estruturada (DcMarco, 1978) foram desenvolvidos inicialmente na década de 1970. Nas décadas de 1980 e 1990, os métodos orientados a funções foram suplementados por métodos orientados a objetos (OO), como os propostos por Booch (1994) e Rumbaugh et al. (1991). Não existe um método ideal, e diferentes métodos possuem diferentes áreas onde são mais aplicáveis. Todos os métodos estão baseados na ideia de modelos de desenvolvimento de um sistema, que pode ser representado graficamente, e na ideia de uso desses modelos como especificação e projeto de um sistema.</p>   |
| O que é CASE?                                  | <p>O acrônimo CASE corresponde a Computer-Aided Software Engineering, ele abrange uma larga faixa de diferentes tipos de programas que são usados para dar apoio às atividades do processo de software, tais como análise de requisitos, modelagem de sistema, depuração e teste. As ferramentas CASE podem também incluir um gerador de código que gera automaticamente o código-fonte com base no modelo do sistema, e algumas orientações sobre o processo para os engenheiros de software.</p>   |

continua

continuação

|   |  |
|---|--|
| <p>Quais são os atributos de um bom software?</p> | <p>Assim como os serviços que ele fornece, os produtos de software possuem outros atributos associados que demonstram a qualidade do software. Esses não estão relacionados diretamente com o que o software faz, mas, em vez disso, refletem o comportamento do software, enquanto está em execução, e a estrutura e a organização do programa-fonte bem como a documentação associada.</p> |
|---|--|

Fonte: Adaptado de Pressmann (2011) e Sommerville (2007).

Sommerville (2007) coloca os seguintes desafios para a engenharia de software:

1. O desafio da heterogeneidade: cada vez mais é necessário que os sistemas de software operem como sistemas distribuídos, através de redes, que incluem diferentes tipos de computadores, com diferentes tipos de sistemas de apoio. O desafio de heterogeneidade consiste em desenvolver técnicas para construção de software confiável que seja flexível o suficiente para adaptar-se a essa heterogeneidade.
2. O desafio da entrega: muitas técnicas tradicionais de engenharia de software demandam tempo, tempo esse necessário para obter a qualidade do software. Porém levando em conta o atual ambiente de negócios há necessidade de apresentar respostas ágeis e mudar rapidamente. O software de apoio deve acompanhar a velocidade da mudança. O desafio da entrega consiste em diminuir os tempos de entrega dos sistemas grandes e complexos, sem comprometer a sua qualidade.
3. O desafio da confiança: como o software está entrelaçado com todos os aspectos da nossa vida, é essencial que possamos confiar nele. O desafio da confiança é desenvolver técnicas que demonstrem que o software pode ter a confiança dos seus usuários.

## 1.1 Evolução do software

Para que um software possua uma ampla utilidade, ele necessita de centenas ou milhares de algoritmos. Atualmente os softwares são bastante complexos e podem ser formados por diversos programas.

O software evoluiu bastante na segunda metade do século XX quando surgiram os primeiros computadores. Na **primeira era**, que teve início na década de 1950 até meados dos anos 1960, o hardware sofreu contínuas mudanças em computadores de baixa capacidade (embora ocupassem salas enormes), com o processamento de dados em lote (sem interação com o usuário no decorrer da interação), e os softwares eram feitos e executados para clientes específicos com distribuição limitada, não havendo preocupação com a documentação.

Já na **segunda era**, que ocorreu entre os anos 1960 a meados dos 1970, surgiram sistemas de tempo real de grande porte multiusuários e as primeiras aplicações de

banco de dados. Teve início a comercialização de softwares genéricos (criação das primeiras software-houses) que atendiam às necessidades de diversos clientes, bem como o surgimento de bibliotecas de software, visto que ocorreu um crescimento no número de sistemas baseados em computador, pois o acesso aos computadores tornou-se mais fácil por parte das médias e grandes empresas. Além do crescimento do uso e comercialização do software, ainda se verificava a pouca importância dada à manutenção, que era difícil de ser feita. No Brasil, softwares com essas características foram utilizados até meados dos anos 1980. Linguagens de programação de alto nível facilitaram em parte a construção de software nessa segunda era.

A **terceira era** começou em meados dos anos 1970 com o surgimento de hardware de baixo custo baseado em microprocessadores (surgimento dos PC's) e estendeu-se até início dos anos 1990 e com o aumento dos sistemas distribuídos e das redes locais e globais, com o hardware a cada dia mais em conta, possibilitando assim às empresas adquirirem mini e microcomputadores. Com esse impacto no consumo, consequentemente, houve uma grande demanda por software. E por conta do aumento do consumo, devido também ao uso generalizado de microprocessadores por parte da indústria, gerando produtos inteligentes (micro-ondas, TV com controle remoto etc.), foi sentida a falta de mão de obra especializada, de técnicas e ferramentas de desenvolvimento de software, e de planejamento e gerenciamento do processo para atender a esse impacto de consumo levando à **crise do software**. Essa crise do software levou ao surgimento de técnicas de programação como modularização e refinamentos sucessivos e metodologias para desenvolvimento de software como a Análise e Projetos Estruturados e o Método de Jackson, que buscavam resolver os problemas de desenvolvimento de software.

A **quarta era**, a partir do final dos anos 1990 e início do século XXI, caracteriza-se pelo surgimento de tecnologias orientadas a objeto. Grandes empresas optaram por trocar computadores de grande porte por sistemas distribuídos num fenômeno denominado de *downsizing*, em que se iniciou o uso de sistemas especialistas e software de inteligência artificial e de redes neurais agora usados na prática. A popularização dos microcomputadores de baixo custo levou ao surgimento de inúmeros softwares aplicativos “de prateleira” comprados por pequenas e microempresas e também por usuários comuns. Também o surgimento e a popularização da Internet e o surgimento da World Wide Web (www), em meados da década de 1990, causaram uma revolução na forma como as aplicações são desenvolvidas, utilizadas e comercializadas. O surgimento de linguagens voltadas para esses ambientes, como Java, e de plataformas de interoperabilidade para sistemas de software distribuídos, como CORBA, vêm indicando que estamos entrando numa nova era.

Apesar de toda essa evolução do software, alguns problemas persistem e outros se intensificaram, tais como:

❑ **A habilidade em construir software deixa a desejar em relação ao potencial do hardware.**

Hoje o potencial do hardware, devido às constantes atualizações tecnológicas, é superior à capacidade dos softwares, pois o hardware é manufaturado e o software não, isto é, depende dos conhecimentos dos profissionais acerca das novas tecnologias disponíveis no mercado, sejam elas novas versões de linguagens de programação, banco de dados etc.

❑ **A construção de software não é rápida o suficiente para atender as necessidades do mercado.**

Como o software depende de pessoas para a realização dos projetos e também devido ao aumento da complexidade das aplicações, o tempo exigido para o desenvolvimento do produto de software muitas vezes não é rápido o suficiente para atender às necessidades dos usuários, sejam eles pessoa física, empresas ou órgãos governamentais.

❑ **A sociedade depende cada vez mais de software confiável; quando ele falha, podem ocorrer gastos enormes e desgaste de muitos profissionais para arrumá-lo.**

A falta de documentação e o uso por parte dos desenvolvedores de metodologias para desenvolvimento e testes de software têm levado muitos projetos a ser liberados para o usuário com algumas falhas; quando isso ocorre, os custos e o tempo para solucionar tais problemas são grandes, muitas vezes sendo necessário alocar profissionais de outros projetos para a realização da manutenção do sistema.

❑ **O esforço para construir software confiável e de qualidade é muito grande.**

❑ **O suporte aos programas existentes é apoiado por projetos pobres e recursos inadequados.**

Uma vez que boa parte dos projetos foi desenvolvida sem uma base sólida de qualidade de software, em virtude do tempo em que seu projeto inicial foi criado, e pelas sucessivas manutenções realizadas sem a devida documentação, o suporte a esses projetos fica comprometido.

Pressmann (2011, p. 30) diz que hoje uma enorme indústria de software tornou-se um fator dominante na economia mundial, com equipes de especialistas em software, cada qual concentrando-se numa parte da tecnologia necessária para distribuir uma aplicação complexa, havendo a substituição do programador solitário para um mais atuante e ligado com as novas tecnologias. Porém, as questões levantadas por esse programador solitário ainda continuam as mesmas feitas hoje, quando no desenvolvimento de novos e modernos sistemas computacionais:

- ❑ Por que concluir um software leva tanto tempo?
- ❑ Por que os custos de desenvolvimento são tão altos?
- ❑ Por que não conseguimos encontrar todos os erros antes de entregarmos o software aos clientes?
- ❑ Por que gastamos tanto tempo e esforço mantendo programas existentes?

Por que continuamos a ter dificuldade em medir o progresso enquanto o software está sendo desenvolvido e mantido?

Tom de Marco (1995 apud PRESSMANN, 2011, p. 31) afirma:

Em vez de perguntar por que software custa tanto, precisamos começar perguntando o que fizemos para tornar possível que o software atual custe tão pouco? A resposta a essa pergunta nos ajudará a continuarmos com o extraordinário nível de realização que sempre tem distinguido a indústria de software.

A preocupação em resolver essas questões tem levado à adoção das práticas da engenharia de software

## 1.2 Software

Pressmann (2011, p. 32) conceitua software como “(1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estruturas de dados que possibilitam ao programa manipular informações adequadamente; e (3) informação descritiva, tanto na forma impressa como na virtual, descrevendo a operação e o uso dos programas”.

Ainda conforme Pressmann (2011), o software é mais um elemento de sistema lógico do que físico e, dessa forma, possui características que são diferentes das do hardware, como mostra o Quadro 2.2.

Quadro 2.2 Características do hardware

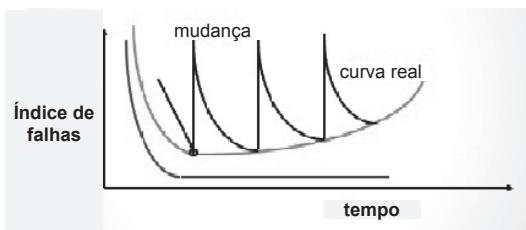
| Característica  | Descrição   |
|---|---|
| <b>Software é desenvolvido ou passa por um processo de engenharia, ele não é fabricado no sentido clássico.</b> | Mesmo havendo algumas similaridades entre o desenvolvimento de software e a fabricação de hardware, as duas atividades são diferentes. Pois, em ambas, alta qualidade é obtida por meio de bom projeto, mas a fase de fabricação de hardware pode gerar problemas de qualidade inexistentes ou facilmente corrigíveis para software. Também se verifica que ambas as atividades são dependentes de pessoas, mas a relação entre pessoas envolvidas e trabalho realizado é completamente diferente. Ambas requerem a construção de um “produto”, entretanto, as abordagens são diferentes. Os custos de software concentram-se no processo de engenharia, isso significa que projetos de software não podem ser geridos como se fossem projetos de fabricação. |

continua

continuação

|  |   |
|--|---|
| <p><b>Software não “se desgasta”.</b></p>  | <p>A Figura 2.1 mostra a taxa de defeitos em função do tempo para hardware. Essa relação indica que o hardware apresenta taxas de defeitos relativamente altas no início de sua vida devido a falhas de projeto ou de fabricação e, uma vez corrigidos esses efeitos, a taxa cai para um nível estável (felizmente, bastante baixo) por certo período. Resumindo, o hardware começa a desgastar-se. Já o software não é suscetível aos maus ambientais que fazem com que o hardware se desgaste. Porém a curva da taxa de defeitos para software deveria assumir a forma da “curva idealizada”, conforme a Figura 2.2. A curva idealizada é uma simplificação grosseira de modelos de defeitos reais para software. Mas a implicação é clara: software não se desgasta, mas sim se <i>deteriora</i>, pois, durante sua vida, passará por várias alterações e, à medida que estas ocorram, é provável que sejam introduzidos erros, fazendo com que a curva de taxa de defeitos se acentue, conforme mostrado na “curva real”. Outro aspecto de desgaste ilustra a diferença entre hardware e software: quando um componente de hardware se desgasta, ele é substituído por uma peça de reposição. Não existem peças de reposição de software. Cada defeito de software indica um erro no projeto ou no processo pelo qual o projeto foi traduzido em código de máquina executável. Portanto, as tarefas de manutenção de software, que envolvem solicitações de mudanças, implicam complexidade consideravelmente maior do que a de manutenção de hardware.</p> |
| <p><b>Embora a indústria caminhe para a construção com base em componentes, a maioria dos softwares continua a ser construída de forma personalizada, sob encomenda.</b></p> | <p>À medida que a disciplina da engenharia evolui, uma coleção de componentes de projeto padronizados é criada. Os componentes reutilizáveis foram criados para que o engenheiro possa se concentrar nos elementos realmente inovadores de um projeto, isto é, nas partes que representam algo novo. No mundo do hardware, a reutilização de componentes é uma parte natural do processo de engenharia. Já quando falamos no mundo do software, é algo que, em larga escala, apenas começou a ser alcançado, pois um componente de software deve ser projetado e implementado de modo que possa ser reutilizado em muitos programas diferentes. Os modernos componentes reutilizáveis encapsulam tanto dados quanto o processamento aplicado a eles, possibilitando criar novas aplicações a partir de partes reutilizáveis.</p>  |

Fonte: Adaptado de Pressmann (2011, p. 32-34).

**Figura 2.1 Curva de defeito de hardware**

Fonte: Adaptada de Pressmann (2011, p. 32).

**Figura 2.2 Curva de defeito de software**

Fonte: Pressmann (2011, p. 33).

Pressmann (2011) ainda subdivide o campo de aplicação do software em categorias que apresentam grandes desafios para os engenheiros de software, conforme podemos ver no Quadro 2.3.

**Quadro 2.3 Categorias de software**

| Categoria                                | Descrição   |
|--|---|
| <b>Software de sistema</b>               | É um conjunto de programas feito para atender a outros programas (compiladores, editores, utilitários etc.) que processam complexas estruturas de informação. Já outras aplicações de sistema (componentes de sistema operacional, drivers, software de rede etc.) processam dados amplamente indeterminados.   |
| <b>Software de aplicação</b>             | São programas desenvolvidos sob medida que solucionam uma necessidade específica de negócio do cliente.   |
| <b>Software científico/de engenharia</b> | São caracterizados por possuir algoritmos para processamento numérico pesado ( <i>number crunching</i> ). Porém, modernas aplicações na área de engenharia/científica estão se afastando dos algoritmos numéricos convencionais, estão sendo usados para projetos com o auxílio de computador, simulação de sistemas e outras aplicações interativas que possuam características de sistemas em tempo real e até mesmo de software de sistemas. |
| <b>Software embutido</b>                 | São aplicações que residem num produto ou sistema, usa-se para controlar e implementar características e funções tanto para o usuário final como para o próprio sistema. Executa funções limitadas e específicas (controle do painel de um forno micro-ondas, por exemplo) ou fornece função significativa e capacidade de controle (como funções digitais de automóveis).  |

continua

continuação

|  |  |
|--|--|
| <b>Software para linhas de produtos</b>    | São softwares projetados para prover um necessidade específica para o uso de muitos clientes diferentes. Pode ser dirigido a um mercado limitado e particularizado (como software para controle de estoques) ou ser dirigido a mercados de consumo de massa (processamento de texto, planilhas eletrônicas, computação gráfica, multimídia, entretenimento, gerenciamento de bancos de dados e aplicações financeiras pessoais e comerciais).  |
| <b>Aplicações Web</b>                      | Também conhecidas como “WebApps”, e, em sua forma mais simples, as WebApps podem ser pouco mais que um conjunto de arquivos de hipertexto interconectados, apresentando informações por meio de texto e informações gráficas limitadas. Mas, com o aparecimento da Web 2.0, essas aplicações evoluíram e se transformaram em sofisticados ambientes computacionais que não apenas fornecedores de recursos especializados, de funções computacionais e de conteúdo para o usuário final, mas também estão integradas a bancos de dados corporativos e aplicações comerciais. |
| <b>Software de inteligência artificial</b> | Usa algoritmos não numéricos para solucionar problemas complexos que não são passíveis de computação ou de análise direta. As aplicações de IA se destinam às áreas de robótica, sistemas especialistas, reconhecimento de padrões (de imagem e de voz), redes neurais artificiais, prova de teoremas, jogos etc.  |

Fonte: Adaptado de Pressmann (2011, p. 34-35).

### 1.2.1 Qualidades do software

Como qualquer produto, o software deve ter qualidade, mas várias são as qualidades do software a serem avaliadas, sendo necessário examinar tanto a qualidade do produto em si como a do processo de desenvolvimento. No Quadro 2.4 a seguir descrevemos algumas das qualidades que podem ser avaliadas.

Quadro 2.4 Qualidades a serem avaliadas em um software

| Qualidade        | Descrição  |
|------------------|--|
| <b>Corretude</b> | Um software precisa funcionar corretamente. Um software correto é aquele que satisfaz a sua especificação e que não possui falhas ou erros.                    |
| <b>Validade</b>  | Um software válido é aquele cuja especificação satisfaz aos requisitos dos usuários e da organização, isto é, está de acordo com as necessidades dos usuários. |
| <b>Robustez</b>  | O software deve prever que o usuário pode agir de forma não esperada e deve ser capaz de resistir a essas eventuais situações incomuns sem apresentar falhas.  |

continua

continuação

|                           |  |
|---------------------------|--|
| <b>Confiabilidade</b>     | Um software correto e robusto ganha a confiança dos usuários, uma vez que ele deve se comportar como esperado e não falha em situações inesperadas.  |
| <b>Eficiência</b>         | O software deve realizar suas tarefas em um tempo adequado à complexidade de cada uma delas. A utilização dos recursos de hardware (memória, disco, tráfego de rede) também deve ser feita de forma eficiente.   |
| <b>Usabilidade</b>        | O software precisa ser fácil de aprender e de usar, permitir maior produtividade do usuário, flexibilidade de utilização, flexibilidade de aplicação e proporcionar satisfação de uso.   |
| <b>Manutenibilidade</b>   | Todo software precisa de manutenção, seja para corrigir erros ou atender a novos requisitos. O software deve ser fácil de manter para que essas correções ou atualizações sejam feitas com sucesso.  |
| <b>Evolutibilidade</b>    | Todo software precisa evoluir para atender a novos requisitos, para incorporar novas tecnologias ou para expansão de sua funcionalidade.   |
| <b>Portabilidade</b>      | O software deve poder ser executado no maior número possível de equipamentos de hardware.  |
| <b>Interoperabilidade</b> | Softwares em diferentes plataformas devem poder interagir entre si. Essa qualidade é essencial em sistemas distribuídos, uma vez que o software pode estar sendo executado em diferentes computadores e sistemas operacionais. É interessante que diferentes elementos de softwares distintos possam ser utilizados em ambos. Por exemplo, certo arquivo com uma imagem feita num aplicativo deve poder ser vista em outros aplicativos. |
| <b>Reusabilidade</b>      | Diversos componentes de um software devem poder ser reutilizados por outras aplicações. O reúso de funções e objetos facilita bastante o desenvolvimento de software.  |

Fonte: Do autor (2014).

### 1.3 Engenharia de software

Pressmann (2011) apresenta alguns fatos reais que os desenvolvedores de softwares devem enfrentar no século XXI:

1. A incorporação do software no nosso dia a dia traz como consequência um número de pessoas interessadas nos recursos e nas funções oferecidas pelos softwares. Quando no desenvolvimento de um software, muitos usuários devem ser ouvidos, pois cada um deles possui pontos de vista (ideias) diferentes de funções, recursos e dispositivos que o software deve incorporar. Assim sendo, desprenda-se do conceito de que a engenharia é “um esforço concentrado para compreender o problema antes de desenvolver uma solução de software”. Dessa forma *devemos fazer um esforço concentrado para compreender o problema antes de desenvolver uma solução*.
2. Boa parte dos requisitos de TI (Tecnologia de Informação) são solicitados por indivíduos, empresas e órgãos governamentais e estão se tornando cada vez

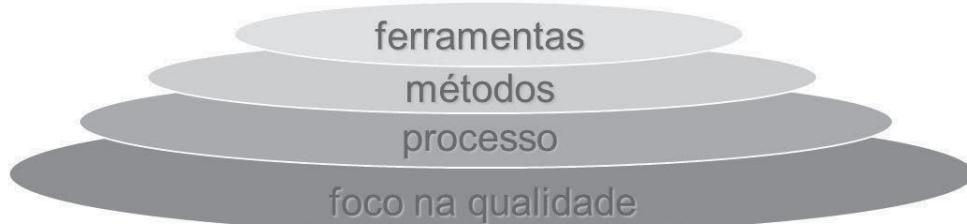
mais complexos a cada dia. Atualmente, o desenvolvimento de sistemas, que antes era realizado por várias pessoas, é realizado por um único indivíduo. Antes, um software sofisticado era implementado de forma independente e previsível em um ambiente computacional; hoje, está incorporado em tudo, desde produtos eletrônicos de consumo a equipamentos médicos e sistemas de armamentos, em que a complexidade desses novos produtos e sistemas exige uma maior atenção para com as interações de todos os elementos do sistema. Então, conclui-se que *projetar tornou-se uma atividade fundamental*.

3. A cada dia que passa nós — pessoas, empresas, governo — estamos mais dependentes de um software para decisões estratégicas e táticas, bem como para o controle das atividades do dia a dia e, dessa maneira, se o software falhar, nós poderemos vivenciar desde pequenos inconvenientes até falhas catastróficas. Assim sendo *um software deve apresentar qualidade elevada*.
4. À medida que a importância de um software específico aumente, a probabilidade é de que o número de usuários e seu tempo de vida também cresçam. Com esse crescimento, tanto de usuários como da longevidade, a demanda por manutenção também crescerá, por isso *um software deve ser passível de manutenção*.

Apenas com essas simples constatações podemos concluir que:

*"Um software, em todas as suas formas e em todos os seus campos de aplicação, deve passar pelos processos de engenharia"*

**Figura 2.3 Camadas da engenharia de software**



Fonte: Adaptada de Pressmann (2011, p. 39).

Bauer (apud PRESSMANN, 2011, p. 29) define a engenharia de software como “o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira económica, que seja confiável e funcione de forma eficiente em máquinas reais” .

Já o IEEE — Institute of Electrical and Electronic Engineers — (apud PRESSMANN, 2011, p. 29) define engenharia de software como “a aplicação e o estudo de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software”.

Com base nessas definições, Figura 2.3, Pressmann (2011) afirma que a engenharia de software é uma tecnologia em camadas, e refere que ela deve estar fundamentada num compromisso com a qualidade. A sua gestão é a pedra fundamental que sustenta a engenharia de software é o foco na qualidade. Já a base para a engenharia de software é a camada de *processos*, ou seja, é a liga que mantém as camadas de tecnologia unidas e possibilita o desenvolvimento de software de forma racional e dentro do prazo. Dessa maneira, o processo define uma *metodologia* que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de software constituindo a base para o controle do gerenciamento de projetos de software, estabelecendo o contexto de onde métodos técnicos serão aplicados e, quais modelos, documentos, dados, relatórios, formulários serão produzidos e, também com o estabelecimento de marcos, a qualidade é garantida e mudanças são geridas de forma apropriada. Os *métodos* fornecem as informações técnicas para desenvolver software, envolvendo uma ampla gama de tarefas, que incluem a comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Os métodos se baseiam em um conjunto de princípios básicos que governam cada área da tecnologia e inserem atividades de modelagem e outras técnicas descritivas. Já *ferramentas* fornecem suporte automatizado ou semiautomatizado para o processo e para os métodos e, uma vez integradas, de modo que as informações criadas por uma ferramenta possam ser usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de software, denominado *engenharia de software com o auxílio do computador (CASE)*.



### Para saber mais

No intuito de fixar melhor seus conhecimentos sobre processos de software, acesse os links:

<<http://www.devmedia.com.br/processos-de-software/21977>>.

<<http://www.devmedia.com.br/introducao-aos-processos-de-software-e-o-modelo-incremental-e-evolucionario/29839>>.

#### 1.3.1 Processo de software

É um conjunto de atividades, ações e tarefas realizadas na criação de algum projeto de trabalho (*workproduct*). Uma *atividade* tem por fim atingir um objetivo e é utilizada independentemente da aplicação, do tamanho e da complexidade do projeto, dos esforços ou do grau de rigor com que a engenharia de software será aplicada. Uma *ação* envolve um conjunto de tarefas que resultam num artefato de software. Já uma *tarefa* se concentra em um objetivo pequeno, porém bem definido e produz um resultado tangível.

Na engenharia de software, um processo *não* é uma prescrição rígida de como desenvolver um software, mas sim uma abordagem adaptável que possibilita às pessoas realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas.

A intenção é a de sempre entregar software dentro do prazo, a um custo e com qualidade suficiente para satisfazer àqueles que patrocinaram sua criação e àqueles que vão utilizá-lo.

Uma metodologia de processo (*framework*) estabelece a base para um processo de engenharia de software completo, por meio da identificação de um pequeno número de atividades estruturais aplicáveis a todos os projetos, independentemente de tamanho ou complexidade. Além disso, a metodologia de processo engloba um conjunto de atividades de apoio, também conhecidas como atividades guarda-chuva, aplicáveis em todo o processo de software; uma metodologia de processo genérica para engenharia de software compreende cinco atividades: **comunicação, planejamento, modelagem, construção e emprego**.

Essas atividades metodológicas genéricas podem ser utilizadas tanto para o desenvolvimento de programas pequenos e simples, como para a criação de grandes aplicações para a internet e para a engenharia de grandes e complexos sistemas baseados em computador. Os detalhes do processo de software poderão ser bem diferentes em cada caso, porém as atividades permanecerão as mesmas.

Para muitos projetos de software, as atividades metodológicas são aplicadas iterativamente conforme o projeto se desenvolve. Ou seja, **comunicação, planejamento, modelagem, construção e emprego** são aplicados repetidamente quantas forem as iterações do projeto, sendo que cada iteração produzirá um *incremento* de software. Este disponibilizará uma parte dos recursos e funcionalidades do software. A cada incremento, o software torna-se mais e mais completo.

As atividades lógicas do processo de engenharia de software são complementadas por uma série de *atividades de guarda-chuva* (Quadro 2.5), geralmente aplicadas ao longo de um projeto, auxiliando a equipe a gerenciar, a controlar o progresso, a qualidade, as mudanças e os riscos.

**Quadro 2.5 Atividades guarda-chuva típicas**

| Atividade  | Descrição   |
|--|---|
| <b>Controle e acompanhamento do projeto</b>      | Possibilita que a equipe avalie o progresso em relação ao plano do projeto e tome as medidas necessárias para cumprir o cronograma.   |
| <b>Administração de riscos</b>                   | Avalia riscos que possam afetar o resultado ou a qualidade do produto/projeto.  |
| <b>Garantia da qualidade de software</b>         | Define e conduz as atividades que garantem a qualidade do software.   |
| <b>Revisões técnicas</b>                         | Avaliam artefatos da engenharia de software, tentando identificar e eliminar erros antes que se propaguem para a atividade seguinte.  |
| <b>Medição</b>                                   | Define e coleta medidas (do processo, do projeto e do produto). Auxilia na entrega do software de acordo com os requisitos; pode ser usada com as demais atividades (metodológicas e de apoio). |
| <b>Gerenciamento da configuração de software</b> | Gerencia os efeitos das mudanças ao longo do processo.  |

continua

continuação

|   |  |
|---|--|
| <b>Gerenciamento da recusabilidade</b>              | Define critérios para o reúso de artefatos (inclusive componentes de software) e estabelece mecanismos para a obtenção de componentes reutilizáveis. |
| <b>Preparo e produção dos artefatos de software</b> | Engloba as atividades necessárias para criar artefatos, por exemplo, modelos, documentos, logs, formulários e listas.                                |

Fonte: Adaptado de Pressmann (2011, p. 41-42).

## 1.4 Crise do software

Em meados dos anos 1970, surgiu o termo “crise do software”, quando a engenharia de software praticamente inexistia, e estava relacionada às dificuldades enfrentadas no desenvolvimento de software, pelo aumento das demandas e da complexidade delas, aliado à inexistência de técnicas apropriadas para resolver tais desafios.

A crise se referia aos softwares desenvolvidos na época, que apresentavam em sua maioria uma qualidade inferior e custos superiores ao previsto: de fato, entre 50% e 80% dos softwares desenvolvidos não apresentavam as configurações desejadas e cerca de 90% tinham seu custo final entre 150% a 400% maior que o previsto.

Esses fatos fizeram aparecer o conceito de crise do software, podendo ser verificada por vários sintomas, tais como:

- software de baixa qualidade;
- projetos com prazos e custos maiores que os planejados;
- software não atendendo aos requisitos dos *stakeholders*;
- custos e dificuldades no processo de manutenção.



### Para saber mais

Aprofundando seu conhecimento, assista ao vídeo:

<[www.youtube.com/watch?v=EbTo14jSJ6Y](https://www.youtube.com/watch?v=EbTo14jSJ6Y)>.

## 1.5 Causas da crise de software

Com base nos sintomas expostos e no que vemos hoje, podemos chegar à conclusão de que a crise ainda está presente. Apesar de dispormos de técnicas apropriadas, ainda presenciamos esses problemas hoje e em um futuro próximo. Mesmo as empresas que têm conhecimento das técnicas propostas às vezes não conseguem praticá-las por pressão do próprio cliente, por exemplo, em relação a prazos, custos e qualidade.

Podemos dizer ainda que a crise se refere a um conjunto de problemas encontrados no desenvolvimento de software, tais como:

**1. As estimativas de prazo e de custo são frequentemente imprecisas**

Pelo fato de não dedicarmos tempo para coletar dados sobre o processo de desenvolvimento de software e também por não haver nenhuma indicação sólida de produtividade, não podemos avaliar com precisão a eficácia de novas ferramentas, métodos ou padrões.

**2. A produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços**

Os projetos de desenvolvimento de software normalmente são efetuados apenas com um vago indício das exigências do cliente.

**3. A qualidade de software às vezes é menos que adequada**

Só recentemente começam a surgir conceitos quantitativos sólidos de garantia de qualidade de software.

**4. O software existente é muito difícil de manter**

A tarefa de manutenção devora o orçamento destinado ao software, uma vez que a facilidade de manutenção não foi enfatizada como um critério importante.

Na atualidade, também encontramos uma crise de desenvolvimento de software que, além de englobar os sintomas relacionados à crise do software, está relacionada com os pontos a seguir:

- └ grande insatisfação de clientes;
- └ custos relativos ao desenvolvimento de sistemas aumentando, enquanto os relativos a hardware diminuem;
- └ prazo de desenvolvimento de software tornando-se mais longo enquanto os custos de manutenção tornam-se maiores;
- └ erros de software tornando-se mais frequentes, refletindo em custos de manutenção e de retrabalho;
- └ processos estruturados de desenvolvimento sendo inflexíveis e ainda muito utilizados;
- └ desenvolvimento de aplicativos/sistemas com alto custo, baixa qualidade e com demanda cada vez maior, aliados a uma complexidade também cada vez maior;
- └ pressão das empresas clientes para entregas rápidas dos sistemas.

## 1.6 Consequências da crise de software

Uma das consequências da crise foi o surgimento de diversos métodos de desenvolvimento de software, os quais, por sua grande quantidade, caracterizaram-se por se tornar mais um problema ante a inexistência de um método genérico e confiável. Outro aspecto está ligado à aceitação desses novos métodos tanto por parte dos desenvolvedores como das empresas produtoras de software, uma vez que, pela imaturidade no mercado, eles dificilmente são incorporados no dia a dia por causa da pouca confiabilidade que os desenvolvedores têm em relação aos mesmos, preferindo usar métodos antigos, porém que têm eficiência garantida pela experiência de uso.

Contudo a maior consequência da crise foi o surgimento da engenharia de software, a qual visa ao desenvolvimento de tecnologias que facilitem e permitam a obtenção de softwares eficientes e baratos; sua importância a fez tornar-se um novo campo de pesquisa na computação.

## 1.7 Causas dos problemas associados à crise de software

Dentre as causas dos problemas que levaram à crise do software podemos destacar:

1. O próprio caráter do software: uma vez que o software é um elemento de sistema lógico e não físico, seu sucesso é medido pela quantidade de **uma única entidade** e não pela quantidade de entidades manufaturadas (implantadas).
2. Falhas das pessoas responsáveis pelo desenvolvimento de software: gerentes sem nenhum *background*, falta de treinamento para os profissionais da área em novas técnicas para o desenvolvimento de software e também pela resistência à mudanças, tanto por parte dos clientes como por parte dos desenvolvedores.
3. Mitos do software que propagaram desinformação e confusão e estão subdivididos em mitos *administrativos, de clientes e de profissionais*.

## 1.8 Mitos relativos ao software

Algumas crenças (mitos) infundadas em relação aos softwares e ao processo usado para criá-los remontam aos primórdios da computação. As crenças, também chamadas de mitos, possuem uma série de atributos que as tornam uma cilada, por exemplo, esses atributos parecem ser afirmações razoáveis e têm uma sensação intuitiva e frequentemente são promulgados por praticantes profissionais que se julgam experientes.

Hoje, boa parte dos engenheiros de software reconhece os mitos por aquilo que eles representam, ou seja, atitudes enganosas que provocaram sérios problemas tanto para gerentes quanto para os profissionais da área. Porém, antigos hábitos e atitudes são difíceis de ser modificados e resquícios de mitos de software permanecem. Os mitos podem ser classificados como mitos administrativos, de clientes e profissionais.

### 1.8.1 Mitos administrativos

A pressão para manter orçamentos, cumprir os cronogramas e elevar a qualidade está levando tanto os gerentes de software como de outras áreas a se agarrar à crença num mito de software para aliviar tal pressão, mesmo que temporariamente.

O Quadro 2.6 mostra alguns mitos que os gerentes enfrentam no desenvolvimento de software.

**Quadro 2.6 Mitos administrativos**

| Mitos  | Realidade  |
|--|--|
| <b>Uma vez que temos um manual cheio de padrões e procedimentos para desenvolver software, então ele é suficiente para o meu pessoal com tudo que eles precisam saber.</b> | Um manual pode até existir, porém, é usado? As pessoas da área sabem que ele existe? Este manual reflete a prática moderna da engenharia de software? Ele é completo? É adaptável? Está alinhado para melhorar o tempo de entrega, mantendo ainda o foco na qualidade? Em muitos casos, a resposta para todas essas perguntas é “não”. |
| <b>Se o cronograma atrasar, é só colocar mais programadores e ficarmos em dia (também chamado de conceito da “horda mongol”).</b>  | O desenvolvimento de software não é um processo mecânico como o de fábrica, pois acrescentar pessoas num projeto atrasado só o tornará mais atrasado ainda. Podem-se adicionar pessoas ao projeto, desde que seja de forma planejada e bem coordenada.   |
| <b>Se terceirizar o projeto de software, posso simplesmente relaxar e deixar essa empresa realizá-lo.</b>  | Se uma empresa não souber gerenciar e controlar seus projetos de software, provavelmente enfrentará dificuldades ao terceirizá-los.  |
| <b>Meu pessoal tem ferramentas de desenvolvimento de software de última geração; afinal compramos os mais novos computadores.</b>  | É preciso muito mais do que os mais recentes computadores para se fazer um desenvolvimento de software de alta qualidade.  |

Fonte: Adaptado de Pressman (2011, p. 46).

### 1.8.2 Mitos dos clientes

O usuário de um software pode ser alguém ao seu lado, um grupo técnico, um departamento da empresa ou mesmo uma empresa externa que encomendou o projeto por contrato. Na maioria das vezes, o cliente acredita em mitos sobre o software porque gerentes e profissionais da área pouco fazem para corrigir falsas informações. Mitos esses que conduzem a falsas expectativas (do cliente) e, em última instância, à insatisfação com o desenvolvedor, como podemos observar no Quadro 2.7.

**Quadro 2.7 Mitos dos clientes**

| Mitos  | Realidade  |
|--|--|
| <b>É preciso muito mais do que os mais recentes computadores para se fazer um desenvolvimento de software de alta qualidade.</b>           | Uma definição inicial ruim é a principal causa de fracassos dos esforços de desenvolvimento de software.<br>É fundamental uma descrição formal e detalhada do domínio da informação, função, desempenho, interfaces, restrições de projeto e critérios de validação.   |
| <b>Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível.</b> | É verdade que os requisitos de software mudam, mas o impacto da mudança varia dependendo do momento em que ela foi introduzida. Quando as mudanças dos requisitos são solicitadas antes de o projeto ou a codificação terem começado, o impacto sobre os custos é relativamente pequeno. Porém, conforme o tempo passa, os recursos foram comprometidos, uma estrutura de projeto já foi estabelecida e mudar isso pode causar uma revolução que exija recursos adicionais e modificações fundamentais no projeto. |

Fonte: Adaptado de Pressmann (2011, p. 46).

### 1.8.3 Mitos dos profissionais

Mitos que ainda sobrevivem entre os profissionais da área têm resistido por mais de 50 anos de cultura de programação. Durante seus primórdios, a programação era vista como uma forma de arte, pois modos e atitudes antigos dificilmente morrem. No Quadro 2.8 podemos verificar alguns mitos profissionais.

Quadro 2.8 Mitos dos profissionais

| Mitos  | Realidade  |
|--|--|
| <b>Assim que escrevermos o programa e o colocarmos em funcionamento nosso trabalho estará completo.</b>                    | Dados da indústria de software indicam que entre 60% e 80% de todo o esforço será despendido após a entrega do software pela primeira vez ao cliente.  |
| <b>Até que o programa entre em funcionamento, não há maneira de avaliar sua qualidade.</b>                                 | Um programa funcionando é somente uma parte de uma configuração de software que inclui todos os itens de informação produzidos durante a construção e manutenção do software.  |
| <b>O único produto passível de entrega é o programa em funcionamento.</b>  | Um programa funcionando é somente uma parte de uma configuração de software que inclui muitos elementos. Uma variedade de produtos derivados (por exemplo, modelos, documentos, planos) constitui uma base para uma engenharia bem-sucedida e, mais importante, uma orientação para suporte de software. |
| <b>A engenharia de software nos fará criar documentação volumosa e desnecessária e, invariavelmente, irá nos retardar.</b> | A engenharia de software não trata de criação de documentos, trata da criação de um produto de qualidade. Melhor qualidade conduz à redução do retrabalho, e menos retrabalho resulta em maior rapidez na entrega.   |

Fonte: Adaptado de Pressmann (2011, p. 47).

Muitos profissionais de software reconhecem os enganos sobre os mitos que acabamos de descrever. Mas, infelizmente, métodos e atitudes habituais estimulam tanto o gerenciamento quanto as medidas técnicas deficientes, mesmo quando a realidade exige uma abordagem mais eficiente. Ter consciência das realidades do software é o primeiro passo para buscar soluções práticas na engenharia de software.



#### Questões para reflexão

Analisando a evolução do software bem como da engenharia de software, reflita se você está preparado para fazer a aplicação da engenharia de software em suas atividades cotidianas.

Analisando a crise do software bem como as suas causas, reflita se os problemas que ocorreram na década de 1970 não são os mesmos encontrados hoje.



## *Atividades de aprendizagem*

1. Cite e explique duas atividades guarda-chuva.
2. Quais são as camadas da engenharia de software?
3. Cite e explique pelo menos um mito administrativo, de cliente e profissional.

## Seção 2 Modelos de processos de software

Um processo de software é um *framework* (conjunto de atividades) que leva à produção de um produto de software de qualidade, sendo que essas atividades podem envolver o próprio desenvolvimento de software, fazendo uso de uma linguagem de programação. No entanto, cada vez mais, o novo software é desenvolvido com a ampliação e/ou modificação de sistemas já existentes, de configuração e integração de software comercial ou de um componente de sistema.

Como todos os processos intelectuais e criativos, os processos de software são complexos e dependem de julgamento humano. E pela necessidade de utilizar o julgamento e a criatividade, as tentativas de automatização desses processos têm um sucesso limitado, pois as ferramentas CASE (Computer-Aided Software Engineering) podem apoiar apenas algumas atividades de processo. Entretanto não existe possibilidade no momento de uma automação mais extensa, em que o software assuma o projeto criativo, liberando os engenheiros envolvidos no processo.

A limitação das ferramentas CASE vem da imensa gama de processos de software. Não há um processo ideal, uma vez que várias organizações desenvolveram abordagens inteiramente diferentes para o desenvolvimento de software.

Os processos de desenvolvimento evoluíram no intuito de explorar as capacidades das pessoas em uma empresa e também as características específicas dos sistemas que estão sendo desenvolvidos. No caso dos sistemas críticos, é necessário um processo de desenvolvimento muito mais estruturado. Já nos sistemas de comerciais, em que os requisitos mudam rapidamente, um processo flexível e ágil é provavelmente mais eficaz.

Embora existam vários tipos de processos de software, algumas atividades fundamentais são comuns a todos eles, tais como:

1. *Especificação de software*: A funcionalidade do software e as restrições sobre sua operação devem ser definidas.
2. *Projeto e implementação de software*: deve atender à especificação deve ser produzido.
3. *Validação de software*: deve ser validado para garantir que faça o que o cliente deseja.
4. *Evolução de software*: deve evoluir para atender às necessidades mutáveis do cliente.

Essas atividades fazem parte de todo processo de software, são atividades complexas que incluem subatividades, como validação de requisitos, projeto de arquitetura, testes unitários, teste de integração e teste de carga.

- **Validação de requisitos** — é uma atividade pela qual se verifica se os requisitos definem o sistema que o cliente realmente quer.

- ❑ **Projeto de arquitetura** — é o primeiro estágio no processo de projeto de software e está preocupado com a compreensão de como um sistema deve ser organizado e com a estrutura geral desse sistema.
- ❑ **Teste unitário** — é o processo de testar os componentes do programa, como métodos ou classes de objeto.
- ❑ **Teste de integração** — envolve a integração de componentes para criação de uma versão do sistema e, em seguida, o teste do sistema integrado.
- ❑ **Teste de carga** — deve ser projetado para assegurar que o sistema possa processar a carga a que se destina.

Há atividades de extrema importância que dão apoio ao processo, como as de documentação e gerenciamento de configuração, ao descrever e discutir os processos, como a especificação de um modelo de dados, o projeto de interface de usuário bem como a organização dessas atividades. No entanto, assim como as atividades, as descrições do processo também podem incluir:

- ❑ **Produtos:** São os resultados de várias atividades do processo.
- ❑ **Papéis:** Refletem as responsabilidades das pessoas envolvidas no processo.
- ❑ **Pré e pós-condições:** São declarações verdadeiras antes e depois de uma atividade do processo ou da produção de um produto.

Embora não exista um processo de software “ideal”, ainda há espaço para aprimoramento do processo de software em várias organizações. Os processos podem incluir técnicas obsoletas e/ou não tirar vantagem das melhores práticas na engenharia de software. De fato, muitas empresas ainda não se beneficiam dos métodos de engenharia de software em seu desenvolvimento de software, pois boa parte delas acha muito difícil sua implementação e dessa forma não faz o uso delas.

Os processos de software podem ser aprimorados pela padronização, onde a diversidade de processos de software ao longo da organização é pequena e por esse motivo, promove o aprimoramento da comunicação e redução no tempo de treinamento e faz que o apoio ao processo automatizado seja mais econômico.

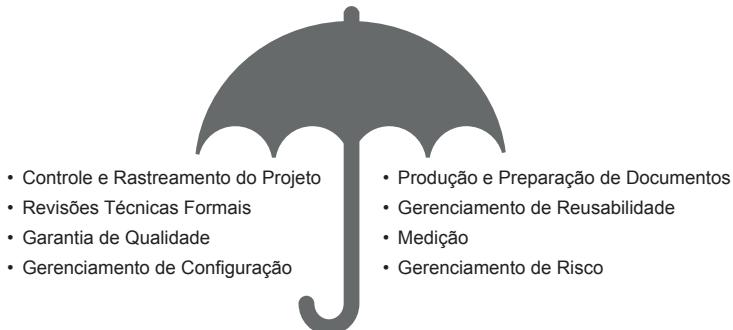
A padronização é também um passo inicial importante na introdução de novos métodos e técnicas de engenharia de software e das boas práticas de engenharia de software.

Um processo foi definido como um conjunto de atividades de trabalho, ações e tarefas realizadas quando algum artefato de software deve ser criado. Cada uma dessas atividades, ações e tarefas alocam-se dentro de uma metodologia ou modelo que determina seu relacionamento com o processo e seu relacionamento umas com as outras. Cada atividade do processo de software é composta por um conjunto de ações de engenharia de software. Cada ação é definida por um *conjunto de tarefas*, o qual identifica as tarefas de trabalho a ser completadas, os artefatos de software que serão produzidos, os fatores de garantia da qualidade que serão exigidos e os marcos utilizados para indicar progresso.

Uma metodologia de processo genérica para engenharia de software estabelece cinco atividades metodológicas: comunicação, planejamento, modelagem, construção

e entrega. Além disso, um conjunto de atividades de apoio (*atividades guarda-chuva*), conforme a Figura 2.4, são aplicadas ao longo do processo, como o acompanhamento e controle do projeto, a administração de riscos, a garantia da qualidade, o gerenciamento das configurações, as revisões técnicas e outras.

Figura 2.4 Atividades guarda-chuva



Fonte: Do autor (2014).



### Para saber mais

Para aprimorar seus conhecimentos sobre o assunto, acesse e leia:

<<http://www.dominiopublico.gov.br/download/texto/cp010140.pdf>>

e

<<http://www.dominiopublico.gov.br/download/texto/cp044363.pdf>>.

## 2.1 Modelos ágeis

O desenvolvimento ágil é um fruto da constatação feita por diversos profissionais na área de engenharia de software, de que, apesar de terem aprendido segundo a cartilha tradicional, só conseguiam minimizar os riscos associados ao desenvolvimento de software pensando e agindo de forma muito diferente do que tradicionalmente está nos livros.

Embora cada envolvido tivesse sua própria prática e/ou teorias preferidas, todos concordavam que em suas experiências prévias os projetos de sucesso tinham em comum um pequeno conjunto de princípios. Com base nisso eles criaram o **Manifesto para o Desenvolvimento Ágil de Software**, frequentemente chamado apenas de Manifesto Ágil.

No desenvolvimento ágil, os projetos adotam o modelo iterativo e em espiral de desenvolvimento, onde todas as fases descritas no modelo em cascata (análise, design, implementação, testes, implantação e manutenção) são executadas diversas vezes ao longo do projeto, produzindo ciclos curtos que se repetem por todo o desenvol-

vimento, sendo que, ao final de cada ciclo, sempre se tem um software funcional, testado e aprovado.

Os ciclos são chamados de iterações e crescem em número de funcionalidades a cada repetição, sendo que, no último, todas as funcionalidades desejadas estarão implementadas, testadas e aprovadas.

### 2.1.1 Manifesto ágil de software

Segundo Pressmann (2011, p. 128), Kent Beck e 16 outros notáveis desenvolvedores, produtores e consultores de software (conhecidos como a “Aliança Ágil”) assinaram o **Manifesto para o Desenvolvimento Ágil de Software**, onde declararam:

[...] estamos descobrindo melhores modos de desenvolvimento de software fazendo-o e ajudando outros a fazê-lo. Por meio desse trabalho passamos a valorizar: indivíduos e interações em vez de processos e ferramentas; softwares funcionando em vez de documentação abrangente; a colaboração do cliente em vez de negociação de contratos; a resposta a modificações em vez de seguir um plano. Isto é, ainda que haja nos itens à direita, valorizamos mais os itens à esquerda.

Um manifesto é normalmente associado com um movimento político emergente, isto é, um movimento que sugira mudanças revolucionárias e, de certa forma, isso é exatamente o que desenvolvimento ágil é.

Embora as ideias que guiam o desenvolvimento ágil existam há anos, somente na década de 1990 é que foram cristalizadas em um “movimento”. Essencialmente os métodos ágeis foram desenvolvidos num esforço para vencer as fraquezas detectadas e reais da engenharia de software convencional. O desenvolvimento ágil pode fornecer importantes benefícios, porém não é aplicável a todos os projetos, produtos, pessoas e situações. Ele também não é contrário à sólida prática de engenharia de software e pode ser aplicado como uma filosofia prevalecente a todo o trabalho de software.

Hoje é difícil ou impossível prever como um sistema baseado em computador evoluirá com o passar do tempo, pois as condições de mercado mudam rapidamente, necessidades dos usuários finais evoluem e as novas ameaças de competição surgem sem alerta. E em muitas situações, não há como definir completamente os requisitos antes do início do projeto, assim os engenheiros de software devem ser ágeis o suficiente para responder a um ambiente de negócios mutante.

Isso significa que um reconhecimento dessas causas realísticas modernas não nos obriga a descartar princípios, conceitos, métodos e ferramentas valiosos de engenharia de software, visto que, como todas as atividades de engenharia, a de software continua a evoluir, podendo ser adaptada facilmente para encarar os desafios colocados pela demanda por agilidade.

O que é exatamente agilidade no contexto do trabalho de engenharia de software? Na verdade é o acolhimento de modificações o principal guia para a agilidade. Os engenheiros de software devem reagir rapidamente se tiverem de acomodar as rápidas modificações, seja no software que está sendo construído, nos membros das equipes,

em modificações por causa de novas tecnologias. Modificações de todas as espécies que podem ter impacto no produto ou no projeto que cria o produto.

Mas agilidade é mais do que uma resposta efetiva à modificação. Ela também engloba a filosofia apresentada no “Manifesto para o Desenvolvimento Ágil de Software”. Encoraja estruturas e atitudes de equipe que tornam a comunicação mais fácil (entre membros da equipe, entre pessoal de tecnologia e de negócios, entre engenheiro de software e seus gerentes). Ela enfatiza a rápida entrega de software operacional e dá menos importância para produtos de trabalho intermediário; adota os clientes como parte da equipe de desenvolvimento e trabalha para eliminar a atitude “nós e eles” que continua a permear muitos projetos de software; ela reconhece que o planejamento em um mundo incerto tem seus limites e que um plano deve ser flexível. No Quadro 2.9 podemos verificar os princípios da Aliança Ágil.

**Quadro 2.9 Princípios da agilidade**

| Princípio | Descrição  |
|-----------|--|
| 1         | Nossa maior prioridade é satisfazer ao cliente desde o início por meio de entrega contínua de software valioso.  |
| 2         | Modificações de requisitos são bem-vindas, mesmo que tardias no desenvolvimento. Os processos ágeis aproveitam as modificações como vantagens para a competitividade do cliente. |
| 3         | Entrega de softwares funcionando frequentemente, a cada duas semanas até dois meses, de preferência no menor espaço de tempo.  |
| 4         | O pessoal do negócio e os desenvolvedores devem trabalhar juntos diariamente durante todo o projeto.   |
| 5         | Construção de projetos em torno de indivíduos motivados. Forneça-lhes o ambiente e apoio que precisam e confie que eles farão o trabalho.  |
| 6         | O método mais eficiente e efetivo de levar informação para dentro de uma equipe de desenvolvimento é conversa face a face.   |
| 7         | Software funcionando é a principal medida de progresso.  |
| 8         | Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante, indefinidamente.             |
| 9         | Atenção contínua a excelência técnica e ao bom projeto facilita a agilidade.   |
| 10        | Simplicidade — a arte de maximizar a quantidade de trabalho não efetuado — é essencial.  |
| 11        | As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.  |
| 12        | Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva, então sintoniza e ajusta adequadamente seu comportamento.   |

Fonte: Adaptado de Pressmann (2011, p. 84-85).

## 2.1.2 Características

O processo ágil de software é caracterizado de forma que atenda às seguintes suposições-chave sobre a maioria dos projetos de software:

- └ É difícil prever quais requisitos de software vão persistir e quais serão modificados. É igualmente difícil prever como as prioridades do cliente serão modificadas à medida que o projeto evolui.
- └ Como em vários tipos de software, o projeto e a construção são intercalados, ou seja, as duas atividades devem ser realizadas juntas de maneira que os modelos de projeto sejam comprovados à medida que são criados.
- └ Análise, projeto, construção e testes não são tão previsíveis (do ponto de vista do planejamento) como gostaríamos.

Dadas essas três suposições, pergunta-se então como criar um processo que possa gerenciar a imprevisibilidade? A resposta está na adaptabilidade do processo. Um processo ágil, portanto, deve ser adaptável. Assim, um processo ágil de software deve ser adaptado incrementalmente e para realizar a adaptação uma equipe ágil requer o *feedback* do cliente. Dessa forma, uma estratégia de desenvolvimento deve ser instituída, na qual os incrementos de software — protótipos executáveis ou partes um sistema operacional — devem ser entregues em curtos períodos de tempo de modo que a adaptação acerte o passo com as modificações. Essa abordagem iterativa habilita o cliente a avaliar o incremento de software regularmente, fornecer o *feedback* necessário à equipe e influenciar as adaptações do processo feitas para acomodar o *feedback*.

## 2.1.3 Ciclo de vida

O ciclo de vida do desenvolvimento dos métodos ágeis é praticamente igual aos demais métodos, porém com a diferença de que são executados ciclos mais rápidos, e estes executados diversas vezes. Sendo estas as etapas:

- └ definir a necessidade do cliente e iniciar o projeto;
- └ planejar o projeto, calculando o esforço;
- └ executar o plano, construindo a solução;
- └ monitorar resultados e entregar ao cliente;
- └ ciclos mais rápidos, executados diversas vezes.

## 2.1.4 Fatores humanos

Pressmann (2011, p. 86) comenta que os proponentes de desenvolvimento ágil de software sofrem muito para enfatizar a importância dos “fatores pessoais” no desenvolvimento ágil bem-sucedido. O ponto-chave dessa declaração é que o processo se molda às necessidades das pessoas e da equipe, e não o contrário. Se os membros de uma equipe de software tiverem de estabelecer as características do processo que é aplicado para construir softwares, uma certa quantidade de características-chave (Quadro 2.10) deve existir entre as pessoas de uma equipe ágil e na equipe em si.

**Quadro 2.10 Características-chave entre os componentes de uma equipe ágil**

| Característica                                | Descrição  |
|---|--|
| <b>Competência</b>                            | Em um contexto de desenvolvimento ágil, “competência” inclui talento nato, habilidades específicas relacionadas a software e conhecimento global do processo que a equipe decidiu aplicar. Habilidade e conhecimento do processo podem e devem ser ensinados a todas as pessoas que servem como membros de equipes ágeis.  |
| <b>Foco comum</b>                             | Embora os membros da equipe ágil possam realizar diferentes tarefas e trazer diferentes habilidades ao projeto, todos deveriam estar focados em uma meta — entregar um incremento de software em funcionamento ao cliente dentro do prazo prometido. Para atingir essa meta, a equipe também vai concentrar-se em adaptações contínuas (pequenas e grandes) que farão o processo satisfazer às necessidades da equipe.   |
| <b>Colaboração</b>                            | Engenharia de software (independentemente do processo) diz respeito a avaliar, analisar e usar as informações que são comuns à equipe de software, criar informações que ajudarão o cliente e outros a entender o trabalho da equipe, e construir informações (software de computador e banco de dados relevantes) que forneçam valor de negócios para o cliente. Para realizar essas tarefas da equipe precisam colaborar — uns com os outros, com o cliente e com os gerentes de negócios.                         |
| <b>Habilidade de resolver problemas vagos</b> | Os gerentes de software devem reconhecer que uma equipe ágil terá de lidar continuamente com ambiguidades e será continuamente confrontada por modificações. Em alguns casos a equipe precisa aceitar o fato de que o problema que está sendo resolvido hoje pode não ser o problema que precisará ser resolvido amanhã. No entanto, as lições aprendidas de qualquer atividade de solução de problema (inclusive aquelas que resolvem o problema errado) podem ser benéficas para a equipe mais adiante no projeto. |
| <b>Auto-organização</b>                       | <p>No contexto de desenvolvimento ágil, a auto-organização implica três coisas:</p> <ul style="list-style-type: none"> <li>└ A equipe ágil organiza-se para o trabalho a ser feito.</li> <li>└ A equipe organiza o processo para melhor acomodar seu ambiente local.</li> <li>└ A equipe organiza o cronograma de trabalho para conseguir melhor entrega do incremento de software.</li> </ul>   |

Fonte: Adaptado de Pressmann (2011, p. 86).

### 2.1.5 Modelos de processo ágeis

Dentro os modelos de processos ágeis, destacamos:

- └ XP — eXtreme Programming
- └ Scrum
- └ Método de Desenvolvimento de Sistemas Dinâmicos (DSDM)
- └ Crystal
- └ Desenvolvimento Dirigido a Funcionalidades (FDD)

### **2.1.5.1 XP — eXtreme Programming**

O método XP talvez seja o mais usado dos métodos de desenvolvimento ágeis. Esse método foi escrito por Kent Beck e recentemente foi refinado. Essa variação foi denominada IXP (Industrial XP) que tende a ser o processo ágil para grandes organizações usarem. É composto por um conjunto de cinco valores (**Comunicação, simplicidade, feedback, coragem e respeito**) para estabelecer as bases para todo trabalho realizado como parte da XP. Esses valores são usados como um direcionador das atividades específicas da XP.

No método XP, todos os requisitos são expressos como cenários, que são implementados diretamente como uma série de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes da escrita do código, assim exercitando cada operação de acordo com sua funcionalidade especificada. Sempre que um incremento é entregue para o cliente, os casos de uso são usados para base de testes de aceitação e geram uma forma de *feedback*. Conforme o surgimento de novas necessidades, rapidamente a equipe informa o cliente sobre o impacto nos custos e no cronograma do desenvolvimento do software.

Um preceito fundamental na engenharia de software é que você deve projetar para a mudança, antecipando melhorias futuras para o software e projetá-lo para que seja facilmente implementado. A maioria das equipes de software argumentam que “projetar para o amanhã” poupará tempo e esforços no longo prazo. Uma equipe XP ágil deve ter disciplina (coragem) para projetar hoje, reconhecendo que as necessidades futuras podem mudar dramaticamente exigindo, consequentemente, substancial retrabalho em relação ao projeto e ao código implementado.

A XP emprega uma abordagem orientada a objetos como seu paradigma de desenvolvimento preferido, envolvendo um conjunto de regras e práticas constantes no contexto de quatro atividades metodológicas: **planejamento, projeto, codificação e testes**.

### **2.1.5.2 Scrum**

O Scrum é definido como um framework no qual se podem tratar e resolver problemas complexos e adaptativos, enquanto entregam produtos com o mais alto valor possível; resumidamente, o Scrum é: leve, simples de entender e difícil de dominar.

O Scrum é um framework estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos, não um processo ou uma técnica para construir produtos e, sim, uma ferramenta na qual se pode employar vários processos ou técnicas. Consiste em times associados a papéis, artefatos, eventos e regras, em que cada componente dentro do framework serve para um propósito específico e essencial. Obedece a algumas regras que integram os eventos, papéis e artefatos.

Na teoria o Scrum é fundamentado em controle de processo ou empirismo, o qual afirma que o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido, emprega abordagem interativa e incremental para aperfeiçoar

a previsibilidade e o controle de riscos. Três pilares apoiam a implementação de controle de processo empírico:

|               |   |
|---------------|---|
| Transparência | É o aspecto significativo do processo, que deve estar visível aos responsáveis pelos resultados. Requer aspectos definidos por um padrão comum para que os observadores compartilhem um mesmo entendimento do que está sendo visto.   |
| Inspeção      | Os artefatos Scrum devem ser inspecionados por causa de variações, porém essa inspeção não deve ser tão frequente a ponto de atrapalhar a própria execução das tarefas. As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se verificar.  |
| Times Scrum   | É composto por ProductOwner, Time de Desenvolvimento e ScrumMaster. Os times Scrum são auto-organizáveis e multifuncionais. Auto-organizáveis por escolherem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora do time. Multifuncionais por possuírem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe. |

Integram produtos de forma interativa e incremental, assim maximizando as oportunidades de realização.

#### **2.1.5.3 Método de Desenvolvimento de Sistemas Dinâmicos (DSDM)**

É uma metodologia de desenvolvimento de software baseada originalmente no RAD (Desenvolvimento Rápido de Aplicação), é uma metodologia de desenvolvimento iterativo e incremental que enfatiza o envolvimento constante do usuário.

Segundo Pressmann (2011 p. 96), o DSDM é uma abordagem ágil de desenvolvimento de software que “fornece um arcabouço para construir e manter sistemas que satisfazem às restrições de prazo apertado por meio do uso de prototipagem incremental em um ambiente controlado de projeto”.

O DSDM possui as seguintes características:

- ❑ envolvimento ativo do usuário;
- ❑ força de equipe;
- ❑ entrega frequente de produtos;
- ❑ adequação para o propósito do negócio;
- ❑ desenvolvimento iterativo e incremental;
- ❑ todas as mudanças durante o desenvolvimento são reversíveis.

Existe um grupo mundial de empresas que coletivamente assume o papel de “mantenedor” do método. Esse consórcio definiu um método de processos ágeis, chamado ciclo de vida DSDM que define três ciclos iterativos diferentes precedidos por duas atividades de ciclo de vida adicionais:

- ❑ **Estudo de viabilidade:** estabelece os requisitos básicos de negócio e restrições associados à aplicação a ser construída e depois avalia se a aplicação é ou não um candidato viável para o processo DSDM.

- ❑ **Estudo do negócio:** estabelece os requisitos funcionais e de informação que permitem à aplicação agregar valor de negócio.
- ❑ **Iteração de modelos funcionais:** produz um conjunto de protótipos que demonstram funcionalidade para o cliente.
- ❑ **Iteração de projetos e desenvolvimentos:** revisão de protótipos desenvolvidos durante a iteração de modelos funcionais para assegurar-se de que cada um tenha passado por um processo de engenharia para capacitá-lo a oferecer, aos usuários finais, valor de negócio em termos operacionais.
- ❑ **Implementação:** aloca a última versão do incremento de software no ambiente operacional. Deve-se notar que o incremento pode não estar 100% completo ou alterações podem vir a ser solicitadas conforme o incremento seja alocado.

#### **2.1.5.4 Crystal**

A família Crystal é, na verdade, um conjunto de processos ágeis que se mostraram efetivos para diferentes tipos de projeto. A intenção é permitir que equipes ágeis selecionem o membro da família Crystal mais apropriado para o seu projeto e ambiente.

Inclui grande número de métodos diferentes que são selecionados de acordo com as características do projeto a ser desenvolvido. Hoje, apenas dois dos quatro métodos propostos mantêm o desenvolvimento de novas práticas para cobrir diferentes tipos de projetos.

Os membros da família Crystal são identificados por cores que indicam a intensidade do método. Neste caso, quanto mais escura a cor, maior é a complexidade do projeto.

Existem algumas características comuns à família Crystal, tais como o desenvolvimento incremental com ciclos de no máximo quatro meses, ênfase maior na comunicação e cooperação das pessoas, não limitação de quaisquer práticas de desenvolvimento, ferramentas ou produtos de trabalho e incorporação de objetivos para reduzir produtos de trabalho intermediários e desenvolvê-los como projetos evoluídos.

O ciclo de vida é composto pelos seguintes estágios: **staging, edição e revisão, monitoramento, paralelismo e fluxo, inspeções de usuários, workshops refletivos, local matters, produtos de trabalho** (WorkProducts), **padrões (standards)** e **ferramenta (tools)**.

#### **2.1.5.5 Desenvolvimento Dirigido a Funcionalidades (FDD)**

Trata-se de um método ágil de abordagem adaptativa. O FDD foi originalmente concebido por Peter Coad e seus colegas como um modelo prático de processo para a engenharia de software orientado a objetos.

No contexto do FDD, uma característica “é uma função valorizada pelo cliente que pode ser implementada em duas semanas ou menos” (PRESSMANN, 2011, p. 98). A ênfase na definição de características fornece os seguintes benefícios:

- └ Como as características são pequenos blocos de funcionalidade passíveis de entrega, os usuários podem descrevê-las mais facilmente, entender como elas se relacionam umas com as outras mais prontamente e revisá-las melhor quanto a ambiguidades, erros ou omissões.
- └ As características podem ser organizadas em um argumento hierárquico relacionado ao negócio.
- └ Como uma característica é um incremento de software passível de entrega do FDD, a equipe desenvolve características operacionais a cada duas semanas.
- └ Como as características são pequenas, suas representações de projeto e de código são mais fáceis de inspecionar efetivamente.
- └ Planejamento de projeto, cronogramação e monitoração são guiados pela hierarquia de características em vez de por um conjunto de engenharia de software arbitrariamente adotado.

## 2.2 Modelo evolucionário

Como o próprio nome já sugere, os modelos evolucionários são projetados para acomodar um produto que evolui conforme o tempo e a necessidade.

Uma vez que o software evolui ao longo do tempo e conforme o desenvolvimento do software avança, também temos mudanças nas necessidades de negócio e de produtos que mudam frequentemente. Isso torna inadequado seguirmos um planejamento em linha reta de um produto. Os modelos de processo evolucionário tornaram-se realidade para que possamos desenvolver um produto que evolua ao longo do tempo.

### 2.2.1 Características

Modelos evolucionários são caracterizados por serem iterativos e apresentarem propriedades que possibilitem desenvolvermos versões cada vez mais completas do software.

O software evolui ao longo do tempo e conforme o desenvolvimento avança, também temos mudanças nas necessidades de negócio e de produtos que mudam frequentemente. Isso torna inadequado seguirmos um planejamento em linha reta de um produto. Os modelos de processo evolucionário tornaram-se realidade para que possamos desenvolver um produto que evolua ao longo do tempo.

Modelos evolucionários são caracterizados por serem iterativos e apresentarem características que possibilitem desenvolvermos versões cada vez mais completas do software.

Em muitos casos, o tempo de colocação de um produto no mercado é o requisito mais importante a ser gerenciado. Se o momento oportuno de entrada no mercado for perdido, o projeto de software pode ficar sem sentido. Os modelos de processo evolucionário foram concebidos para tratar dessas questões e, mesmo assim, como uma classe genérica de modelos de processo, apresentam seus pontos fracos:

- └ O primeiro é que a prototipação (e outros processos evolucionários mais sofisticados) traz um problema para o planejamento do projeto pelo número incerto

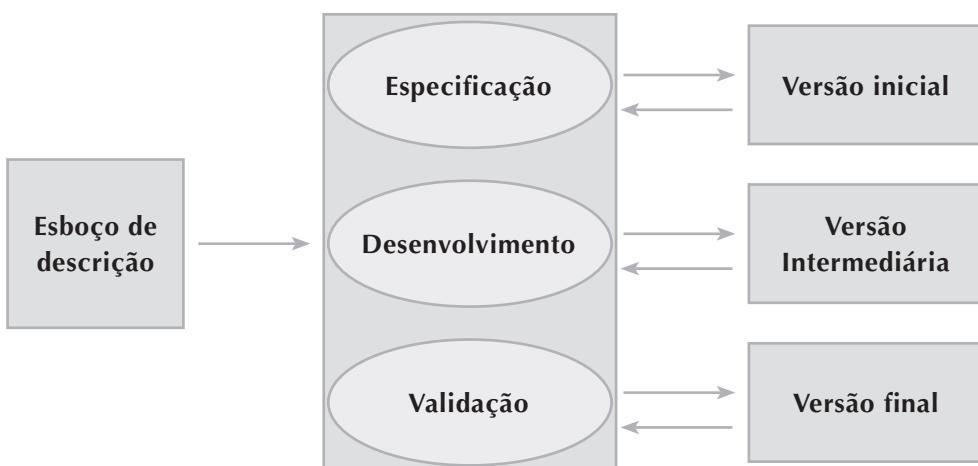
de ciclos necessários para construir o produto. A maior parte dessas técnicas de gerenciamento e de estimativas do projeto baseia-se em layouts lineares das atividades, o que faz com que não se adequem completamente.

- └ Segundo, os processos de software evolucionários não estabelecem a velocidade máxima da evolução. Se as evoluções ocorrerem numa velocidade excessivamente rápida, sem um período de acomodação, é certo que o processo cairá no caos. Por outro lado, se a velocidade for muito lenta, então a produtividade poderia ser afetada.
- └ Terceiro, os processos de software devem manter seu foco mais na flexibilidade e na extensibilidade do que na alta qualidade. Entretanto, deve-se priorizar mais a velocidade de desenvolvimento do que a do desenvolvimento com zero defeito. Prolongar o desenvolvimento em busca da alta qualidade pode resultar em entrega tardia do produto, quando o nicho de oportunidade já desapareceu. Essa mudança de paradigma é imposta pela concorrência em situações em que se está à beira do caos.

O objetivo dos modelos evolucionários é desenvolver software de alta qualidade de modo iterativo ou incremental. Entretanto, é possível usar um processo evolucionário para enfatizar a flexibilidade, a extensibilidade e a velocidade do desenvolvimento. O desafio para as equipes de software e seus gerentes será estabelecer um equilíbrio apropriado entre esses parâmetros críticos de projeto e produto e a satisfação do cliente (o último a aprovar a qualidade do software).

Tem como base a ideia de desenvolver uma implementação inicial, expor o resultado ao usuário e fazer seu aprimoramento por meio de muitas versões, até que tenha sido desenvolvido.

**Figura 2.5 Atividades concorrentes**



Fonte: Do autor (2014).

O modelo evolucionário apresenta vantagens, como o fato de um sistema funcional ser apresentado antecipadamente ou de o cliente se envolver na avaliação do protótipo.

Contudo as desvantagens desse modelo também existem: uma delas é que o processo de desenvolvimento não é visível, pois, como o sistema é desenvolvido rapidamente, não há tempo de documentar as versões. Além disso, os sistemas tendem a ser mal estruturados e sua mudança constante pode corromper a estrutura do software.

O modelo evolucionário é aplicável a sistemas interativos de pequeno ou médio porte. Também é aplicável para partes de sistemas grandes, como a interface com o usuário. Além disso, também é usado em sistemas de vida curta.

## 2.2.2 Modelos de processo evolucionários

Dentro os modelos de processos evolucionários, destacamos:

### 2.2.2.1 RUP — RationalUnifiedProcess

O RUP (RationalUnifiedProcess) é um exemplo de modelo de processo moderno, derivado de trabalhos sobre a UML (ferramenta de Diagramas de Caso de Uso) e o Unified Software Development Process associado. Ele reúne elementos de todos os modelos de processo genérico, ilustra boas práticas na especificação e no projeto e apoia a prototipação e a entrega incremental.

O RUP reconhece que os modelos de processo convencionais apresentam uma visão única do processo. Em contrapartida, ele é normalmente descrito em três perspectivas:

- └ Uma perspectiva dinâmica, que mostra as fases do modelo ao longo do tempo.
- └ Uma perspectiva estática, que mostra as atividades realizadas no processo.
- └ Uma perspectiva prática, que sugere boas práticas a serem usadas durante o processo.

A perspectiva prática sobre o RUP descreve as boas práticas da engenharia de software que são recomendadas para uso no desenvolvimento de sistemas. Seis boas práticas fundamentais são recomendadas:

1. **Desenvolver software iterativamente.**
2. **Gerenciar os requisitos.**
3. **Usar arquiteturas baseadas em componentes.**
4. **Modelar o software visualmente.**
5. **Verificar a qualidade do software.**
6. **Controlar as mudanças do software.**

A maioria das descrições do RUP tenta combinar as perspectivas estática e dinâmica em um único diagrama. Ele é um modelo constituído de fases que identifica quatro fases distintas no processo de software: **concepção, elaboração, construção e transição**.

No entanto, ao contrário do modelo em cascata, no qual as fases são equalizadas com as atividades do processo, as fases do RUP são estreitamente relacionadas ao negócio, e não a assuntos técnicos. Cada fase pode comportar várias iterações, e cada iteração está organizada em workflows, que descrevem o que deve ser feito em termos de atividades reponsáveis e artefatos.

As iterações também são finalizadas com *milestones*, que devem controlar se foram cumpridos os objetivos específicos da iteração, como a realização de um grupo de casos de uso, por exemplo. O RUP possui nove workflows, seis de engenharia de software e três de suporte. Os workflows de engenharia são **modelagem de negócio, requisitos, análise e projeto, implementação, teste, distribuição**. Os workflows de suporte compreendem atividades necessárias para a execução dos workflows de engenharia. São eles: **gerência de projeto, gerência de configuração e mudanças e configuração do ambiente**.

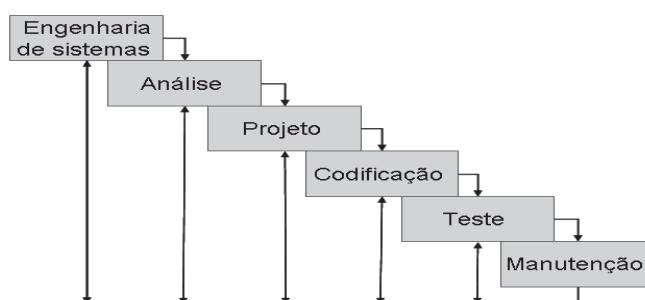
### 2.2.2.2 Cascata

O modelo cascata tornou-se conhecido na década de 1970 e é referenciado na maioria dos livros de engenharia de software ou manuais de padrões de software. Nele, as atividades do processo de desenvolvimento são estruturadas numa cascata onde a saída de uma é a entrada para a próxima.

As suas principais atividades (Figura 2.6) são:

- └ engenharia de sistemas (estudo de viabilidade);
- └ análise (especificação de requisitos);
- └ projeto (design e especificação);
- └ codificação e testes de componentes;
- └ teste do sistema e integração;
- └ entrega e instalação;
- └ manutenção.

**Figura 2.6 Ciclo de vida do modelo em cascata**



Fonte: Do autor (2014).

Esse modelo, quando proposto, introduziu importantes qualidades ao desenvolvimento. A primeira chama a atenção de que o processo de desenvolvimento deve ser conduzido de forma disciplinada, com atividades claramente definidas, determinada a partir de um planejamento e sujeitas a gerenciamento durante a realização.

Outra qualidade define de maneira clara quais são essas atividades e quais os requisitos para desempenhá-las. Por fim, o modelo introduz a separação das atividades da definição e design da atividade de programação que era o centro das atenções no desenvolvimento de software.

O modelo Cascata também é criticado por ser linear, rígido e monolítico. Inspirados em modelos de outras atividades de engenharia, argumenta que cada atividade apenas deve ser iniciada quando a outra estiver terminada e verificada. Ele é considerado monolítico por não introduzir a participação de clientes e usuário durante as atividades do desenvolvimento, mas apenas após o software ter sido implementado e entregue. Não existe como o cliente verificar antecipadamente qual o produto final para detectar eventuais problemas.

### **2.2.2.3 Incremental**

É uma adaptação da forma de pensar o desenvolvimento de software como um “desenvolvimento linear”, pois se trata de um arranjo de vários pequenos ciclos em cascata.

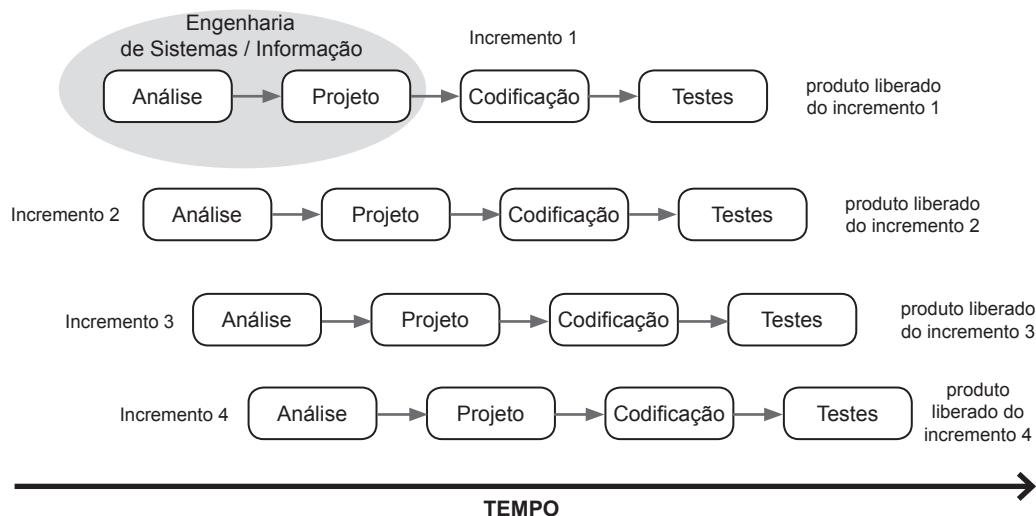
A diferença é que cada versão do produto de software tem uma nova funcionalidade (incremento), definida no início desse ciclo. Depois de desenvolvida cada versão, será entregue ao usuário para testes. Cada versão implementa uma nova funcionalidade (incremento).

O modelo que é a versão evolucionária do Modelo Sequencial Linear. O software desenvolvido pode sempre crescer e agregar novas funcionalidades, assim será desenvolvido por um incremento e esse incremento segue todas as etapas descritas no modelo linear.

O modelo incremental é muito utilizado em projetos longos e que tendem a crescer com o passar do tempo e o prazo de entrega é curto. Como, ao final de cada incremento, é gerada uma versão do produto final, é possível mostrar ao cliente como está o andamento do projeto e atingir as metas definidas no começo.

Em cada incremento é realizado todo o ciclo do desenvolvimento de software, do planejamento aos testes do sistema já em funcionamento. Cada etapa produz um sistema totalmente funcional, apesar de ainda não cobrir todos os requisitos. Alguns projetos de software definem requisitos iniciais de software razoavelmente bem definidos. Pode ser necessário o rápido fornecimento de um determinado conjunto funcional aos usuários, para que após esse fornecimento possamos melhorar e expandir suas funcionalidades em versões de software posteriores. Nesses casos, podemos optar por um modelo de processo que desenvolve software de uma forma incremental.

Podemos notar pela Figura 2.7 que o modelo de processo incremental aplica sequências lineares (como no modelo cascata) de forma escalonada, à medida que o tempo for avançando. Cada uma das sequências lineares gera um incremento do software. Esses incrementos são entregáveis e prontos para o cliente.

**Figura 2.7 Ciclo de vida do modelo incremental**

Fonte: Do autor (2014).

O modelo de processo incremental entrega um produto operacional a cada incremento, ou seja, um produto sem erros e pronto para o usuário utilizar. Mesmo que os primeiros incrementos sejam partes do produto, essas partes são operacionais e funcionam sem as outras. Portanto, os incrementos possuem totais condições de atender ao usuário.

#### **2.2.2.4 Espiral**

O objetivo do modelo espiral é prover um modelo que pode acomodar diversos processos específicos. Prevê prototipação, desenvolvimento evolutivo e cíclico, e as principais atividades do modelo cascata.

Sua principal inovação é guiar o processo de desenvolvimento gerado a partir desse modelo com base em **análise de riscos** e um **planejamento** que é realizado durante toda a evolução do desenvolvimento.

Riscos são circunstâncias adversas que podem surgir durante o desenvolvimento de software impedindo o processo ou diminuindo a qualidade do produto. São exemplos de riscos: pessoas que abandonam a equipe de desenvolvimento, ferramentas que não podem ser utilizadas, falha em equipamentos usados no desenvolvimento ou que serão utilizados no produto final etc. A identificação e o gerenciamento de riscos é hoje uma atividade importantíssima no desenvolvimento de software por causa da imaturidade da área e da falta de conhecimento, técnicas e ferramentas adequadas.

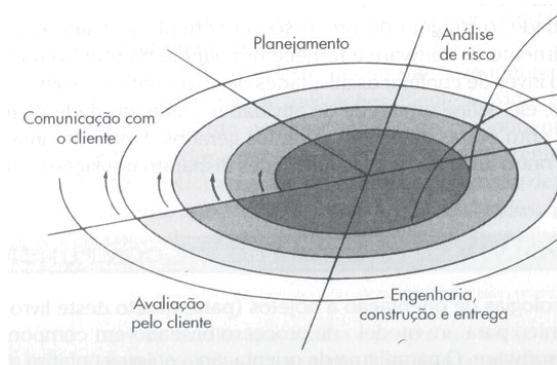
O modelo espiral descreve um fluxo de atividades cíclico e evolutivo constituído de quatro estágios, em que no primeiro devem ser determinados objetivos, soluções alternativas e restrições. No segundo devem ser analisados os riscos das decisões do

estágio anterior. Já o terceiro consiste nas atividades da fase de desenvolvimento, incluindo design, especificação, codificação e verificação. Por fim o quarto estágio compreende a revisão das etapas anteriores ao planejamento da próxima fase.

A Figura 2.8 mostra o modelo de ciclo de vida, o qual define quatro importantes atividades:

- **Planejamento:** determinação dos objetivos do sistema que será desenvolvido, restrições impostas à aplicação, tais como: desempenho, funcionalidade, capacidade de acomodar mudanças, meios alternativos de implementação.
- **Análise de risco:** análise das alternativas e identificação/resolução dos riscos. Uma vez avaliados os riscos, podem-se construir protótipos para verificar se são realmente robustos para servir de base para a evolução futura do sistema.
- **Engenharia:** desenvolvimento do produto do “próximo nível”.
- **Avaliação do cliente:** avaliação dos resultados de engenharia podem-se efetuar a verificação e validação.

**Figura 2.8 Ciclo de vida do modelo espiral**



Fonte: Do autor (2014).

### 2.2.2.5 Prototipação

Prototipação é uma abordagem baseada em uma visão evolutiva do desenvolvimento de software, afetando o processo como um todo. Envolve a produção de versões iniciais — protótipos — de um sistema futuro com o qual é possível realizar verificações e experimentos, com o intuito de avaliar algumas de suas características antes que o sistema venha realmente a ser construído de forma definitiva.

Apesar de a prototipagem poder ser usada como um modelo de processo independente, ela é mais comumente usada como uma técnica que pode ser implementada dentro do contexto de qualquer processo (modelo cascata, espiral, por exemplo). Independentemente da maneira como é aplicado, o paradigma de prototipagem auxilia

o engenheiro de software e o cliente a entenderem melhor o que deve ser construído quando os requisitos estão confusos.

Utilizada como uma maneira de se obter informações e apresentar essas informações aos usuários. O protótipo vai sendo melhorado até atingir o objetivo final, ou seja, até que o mesmo atinja o sistema. A prototipagem pode ser:

- └ Evolucionária: abordagem para o desenvolvimento do sistema em que um protótipo inicial é produzido e refinado em vários estágios até atingir o sistema final. O objetivo da prototipação evolucionária é fornecer aos usuários finais um sistema funcionando.
- └ Descartável: um protótipo que é usualmente uma implementação prática do sistema é produzido para ajudar a levantar os problemas com os requisitos, e depois descartado. O sistema é então desenvolvido usando algum outro processo de desenvolvimento. O objetivo da prototipação descartável é validar ou derivar os requisitos do sistema.



### *Questões para reflexão*

Reflita a respeito dos modelos de processos de desenvolvimento de software, verificando qual modelo seria o melhor para uma empresa de desenvolvimento de software nos dias de hoje.



### *Atividades de aprendizagem*

1. De acordo com o que estudamos sobre modelos de processo, podemos então dizer que um exemplo de modelo de processo ágil é:
  - a) O modelo RAD.
  - b) A Programação Extrema.
  - c) O modelo incremental.
  - d) O Processo Unificado.
  - e) O modelo espiral.
2. Levando em consideração o ciclo de vida de um software, o qual descreve sua existência desde sua concepção até fim de sua utilização e considerando os métodos de produção e dos processos de desenvolvimento de software, analise os itens que se seguem.

- I. A construção de um produto de software pressupõe o compromisso com um conjunto de requisitos antes do início do desenvolvimento do produto, seguido de um processo o qual é finalizado com a implantação do referido produto.
- II. Em um modelo de processo de desenvolvimento em cascata, os testes de software são realizados em todos estágios do desenvolvimento e sua finalização se dá na fase de implementação.
- III. Em uma empresa que tenha adotado um processo de desenvolvimento de software em cascata, falhas no levantamento de requisitos têm maior possibilidade de gerar grandes prejuízos do que naquelas que tenham adotado desenvolvimento evolucionário.

Considerando os itens anteriores, é correto o que consta em:

- a) I, II e III.
- b) I, apenas.
- c) II, apenas.
- d) III, apenas.
- e) I e II apenas

**Fique ligado!**



Nesta unidade, foram abordados os seguintes aspectos relativos aos requisitos de software, à engenharia de requisitos e ao gerenciamento de projetos relacionados aos objetivos de aprendizagem:

- └ Software é o elemento-chave na evolução de produtos e sistemas baseados em computador e uma das mais importantes tecnologias no cenário mundial e, nos últimos 50 anos, evoluiu de uma ferramenta especializada em análise de informações e resolução de problemas para uma indústria propriamente dita. Porém, ainda temos problemas para desenvolver software de boa qualidade dentro do prazo e do orçamento estabelecidos.
- └ Softwares (programas, dados e informações descritivas) contemplam uma ampla gama de áreas de aplicação e tecnologia. O software legado continua a representar desafios especiais àqueles que precisam fazer sua manutenção.

- └ A engenharia de software engloba processos, métodos e ferramentas que possibilitam a construção de sistemas complexos baseados em computador, dentro do prazo e com qualidade.
- └ O processo de software incorpora cinco atividades estruturais: comunicação, planejamento, modelagem, construção e emprego; e elas se aplicam a todos os projetos de software. A prática da engenharia de software é uma atividade de resolução de problemas que segue um conjunto de princípios básicos.
- └ Inúmeros mitos em relação ao software continuam a levar gerentes e profissionais para o mau caminho, mesmo com o aumento do conhecimento coletivo de software e das tecnologias necessárias para construí-los. À medida que for aprendendo mais sobre a engenharia de software, você começará a compreender por que esses mitos devem ser derrubados toda vez que deparar com eles.
- └ Um modelo de processo genérico para engenharia de software consiste num conjunto de atividades metodológicas e de apoio (atividades guarda-chuvas), ações e tarefas a realizar. Cada modelo de processo, dentre os vários existentes, pode ser descrito por um fluxo de processo diferente, isto é, descrição de como as atividades metodológicas, ações e tarefas são organizadas sequencial e cronologicamente.
- └ Padrões de processo são utilizados para resolver problemas comuns encontrados como parte do processo de software.
- └ Os modelos de processo sequenciais, tal como o de cascata, são os paradigmas da engenharia de software mais antigos. Eles sugerem um fluxo de processos linear que, frequentemente, é inadequado para considerar as características dos sistemas modernos (por exemplo, contínuas alterações, sistemas em evolução, prazos apertados). Entretanto, têm, realmente, aplicabilidade em situações em que os requisitos são bem definidos e estáveis.
- └ Modelos de processo incremental são iterativos por natureza e produzem rapidamente versões operacionais do software. Modelos de processos evolucionários reconhecem a natureza iterativa e incremental da maioria dos projetos de engenharia de software e são projetados para adequar mudanças.
- └ Esses modelos, como prototipação e o modelo espiral, produzem rapidamente artefatos de software incrementais (ou versões operacionais do software). Podem ser adotados para ser aplicados por todas as atividades de engenharia de software, desde o desenvolvimento de conceitos até a manutenção do sistema a longo prazo.

- └ O Processo Unificado é um processo de software “dirigido a casos práticos, centrado na arquitetura, iterativo e incremental”, desenvolvido como uma metodologia para os métodos e as ferramentas da UML.
- └ Uma filosofia ágil na engenharia de software enfatiza a importância das equipes que se auto-organizam para que tenham o controle sobre o trabalho realizado, sobre a comunicação e colaboração entre os componentes da equipe e entre os desenvolvedores e seus clientes, o reconhecimento de que as mudanças representam oportunidades e, por fim, a ênfase na entrega rápida do software para satisfazer o cliente.
- └ O eXtremeProgramming (XP) é o processo ágil mais amplamente usado, e que sugere um número de técnicas inovadoras que possibilitam a uma equipe ágil criar versões de software rapidamente, priorizando os envolvidos.

### *Para concluir o estudo da unidade*



Concluindo a unidade podemos, então, notar a importância do software para a sociedade, bem como a sua evolução. Também notamos que nas últimas décadas o desenvolvimento de produtos de software evoluiu muito e que vários métodos de desenvolvimento foram criados, o que auxiliou muito para a adaptação das mudanças ocorridas nas organizações que produzem e consomem um software.

Podemos afirmar quer as Tecnologias de Informação são relativamente novas, têm menos de 70 anos, e sua evolução é espantosa se compararmos com outras tecnologias. Durante o seu desenvolvimento muitos mitos foram quebrados, muitas atividades sofreram evolução, tais como os modelos de processos de software que até o início da década de 1980 eram ainda desconhecidos e atualmente já temos um aparato muito grande e robusto de procedimentos que auxiliam o desenvolvedor e/ou gerente de projeto em seu trabalho cotidiano.

Há ainda espaço para muitas mudanças que estão por vir, visto que o desenvolvimento web exige, agora, novas formas de atuação e de conhecimento das tecnologias, pois estão centradas na conectividade e no compartilhamento de tudo — dados, som, vídeo — em tempo real. Vendo isso, devemos encarar como um novo e grande desafio para a criação de novos modelos de processos que nos auxiliem nesta nova e desafiadora jornada.



## Atividades de aprendizagem da unidade

1. Considerando o modelo de processo de desenvolvimento de software RUP, e suas fases e artefatos, analise cada artefato com a sua respectiva fase em que deverá ser realizado:

| FASES      | ARTEFATOS                           |
|------------|-------------------------------------|
| Concepção  | I — Avaliação de riscos iniciais    |
| Elaboração | II — Relatório de execução (testes) |
| Construção | III — Modelo de projeto finalizado  |
| Transição  | IV — Modelo de negócio              |
|            | V — Projeto de arquitetura          |

De acordo com o quadro anterior, é correto afirmar que:

- a) I e IV correspondem à fase de concepção e V à fase de transição.
  - b) II corresponde à fase de transição, III à fase de construção e I, IV e V à fase de concepção.
  - c) I e IV correspondem à fase de concepção, II à fase de transição e III e V à de elaboração.
  - d) II corresponde à fase de transição, V à fase de elaboração, I e IV à de concepção e III à de construção.
  - e) I e IV correspondem à fase de elaboração, II à fase de construção, III à fase de transição e V à fase de concepção.
2. Na engenharia de software, qual o modelo de processo em que o cliente deve declarar todos os requisitos explicitamente na primeira parte do projeto, o qual gera insegurança, uma vez que eles às vezes não têm um entendimento completo do mesmo. Para minimizar tal dificuldade, é posta em prática uma técnica utilizada para minimizar esse problema de definição de requisitos conhecida como:
- a) Modelo de processo em cascata.
  - b) Modelo de processo ágil.
  - c) Modelo de processo unificado.
  - d) Modelo Scrum de projeto de software incremental.
  - e) Prototipação.
3. Um software que resida apenas na memória de leitura e que é utilizado para controlar produtos e sistemas para os mercados industriais e de consumo é chamado de:

- a) Software básico.
  - b) Software embutido.
  - c) Software de tempo real.
  - d) Software comercial.
  - e) Software de inteligência artificial.
4. O modelo de processo de desenvolvimento de software em cascata, também conhecido como modelo clássico do processo de desenvolvimento de software, continua sendo usado para o desenvolvimento de sistemas de informação. Analise as afirmativas que correspondam aos estágios do modelo em cascata.
- I. Análise de Sistemas, Análise de Negócio e Análise Ética.
  - II. Análise de Sistemas, Programação e Testes.
  - III. Projeto de Sistemas, Testes e Manutenção.
  - IV. Projeto de Sistemas, Integração e Terceirização.
- Está(ão) correta(s) as afirmativas:
- a) I e II.
  - b) I, II e III.
  - c) III apenas.
  - d) II e III.
  - e) III e IV.
5. De acordo com a evolução histórica do software, podemos notar a ocorrência da crise e a ocorrência dos mitos de software. Sendo assim, analise as sentenças a seguir e assinale a alternativa correta.
- a) No que diz respeito à crise do software é correto afirmar que se refere a problemas encontrados no desenvolvimento, tais como estimativas de prazo e de custo frequentemente imprecisas; a produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços; e a qualidade de software às vezes é menos que adequada.
  - b) Com relação aos mitos de software relacionados a cliente, é certo dizer: se nós estamos atrasados nos prazos, podemos adicionar mais programadores e tirar o atraso, porém o que acontece na realidade é que o desenvolvimento de software não é um processo mecânico igual à manufatura. Acrescentar pessoas a um projeto torna-o ainda mais atrasado. Pessoas podem ser acrescentadas, mas somente de uma forma planejada.

- c) Já em relação aos mitos administrativos, notamos que, enquanto não tiver o programa “funcionando”, não teremos realmente nenhuma maneira de avaliar sua qualidade, porém na realidade um programa funcionando é somente uma parte de uma configuração de software que inclui todos os itens de informação produzidos durante a construção e manutenção.
- d) Nos mitos profissionais, vimos que os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível.
- e) Ainda com relação ao mitos profissionais, podemos dizer que uma declaração geral dos objetivos é suficiente para se começar a escrever programas — podemos preencher os detalhes mais tarde.

---

## Referências

PRESSMANN, Roger S. **Engenharia de software:** uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, Ian. **Engenharia de software.** 8. ed. São Paulo: Pearson, 2007.



# Engenharia de requisitos e gerenciamento de projeto de software

*Luis Cláudio Perini*

## Objetivos de aprendizagem:

- └ Compreender os conceitos dos requisitos de usuário e de sistema e por que são escritos de forma diferente.
- └ Saber a diferença entre requisitos funcionais e não funcionais.
- └ Entender como os requisitos podem ser organizados em um documento de requisitos.
- └ Compreender as principais atividades da engenharia de requisitos e seus relacionamentos.
- └ Entender as técnicas de elicitação e análise de requisitos.
- └ Compreender a importância da validação e das revisões de requisitos.
- └ Compreender por que o gerenciamento de requisitos é necessário e como o mesmo apoia outras atividades da engenharia de requisitos.
- └ Compreender as principais tarefas dos gerentes de projeto.
- └ Compreender como a natureza do software torna o gerenciamento de projeto mais difícil que o gerenciamento de projetos de outras engenharias.
- └ Descobrir a necessidade de planejamento de projetos.
- └ Saber como as representações gráficas são usadas por gerentes de projeto para representar cronogramas de projeto.
- └ Ter a noção de gerenciamento de riscos e alguns riscos que podem surgir em processos de software.

**» Seção 1: Requisitos de software**

Nessa seção, abordaremos uma discussão sobre requisitos de software bem como o processo de documentação e especificação de requisitos.

**» Seção 2: Engenharia de requisitos**

Nessa seção, abordaremos as atividades envolvidas no processo de engenharia de requisitos.

**» Seção 3: Gerenciamento de projetos de software**

Nessa seção será abordada uma visão geral do gerenciamento de projetos de software.

---

## Introdução ao estudo

Talvez o maior problema que enfrentamos no desenvolvimento de sistemas de software grandes e complexos seja o da engenharia de requisitos. Ela está relacionada com a definição de que o sistema deve fazer suas propriedades emergentes desejáveis e essenciais e as restrições quanto à operação do sistema e quanto aos processos de desenvolvimento de software. Você pode, portanto, pensar na engenharia de requisitos como o processo de comunicação entre os clientes e os usuários de software e os desenvolvedores de software.

A engenharia de software não é simplesmente um processo técnico. Os requisitos do sistema são influenciados pelas preferências, recusas e preconceitos dos usuários, além das questões políticas e organizacionais. Esses fatores são características humanas fundamentais, e novas tecnologias, como casos de uso, cenários e métodos formais, não nos ajudam muito na resolução desses difíceis problemas.

---

### Seção 1 Requisitos de software

Os requisitos de um sistema são as descrições de tudo o que o sistema deve realizar, ou seja, os serviços que oferece bem como as restrições em seu funcionamento. Os requisitos são o reflexo das necessidades dos clientes de como resolver um determinado problema, de como controlar um dispositivo, controlar pedidos ou encontrar determinadas informações.

Nesta seção vamos então aprender um pouco mais sobre o assunto.



#### Para saber mais

Iniciando nosso estudo sobre requisitos de software, acesse e leia: <<http://www.devmedia.com.br/introducao-a-requisitos-de-software/29580>>.

### 1.1 Requisitos de software

Os requisitos de um sistema são descrições dos serviços fornecidos pelo sistema bem como as suas restrições operacionais. Requisitos esses que refletem as necessidades dos clientes de um sistema que os ajude a resolver algum problema, tal como controlar um dispositivo, enviar um pedido ou encontrar informações. Esse processo de descobrir, analisar, documentar e verificar esses serviços e restrições é denominado de *engenharia de requisitos* (RE — Requirements Engineering).

O termo *requisito* não é usado pelos desenvolvedores de software de forma consistente, visto que em algumas situações um requisito é apenas uma declaração abstrata de alto nível de um serviço que o sistema deve fornecer ou uma restrição do sistema. Já em outro extremo, é uma definição formal e detalhada de uma função do sistema.

Alguns dos problemas que surgem durante o processo de engenharia de requisitos são resultantes da falta de uma separação entre esses diferentes níveis de descrição. Sommerville (2007) faz distinção entre eles usando o termo **requisitos de usuário** para requisitos abstratos de alto nível e **requisitos de sistema** para a descrição detalhada do que o sistema deve fazer. Sommerville (2007, p. 80) diz que os requisitos de usuário e de sistema podem ser definidos da seguinte maneira:

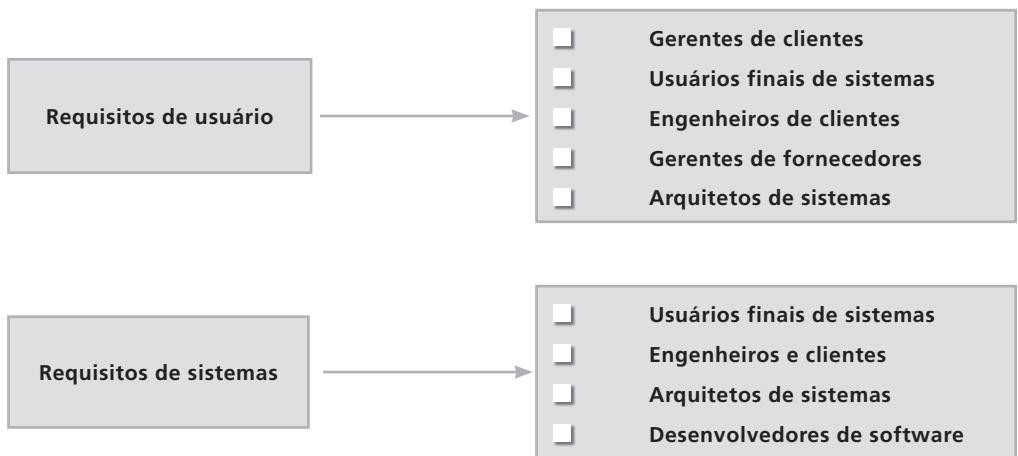
1. *Requisitos de usuário* são declarações, em uma linguagem natural com diagramas, de quais serviços são esperados do sistema e as restrições sob as quais ele deve operar.
2. *Requisitos de sistema* definem, detalhadamente, as funções, os serviços e as restrições operacionais do sistema. O documento de requisitos de sistema, também chamado de especificação funcional, deve ser preciso. Ele deve definir exatamente o que será implementado. Pode ser parte do contrato entre o comprador do sistema e os desenvolvedores de software.

A Figura 3.1 mostra o tipo de leitores para os requisitos de usuários e de sistemas; os leitores de requisitos geralmente não se preocupam com o modo que o sistema será desenvolvido e também podem ser gerentes que não se interessam pelo detalhamento dos recursos do sistema.

### 1.1.1 Requisitos funcionais e não funcionais

De acordo com Sommerville (2007) os requisitos de sistema de software são, frequentemente, classificados em requisitos funcionais, requisitos não funcionais ou requisitos de domínio:

1. *Requisitos funcionais*. São as declarações de serviços que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também estabelecer explicitamente o que o sistema não deve fazer.
2. *Requisitos não funcionais*. São restrições sobre os serviços ou as funções oferecidas pelo sistema. Eles incluem restrições de timing, restrições sobre o processo de desenvolvimento e padrões. Os requisitos não funcionais aplicam-se, frequentemente, ao sistema como um todo. Em geral, eles não se aplicam às características ou serviços individuais de sistema.
3. *Requisitos de domínio*. São requisitos provenientes do domínio da aplicação do sistema e que refletem as características e as restrições desse domínio. Podem ser requisitos funcionais ou não funcionais.

**Figura 3.1 Leitores de tipos de especificação**

Fonte: Adaptada de Sommerville (2007, p. 81).

### 1.1.1.1 *Requisitos funcionais*

Os requisitos funcionais de um sistema descrevem o que ele deve fazer. Dependem do tipo do software que está sendo desenvolvido, dos usuários aos quais o software se destina e também da abordagem considerada pela organização ao redigir os requisitos. Os requisitos funcionais descrevem a função do sistema detalhadamente, suas entradas e saídas, exceções etc.

Os requisitos funcionais de um sistema de software podem ser expressos de várias formas. Eles definem recursos específicos a serem fornecidos pelo sistema. Geralmente originaram do documento de requisitos de usuário e mostram que os requisitos funcionais podem ser escritos em níveis diferentes de detalhes.

A imprecisão na especificação de requisitos é um dos motivos de muitos problemas de engenharia de software. É comum que um desenvolvedor de sistema interprete um requisito ambíguo de modo a simplificar sua implementação, pois frequentemente isso não é o que o cliente deseja. Novos requisitos necessitam ser definidos e mudanças devem ser projetadas no sistema. Com isso há atrasos na entrega do sistema com um aumento dos custos.

Em princípio, a especificação de requisitos funcionais de um sistema deve ser completa e consistente.

- └ *Completa*za significa que todos os serviços exigidos pelo usuário devem ser definidos.
- └ *Consistênci*a significa que os requisitos não devem ter definições contraditórias.

Na prática, em sistemas grandes e complexos, é praticamente impossível atingir a consistência e a completeza de requisitos. Uma razão para isso é que é fácil co-

meter erros e omissões quando se redigem especificações para sistemas grandes e complexos.

### ***1.1.1.2 Requisitos não funcionais***

Os requisitos não funcionais, como o próprio nome sugere, são aqueles não diretamente relacionados com as funções específicas fornecidas pelo sistema. Podem estar relacionados com as propriedades do sistema tais como confiabilidade, tempo de resposta e espaço de armazenamento. Como resposta, os requisitos não funcionais podem definir restrições, como a capacidade dos dispositivos de E/S (entrada/saída) e as representações de dados usadas nas interfaces do sistema.

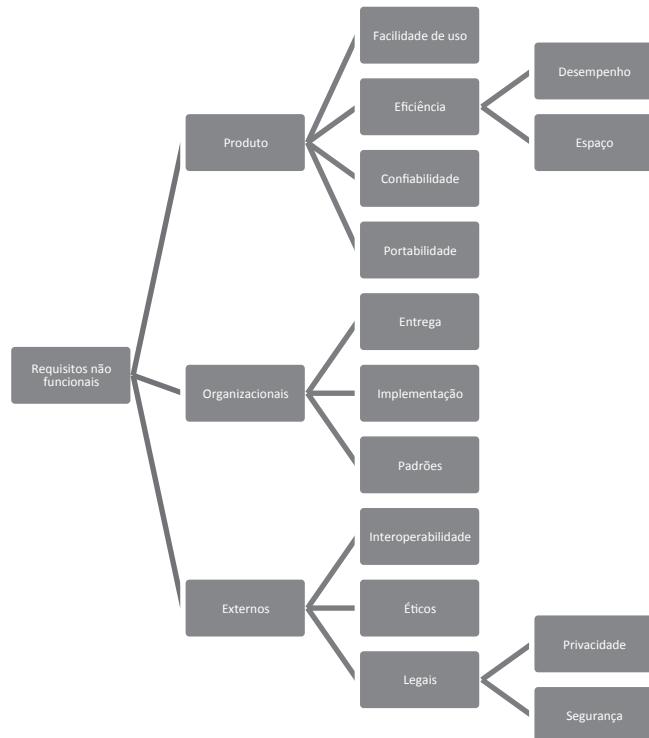
Os requisitos não funcionais estão raramente associados às características individuais do sistema; ao contrário, eles especificam ou restringem as propriedades emergentes de sistema. Assim sendo, podem especificar desempenho, proteção, disponibilidade e outras.

Isso significa que eles são mais importantes do que os requisitos funcionais individuais, uma vez que os usuários do sistema em geral encontram meios de contornar uma função do sistema que não atenda às suas necessidades. Entretanto uma falha no atendimento de um requisito não funcional pode significar que todo o sistema é inútil. Por exemplo, se um sistema de controle de tempo real falhar em atender aos requisitos de desempenho, as funções de controle não operarão corretamente.

Os requisitos não funcionais não estão relacionados apenas com o sistema de software a ser desenvolvido; alguns deles podem restringir o processo que deve ser usado para desenvolver o sistema. Eles surgem por causa das necessidades do usuário, das restrições de orçamento, das políticas organizacionais, da necessidade de interoperabilidade com outros sistemas de software ou hardware ou de fatores externos, como regulamentos de segurança ou legislação a respeito da privacidade. A Figura 3.2 apresenta uma classificação de requisitos não funcionais

Você pode observar, no diagrama, que os requisitos não funcionais podem vir de características necessárias do software (requisitos de produto), da organização que desenvolve o software (requisitos organizacionais) ou de fontes externas.

Figura 3.2 Tipos de requisitos não funcionais



Fonte: Adaptada de Sommerville (2007, p. 82).

Sommerville (2007, p. 83) define os seguintes tipos de requisitos não funcionais:

■ *Requisitos de produto.*

Especificam o comportamento do produto, tais como os requisitos de desempenho quanto à rapidez com que o sistema deve operar e quanto de memória ele requer; os requisitos de confiabilidade que definem a taxa aceitável de falhas; os requisitos de portabilidade e requisitos de usabilidade.

■ *Requisitos organizacionais.*

São oriundos de políticas e procedimentos da organização do cliente e do desenvolvedor, incluindo padrões de processo que devem ser usados, requisitos de implementação, como a linguagem de programação, o método de projeto usado, e requisitos de entrega que especificam quando o produto e a sua documentação devem ser entregues.

■ *Requisitos externos.*

Abrangem todos os requisitos derivados de fatores externos ao sistema e seu processo de desenvolvimento, incluindo: requisitos de interoperabilidade, o qual define como o sistema interage com os sistemas em outras organizações; requisitos legais que devem ser seguidos para assegurar que o sistema funciona

dentro da lei; e requisitos éticos (aqueles incluídos no sistema para assegurar que ele será aceito por seus usuários e pelo público em geral).

Um requisito de produto restringe a liberdade dos projetistas na implementação da interface com o usuário de sistema.

Um requisito organizacional especifica que o sistema deve ser desenvolvido de acordo com um processo padrão da empresa. O requisito externo é derivado da necessidade de o sistema estar em conformidade com a legislação de privacidade.

Um problema comum com os requisitos não funcionais é que eles podem ser difíceis de verificar. Os usuários ou os clientes frequentemente definem esses requisitos como metas gerais, como usabilidade, capacidade do sistema para se recuperar de falhas ou resposta rápida ao usuário. Essas metas vagas causam problemas para os desenvolvedores de sistema, pois eles deixam o escopo para interpretação e discussão subsequentes, depois que o sistema é entregue.

Sempre que possível, você deve escrever os requisitos não funcionais quantitativamente, de modo que eles possam ser testados objetivamente. Na Tabela 3.1 podemos ver uma série de métricas possíveis de serem utilizadas para especificar as propriedades não funcionais do sistema. Aí você pode medir essas características quando o sistema estiver sendo testado para verificar se ele atendeu ou não aos requisitos não funcionais; os clientes de um sistema podem considerar praticamente impossível traduzir suas metas em requisitos quantitativos. Para algumas metas, como facilidade de manutenção, não existem métricas. Eles não compreendem o que um número que define a confiabilidade necessária, por exemplo, significa em termos de sua experiência diária com sistemas de computador.

**Tabela 3.1 Métricas de especificação de requisitos não funcionais**

| Propriedade              | Medida   |
|--------------------------|--|
| <b>Velocidade</b>        | Transações processadas por segundo<br>Tempo de respostas de usuário por evento<br>Tempo de atualização da tela                 |
| <b>Tamanho</b>           | Kbytes<br>Número de memórias RAM   |
| <b>Facilidade de Uso</b> | Tempo de treinamento<br>Número de frames de ajuda  |
| <b>Confiabilidade</b>    | Tempo médio de falhas<br>Probabilidade de indisponibilidade<br>Taxa de ocorrência de falhas<br>Disponibilidade                 |
| <b>Robustez</b>          | Tempo para reiniciar após a falha<br>Porcentual de eventos que causam falhas<br>Probabilidade de corrupção de dados por falhas |
| <b>Portabilidade</b>     | Percentagem de declarações dependentes<br>Número de sistemas-alvo  |

Fonte: Sommerville (2007, p. 84).

Os requisitos não funcionais frequentemente entram em conflito e interagem com outros requisitos funcionais e não funcionais.

Será útil se pudermos diferenciar os requisitos funcionais e não funcionais no documento de requisitos, mas, na prática, isso é fácil de ser feito, uma vez que, se os requisitos não funcionais forem definidos separadamente dos funcionais, será difícil verificar os relacionamentos entre eles. Porém, se eles forem definidos com os requisitos funcionais, podemos ter dificuldade em separar as considerações funcionais e não funcionais e identificar os requisitos relacionados ao sistema como um todo. Mas, devemos, então, destacar explicitamente os requisitos claramente relacionados com as propriedades do sistema, como o desempenho e a confiabilidade, e desse modo colocar os requisitos em uma seção distinta do documento de requisitos ou diferenciando-os, de alguma forma, dos demais requisitos de sistema.

### **1.1.1.3 Requisitos de domínio**

Os requisitos de domínio são derivados do domínio de aplicação do sistema, em vez das necessidades específicas dos usuários do sistema. Podem ser novos requisitos funcionais em si, mas também restringir os requisitos funcionais existentes ou estabelecer como cálculos específicos devem ser realizados. Como esses requisitos são especializados, os engenheiros de software muitas vezes encontram dificuldade em compreender como eles estão relacionados a outros requisitos do sistema.

Os requisitos de domínio são importantes porque refletem os fundamentos do domínio da aplicação. Caso esses requisitos não sejam satisfeitos, pode ser difícil fazer que o sistema funcione satisfatoriamente.

Sommerville (2007, p. 85) ilustra os requisitos de domínio por meio da especificação de requisitos de sistemas de proteção de trem automatizado, na qual cálculos devem ser realizados, pois esse sistema para automaticamente um trem caso ele ultrapasse um sinal vermelho. Com esse requisito é definido como a desaceleração do trem é calculada pelo sistema, usando para isso a terminologia específica do domínio; para que seja comprehensível, é necessário algum conhecimento sobre operação de sistemas ferroviários e das características do trem. Tal exemplo ilustra um grande problema com os requisitos de domínio: na maioria das vezes eles são redigidos na linguagem de domínio da aplicação (equações matemáticas, nesse caso), e geralmente os engenheiros de software têm dificuldade em comprehendê-los. Os especialistas de domínio podem deixar determinadas informações fora de um requisito, simplesmente por serem muito óbvias para eles.

### **1.1.2 Requisitos de usuário**

Os requisitos de usuário de um sistema devem descrever os requisitos funcionais e não funcionais, de uma maneira que sejam comprehensíveis pelos usuários que não possuam um conhecimento técnico do sistema. Esses requisitos devem especificar apenas o comportamento externo e evitar características de projeto do sistema. Outro cuidado que devemos ter enquanto escrevemos os requisitos de usuário é não usar

jargões de software, notações estruturadas ou formais ou descrever os requisitos por meio da implementação do sistema. Devemos sempre que possível ao escrever os requisitos de usuário usar uma linguagem simples, com tabelas e formulários simples e diagramas intuitivos.

Entretanto, no meio do caminho alguns problemas podem surgir quando os requisitos são redigidos em um documento com texto em linguagem natural, tais como:

- a) **Falta de clareza.** Às vezes, é difícil usar a linguagem de maneira precisa e não ambígua sem tornar o documento prolixo e difícil de ler.
- b) **Confusão de requisitos.** Os requisitos funcionais, não funcionais, metas de sistema e informações de projeto podem não estar claramente diferenciados.
- c) **Fusão de requisitos.** Vários requisitos diferentes podem ser expressos juntos como um único requisito.

É uma boa prática separar os requisitos de usuário dos requisitos mais detalhados do sistema em um documento de requisitos. De outro lado, os leitores não técnicos dos requisitos de usuário podem ser sobrecarregados por detalhes relevantes apenas para os técnicos. Os requisitos de usuário que incluem muitas informações restringem a liberdade do desenvolvedor do sistema de apresentar soluções inovadoras para os problemas de usuário e são difíceis de ser compreendidos. Esse requisito de usuário deve simplesmente enfocar os recursos principais a serem fornecidos.

Sempre que for possível, devemos tentar associar uma justificativa lógica a cada requisito do usuário. A justificativa deve esclarecer por que o requisito foi incluído, e é particularmente útil quando os requisitos são mudados.

Para minimizar os mal-entendidos ao redigir requisitos de usuário, Sommerville (2007, p. 86) recomenda que devemos usar algumas diretrizes simples:

1. Invente um formato padrão e assegure se de que todas as definições de requisitos aderiram a esse formato. A padronização de formato toma as omissões menos prováveis e os requisitos mais fáceis de serem verificados. O formato que uso mostra o requisito inicial em negrito, inclui uma declaração da justificativa lógica de cada requisito de usuário e uma referência à especificação detalhada de requisitos do sistema. Você pode também incluir informações sobre quem propôs o requisito de modo que se saiba a quem consultar se o requisito tiver de ser mudado.
2. Use a linguagem de forma consistente. Você deve sempre fazer distinção entre requisitos obrigatórios e desejáveis. Requisitos obrigatórios são aqueles a que o sistema deve atender e são escritos com o uso da palavra ‘deve’. Requisitos desejáveis não são essenciais e são escritos com o uso da palavra ‘pode’.
3. Use destaque no texto (negrito, itálico ou cor) para ressaltar as partes principais do requisito.
4. Evitar, sempre que possível, o uso de jargões de informática. Contudo, inevitavelmente aparecerão termos técnicos detalhados nos requisitos de usuário.

### 1.1.3 Requisitos de sistema

Os requisitos de sistema são versões expandidas dos requisitos de usuário, agora sendo utilizados pelos engenheiros de software como ponto de partida para o projeto do sistema. Esses engenheiros de software adicionam detalhes e explicam como os requisitos de usuário devem ser fornecidos pelo sistema e também podem ser usados como parte do contrato para a implementação do sistema e devem, dessa forma, ser uma especificação completa e consistente de todo o sistema.

Idealmente, os requisitos de sistema devem descrever o comportamento externo do sistema bem como suas restrições operacionais, não devendo ficar relacionados com a forma na qual o sistema pode ser projetado ou implementado. Porém, para especificar completamente um software complexo no nível de detalhamento necessário, é impossível, na prática, excluir todas as informações de projeto. Podemos identificar algumas razões para tal:

- a) A necessidade de projetar uma arquitetura inicial do sistema para auxiliar na estruturação da especificação de requisitos, os quais são organizados de acordo com os diferentes subsistemas que constituem o sistema.
- b) Na maioria das vezes, os sistemas devem interoperar com outros sistemas existentes, com isso restringindo o projeto; essas restrições impõem novos requisitos ao novo sistema.
- c) Utilizar uma arquitetura específica para satisfazer os requisitos não funcionais (tal como programação de  $n$  versões para conseguir a confiabilidade) pode ser necessário. Um órgão certificador externo que necessite certificar que o sistema é seguro pode especificar que um projeto de arquitetura que já tenha sido certificado seja usado.

O uso de uma linguagem natural é muito útil para redigir especificações de requisitos de sistema bem como de usuário. Porém, como os requisitos de sistema são mais detalhados que os de usuário, as especificações em linguagem natural podem ser difíceis de ser compreendidas. Dessa forma:

- a) A compreensão da linguagem natural depende do uso das mesmas palavras para os mesmos conceitos pelos leitores e pelos elaboradores da especificação e isso pode levar a mal-entendidos por causa de palavras com duplo sentido da linguagem natural.
- b) Especificar requisitos em linguagem natural é flexível demais, pois podemos dizer a mesma coisa de maneiras completamente diferentes, ficando a cargo do leitor descobrir quando os requisitos são ou não os mesmos.
- c) Não há uma forma fácil de padronizar os requisitos usando uma linguagem natural. Pode ser difícil encontrar todos os requisitos relacionados e, para descobrir as consequências de uma mudança, às vezes é necessário verificar todos os requisitos em vez de apenas um grupo de requisitos relacionados.

Por causa de problemas, as especificações de requisitos escritas em linguagem natural são propensas a mal-entendidos. Eles frequentemente não são descobertos até as fases finais do processo de software e, portanto, sua solução pode ser muito onerosa

### 1.1.3.1 Especificações em linguagem estruturada

A linguagem natural estruturada é uma forma de escrever os requisitos de sistema, na qual a liberdade do elaborador de requisitos é limitada e todos os requisitos são redigidos de maneira padronizada. A vantagem dessa abordagem é que ela mantém a maior parte da facilidade de expressão e compreensão da linguagem natural, mas assegura que algum grau de uniformidade seja imposto nas especificações. As notações de linguagem estruturada limitam a terminologia que pode ser usada e usa templates para especificar os requisitos de sistema. Elas podem incorporar construções de controle derivadas de linguagens de programação e destaques gráficos para dividir a especificação.

### 1.1.4 Documento de requisitos de software

O documento de requisitos de software também chamado de especificação de requisitos de software ou SRS (Software Requirements Specification) é a declaração oficial do que os desenvolvedores de sistema devem implementar, devendo incluir os requisitos de usuário de um sistema e uma especificação detalhada dos requisitos de sistema; em alguns casos, os requisitos de usuário e de sistema podem estar integrados em uma única descrição. Já em algumas situações, os requisitos de usuário estão definidos em uma introdução à especificação dos requisitos de sistema e, nesse caso, se houver um grande número de requisitos, os requisitos detalhados de sistema podem ser apresentados em um documento separado.

O documento de requisitos possui um conjunto diversificado de usuários, desde a gerência sênior da organização, que paga pelo sistema, até os engenheiros responsáveis pelo desenvolvimento do software. A Tabela 3.2 apresenta os possíveis usuários do documento de requisitos e como fazem uso deles.

**Tabela 3.2 Usuário de um documento de requisitos**

| Usuários                                    | Uso  |
|---|--|
| <b>Clientes de sistema</b>                  | Especificam e leem os requisitos para verificar se eles atendem às suas necessidades<br>Os clientes especificam as mudanças nos requisitos |
| <b>Gerentes</b>                             | Usam o documento de requisitos para planejar um pedido de proposta para o sistema e planejar o processo de desenvolvimento do sistema      |
| <b>Engenheiros de sistema</b>               | Usam os requisitos para compreender qual sistema será desenvolvido   |
| <b>Engenheiros de testes de sistema</b>     | Usam os requisitos para desenvolver testes de validação para o sistema   |
| <b>Engenheiros de manutenção de sistema</b> | Usam os requisitos para compreender o sistema e os relacionamentos entre as partes   |

Fonte: Adaptada de Kotonya e Sommerville (1998 apud SOMMERVILLE, 2007, p. 91).

Como há uma grande diversidade de possíveis usuários, isso nos leva a crer que o documento de requisitos precisa ser um compromisso entre a comunicação dos requisitos para os clientes, a definição dos requisitos em detalhes precisos para os desenvolvedores e testadores e a inclusão de informações sobre uma possível evolução do sistema. O nível de detalhamento a ser incluído em um documento de requisitos depende do tipo de sistema que está sendo desenvolvido e do processo de desenvolvimento usado. No caso de o sistema utilizar um desenvolvedor externo, as especificações de sistema crítico devem ser precisas e muito detalhadas. Já se usarmos um processo de desenvolvimento interno e iterativo, o documento de requisitos pode ser muito menos detalhado e qualquer ambiguidade será resolvida durante o desenvolvimento do sistema.



### *Para saber mais*

Para saber mais sobre documentos de requisitos acesse:

<<http://www.devmedia.com.br/artigo-engenharia-de-software-10-documento-de-requisitos/11909>>.

De acordo com Sommerville (2007), uma série de grandes organizações, dentre as quais o Departamento de Defesa dos EUA e o IEEE, definiu padrões para documentos de requisitos. O padrão do IEEE sugere a seguinte estrutura para os documentos de requisitos:

#### **1. Introdução**

- 1.1 Propósito do documento de requisitos
- 1.2 Escopo do produto
- 1.3 Definições, acrônimos e abreviaturas
- 1.4 Referências
- 1.5 Visão geral do restante do documento

#### **2. Descrição geral**

- 2.1 Perspectiva do produto
- 2.2 Funções do produto
- 2.3 Características dos usuários
- 2.4 Restrições gerais
- 2.5 Suposições e dependências

#### **3. Requisitos específicos**

Abrangem requisitos funcionais, não funcionais e de interface. Esta é, obviamente, a parte mais substancial no documento, mas, pela grande variação na prática organizacional, não é apropriado definir uma estrutura-padrão para esta seção. Os requisitos podem documentar interfaces externas, descrever a

funcionalidade e o desempenho do sistema, especificar requisitos lógicos de banco de dados, restrições de projeto, propriedades emergentes do sistema e características de qualidade.

#### 4. Apêndices

#### 5. Índice

Naturalmente, as informações incluídas em um documento de requisitos devem depender do tipo de software que está sendo desenvolvido e da abordagem usada para o desenvolvimento. Caso seja adotada uma abordagem evolucionária para um produto de software, o documento de requisitos deixará de fora vários capítulos detalhados sugeridos anteriormente e o foco será a definição dos requisitos de usuário e dos requisitos não funcionais de alto nível do sistema. Nesse caso, os projetistas e os programadores usam suas avaliações para decidir como atender aos requisitos de usuários delineados para o sistema.

Por outro lado, quando o software é parte de um grande projeto de engenharia que inclui a interação de sistemas de hardware e de software, frequentemente é indispensável definir os requisitos em um nível refinado de detalhes. Para documentos extensos, é importante incluir uma tabela abrangente de conteúdo e índice do documento, para que os leitores possam encontrar as informações de que precisam.

Os documentos de requisitos são indispensáveis quando um fornecedor externo está desenvolvendo o sistema de software, os métodos ágeis de desenvolvimento argumentam que os requisitos mudam tão rapidamente que um documento de requisitos fica desatualizado tão logo seja redigido, por isso o esforço é desperdiçado. Ao contrário de um documento formal, abordagens como *eXtreme Programming* propõem que os requisitos de usuário sejam coletados de maneira incremental e escritos em cartões. O usuário, então, prioriza os requisitos a serem implementados no próximo incremento do sistema.



#### Para saber mais

Para que você possa fixar melhor os conhecimentos leia

<<http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-engenharia-de-requisitos/8034>>.



#### Questões para reflexão

Finalizando esta seção, gostaria que você refletisse sobre a importância de coletar corretamente os requisitos com os usuários principalmente os requisitos não funcionais, pois é através deles que definimos as restrições do sistema.



## Atividades de aprendizagem

1. Análise as seguintes afirmações sobre a análise de requisitos de software.
  - I. Deve-se dar uma maior ênfase aos aspectos tecnológicos de implementação.
  - II. É importante a participação do usuário para a correta definição do escopo e dos critérios para a avaliação da qualidade do software.
  - III. Os desenvolvedores devem se concentrar na definição dos domínios funcionais, comportamentais e de informação de um problema.

Sobre as afirmações, pode-se dizer que está correta:

  - a) Apenas I.
  - b) Apenas I e II.
  - c) Apenas I e III.
  - d) Apenas II e III.
  - e) Todas as afirmações.
2. Levando em consideração a lista de requisitos de um sistema que será desenvolvido, a seguir.
  - I. O sistema deverá emitir relatórios de compras a cada 15 dias.
  - II. O sistema só permitirá a visualização do campo “valor máximo” para gerentes.
  - III. O sistema deverá fornecer diariamente o relatório de despesas.
  - IV. O sistema não poderá excluir um fornecedor do cadastro se o fornecedor estiver inadimplente.
  - V. O sistema não permitirá acesso aos registros de compras após as 17h.

Considerando os requisitos anteriores, é correto afirmar que:

  - a) I e V são requisitos não funcionais e II, III e IV são requisitos funcionais.
  - b) somente o requisito V é não funcional.
  - c) I e V são requisitos funcionais e II, III e IV são requisitos não funcionais.
  - d) são todos requisitos não funcionais.
  - e) são todos requisitos funcionais.

## Seção 2 Engenharia de requisitos

Compreender os requisitos de um problema está entre as tarefas mais difíceis que um engenheiro de software tem pela frente, e, quanto mais pensa nisso, mais difícil parece ser enfrentá-lo. Pois o cliente não sabe geralmente o que é necessário em um sistema, os usuários finais não têm um bom entendimento das características e funções que o software vai oferecer.

Os clientes e usuários finais devem ser explícitos quanto às suas necessidades e que vão se modificar ao longo do projeto, por isso a engenharia de requisitos é difícil.

Nesta seção, vamos, então, nos aprofundar um pouco mais sobre o assunto.

### 2.1 Engenharia de requisitos

O objetivo do processo de engenharia de requisitos é criar e manter um documento de requisitos de sistema. O processo geral inclui quatro subprocessos de alto nível de engenharia de requisitos, sendo o primeiro relacionado ao **estudo de viabilidade**, ou seja, à avaliação de se o sistema é útil para a empresa (estudo de viabilidade); o segundo com a **elicitação e análise**, isto é com a obtenção de requisitos; o terceiro com **especificação**, ou seja, com a conversão desses requisitos em alguma forma padrão; e por fim a **validação**, que é a checagem se os requisitos realmente definem o sistema que o cliente deseja.

A Figura 3.3 mostra o relacionamento das atividades do processo de engenharia de requisitos, bem como os documentos produzidos em cada estágio do processo.

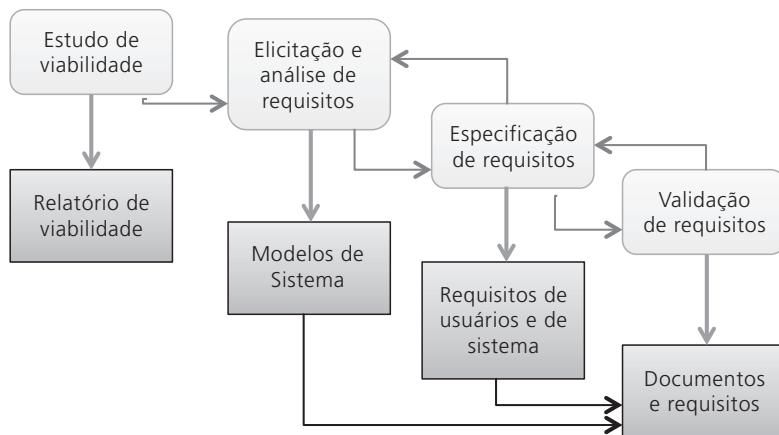


#### Para saber mais

Para saber mais sobre técnicas de levantamento de requisitos leia:

<<http://www.devmedia.com.br/artigo-engenharia-de-software-10-documento-de-requisitos/11909>>.

**Figura 3.3 Processo de engenharia de requisitos**



Fonte: Adaptada de Sommerville (2007, p. 96).

As atividades mostradas na Figura 3.3 relacionam-se à obtenção, documentação e validação de requisitos, sendo que na prática todos esses requisitos se alternam, e as pessoas desenvolvem um maior entendimento naquilo que desejam que o sistema faça, a organização que está comprando o sistema muda e, dessa forma, poderão ser feitas mudanças no hardware, no software e no próprio ambiente do sistema; é a isso que chamamos de gerenciamento de requisitos.

Algumas pessoas consideram a engenharia de requisitos um processo no qual se aplica um método estruturado de análise (por exemplo, análise orientada a objetos, ou análise estruturada). Porém isso fica com a parte de análise de sistemas e no desenvolvimento de um conjunto de modelos gráficos de sistemas, como modelos de Use Case (casos de uso), que têm como objetivo servir como uma especificação de sistema. O conjunto de modelos descreve o comportamento do sistema e inclui anotações com descrições adicionais de informações, por exemplo, o desempenho ou a confiabilidade necessária.

Embora os métodos estruturados possuam um papel a desempenhar no processo de engenharia de requisitos, a elicitação de requisitos, em particular, é uma atividade centrada em pessoas, e as pessoas não gostam das restrições impostas pelos modelos rígidos de sistema.

## 2.2 Estudos de viabilidade

No início de todos os sistemas novos, o processo de engenharia de requisitos deve começar com um estudo de viabilidade, em que a entrada dessa atividade consiste de um conjunto inicial dos requisitos de negócios, de um esboço da descrição do sistema e de como o sistema pretende apoiar os processos de negócios. Os resultados do estudo de viabilidade devem estar descritos em um relatório que recomenda ou

não prosseguir com os processos de engenharia de requisitos e de desenvolvimento do sistema em questão.

O estudo de viabilidade é um estudo breve e focado que procura responder às seguintes questões:

- a) O sistema contribui para os objetivos gerais da organização?
- b) O sistema pode ser implementado com tecnologia atual e dentro das restrições definidas de custo e prazo?
- c) O sistema pode ser integrado a outros sistemas já implantados?

Como podemos verificar se o sistema contribui ou não para os objetivos da empresa? Se o sistema não apoia esses objetivos, ele não tem valor real para a empresa. Mesmo que pareça óbvio, muitas empresas desenvolvem sistemas que não contribuem para seus objetivos, porque muitas delas não possuem um perfil claro desses objetivos, pois falham na definição dos requisitos de negócios para o sistema, ou porque outros fatores (políticos ou organizacionais) influenciaram na compra do sistema.

Na realização de um estudo de viabilidade, é necessário fazer uma avaliação de informações, sua coleta e montagem de um relatório. Logo após a identificação das informações, é necessário falar com as fontes de informações no intuito de descobrir as respostas a essas questões. Para a coleta de tais informações algumas perguntas poderiam ser feitas:

- a) Como a organização se comportaria se esse sistema não fosse implementado?
- b) Quais são os problemas com os processos atuais e como o novo sistema poderia ajudar na redução desses problemas?
- c) Qual será a contribuição direta do sistema para os objetivos e requisitos da empresa?
- d) As informações podem ser integradas (trocadas) com outros sistemas da organização?
- e) O sistema necessitará de tecnologias que a empresa ainda não possui?
- f) Quais processos podem e devem ser apoiados pelo sistema e quais não precisam ser apoiados?

Em um estudo de viabilidade, é conveniente consultar fontes de informações tais como os gerentes dos setores nos quais o sistema será usado, os engenheiros de software já familiarizados com o tipo de sistema proposto, os especialistas em TI e os usuários finais do sistema. Normalmente, o tempo para concluir tal estudo varia de duas ou três semanas.

De posse dessas informações, monte o relatório de estudo de viabilidade, no qual você deve fazer uma recomendação se o desenvolvimento do sistema deve ou não prosseguir. Ainda no relatório, podem-se propor mudanças de escopo, no orçamento e no prazo e também sugerir requisitos de alto nível adicionais para o sistema.

## 2.3 Elicitação e análise de requisitos

Nessa atividade, os engenheiros de software trabalham com os clientes e os usuários finais do sistema para aprender sobre o domínio da aplicação, quais serviços o sistema deve fornecer, o desempenho esperado do sistema, restrições de hardware etc.

A elicitação e a análise de requisitos podem envolver várias pessoas de uma organização. O *stakeholder* é qualquer pessoa ou grupo afetado pelo sistema, direta ou indiretamente, incluindo os usuários finais que interagem com o sistema e todo pessoal na empresa que possa ser afetado por ele. Também podemos entender como outros *stakeholders* no sistema os engenheiros que estão desenvolvendo ou mantendo sistemas relacionados, os gerentes de negócios, os especialistas do domínio e até os representantes do sindicato.

As razões da dificuldade dos *stakeholders* na elicitação e na compreensão dos requisitos decorrem de:

- Frequentemente não sabem o que querem do sistema a não ser em termos gerais.
- Expressam os requisitos naturalmente em seus próprios termos e com o conhecimento implícito de seu trabalho.
- Vários *stakeholders* possuem diferentes requisitos, expressos de diferentes formas e os engenheiros de requisitos precisam considerar todas as fontes potenciais de requisitos e descobrir pontos em comum e conflitos.
- Fatores políticos podem influenciar os requisitos do sistema, pois os gerentes podem solicitar requisitos específicos do sistema que poderá aumentar sua influência na organização.
- O próprio ambiente econômico e de negócios sobre o qual a análise é realizada é dinâmico, ou seja, muda inevitavelmente durante o processo de análise e, dessa forma, a importância de determinado requisito pode mudar.

As atividades de processo de elicitação e análise de requisitos são:

| Atividades                                       | Descrição   |
|--|---|
| <b>Obtenção de requisitos</b>                    | É o processo de interação com os <i>stakeholders</i> no intuito de coletar seus requisitos, incluindo os requisitos de domínio que também são descobertos durante essa atividade, provenientes dos <i>stakeholders</i> e da documentação.           |
| <b>Classificação e organização de requisitos</b> | Essa atividade envolve a coleta de requisitos não estruturados, agrupando os requisitos relacionados e organizando-os em conjuntos coerentes.   |
| <b>Priorização e negociação de requisitos</b>    | Quando vários <i>stakeholders</i> participam do processo, inevitavelmente os requisitos serão conflitantes. Essa atividade está relacionada à priorização de requisitos, à procura e à resolução de conflitos de requisitos por meio da negociação. |
| <b>Documentação de requisitos</b>                | Os requisitos são documentados, podendo ser produzidos documentos de requisitos formais ou informais.   |

A elicitação e análise de requisitos é um processo iterativo com realimentação contínua de cada atividade para outras atividades. É como um ciclo começado com a obtenção de requisitos e terminando com a documentação de requisitos. O entendimento dos requisitos pelo analista aumenta a cada volta do ciclo.

A classificação e a organização de requisitos estão relacionadas principalmente com a identificação da sobreposição de requisitos dos diferentes *stakeholders* e o agrupamento de requisitos relacionados. O modo mais comum de agrupamento de requisitos é usar um modelo de arquitetura de sistema para identificar os subsistemas e associar os requisitos a cada um deles. Dessa forma a engenharia de requisitos e o projeto de arquitetura nem sempre podem estar separados.

Uma vez que os *stakeholders* possuem visões diferentes sobre a importância e a prioridade dos requisitos e essas visões entram em conflito, devemos organizar negociações contínuas com os *stakeholders*, a fim de que possamos atingir os compromissos definidos. É quase impossível satisfazer completamente a todos os *stakeholders*, pois, se alguns deles perceberem que suas visões não foram consideradas de maneira apropriada, eles podem tentar boicotar intencionalmente o processo de engenharia de requisitos (RE — Requirements Engineering).

Já no estágio de documentação, os requisitos são documentados de uma forma que possam ser usados para ajudar as próximas obtenções de requisitos.

### 2.3.1 Obtenção de requisitos

A obtenção de requisitos é o processo em que reunimos as informações sobre o sistema proposto e os existentes no intuito de obter os requisitos de usuário e de sistema com base nessas informações. As fontes de informações, nessa fase de obtenção de requisitos, incluem documentação, os *stakeholders* de sistema e especificações de sistemas similares. Já a interação com os *stakeholders* ocorre por intermédio de entrevistas e observações, podendo fazer uso de cenários e protótipos como auxílio na obtenção dos requisitos.

Além dos *stakeholders* no sistema, os requisitos podem ser oriundos do domínio de aplicação e de outros sistemas que interagem com o que está sendo especificado, onde esses dados devem ser considerados durante o processo de elicitação de requisitos.

Todas essas fontes de requisitos (*stakeholders*, domínio, sistemas) podem ser representadas como pontos de vista do sistema, onde cada ponto de vista apresenta um subconjunto de requisitos do sistema e cada ponto de vista fornece uma perspectiva nova do sistema, porém, essas perspectivas não são completamente independentes uma vez que elas geralmente se sobrepõem de modo que haja requisitos comuns.

#### 2.3.1.1 Pontos de vista

As abordagens orientadas a pontos de vista para engenharia de requisitos organizam o processo de elicitação e os próprios requisitos usando pontos de vista. Um ponto forte da análise orientada a pontos de vista é que ela reconhece várias pers-

pectivas e fornece um framework para descobrir conflitos nos requisitos propostos por diferentes *stakeholders*.

Os pontos de vista podem ser usados como um meio de classificação de *stakeholders* e de outras fontes de requisitos. Existem três tipos genéricos de pontos de vista:

1. *Pontos de vista de interação*: representam pessoas ou outros sistemas que interagem diretamente com o sistema.
2. *Pontos de vista indiretos*: representam os *stakeholders* que não usam o sistema diretamente, mas que influenciam os requisitos de alguma forma.
3. *Pontos de vista de domínio*: representam características e restrições de domínio que influenciam os requisitos de sistema.

Tipicamente, esses pontos de vista fornecem diferentes tipos de requisitos. Os pontos de vista de interação fornecem requisitos de sistema detalhados que abrangem as características e as interfaces do sistema. Os pontos de vista indiretos são os que mais provavelmente fornecem requisitos e restrições organizacionais de alto nível. Os pontos de vista de domínio fornecem normalmente as restrições de domínio que se aplicam ao sistema.

Os pontos de vista de engenharia de requisitos podem ser importantes por duas razões: primeiro, os engenheiros que desenvolvem o sistema podem ter experiência com sistemas similares e, dessa forma, sugerem requisitos levando em conta sua experiência; e, segundo, o pessoal técnico que deve gerenciar e manter o sistema pode ter requisitos que ajudarão a simplificar o apoio ao sistema.

Assim sendo, os pontos de vista que fornecem requisitos podem ser provenientes dos departamentos de marketing ou de negócios externos em uma organização. É importante em sistemas de e-commerce e softwares comerciais. Nos sistemas baseados na web devem apresentar uma imagem favorável da empresa, e ao mesmo tempo oferecer funcionalidade ao usuário. Já no que diz respeito a produtos de software, o departamento de marketing deve conhecer as características que tornarão o sistema mais atraente aos compradores.

Em qualquer sistema não trivial, há um grande número de possíveis pontos de vista, e é quase impossível eliciar os requisitos baseado em todos eles. Dessa maneira, é importante que você organize e estruture os pontos de vista hierarquicamente, onde pontos de vista do mesmo ramo provavelmente compartilharão requisitos comuns e, depois que os pontos de vista forem identificados e estruturados, podemos, então, identificar os mais importantes e iniciar com eles a obtenção dos requisitos do sistema.

### 2.3.1.2 *Entrevista*

Nos processos de engenharia de requisitos, as entrevistas com os *stakeholders* são corriqueiras, e podem ser formais ou informais. Nas elas são formuladas questões para os *stakeholders* sobre o sistema que eles usam e sobre o sistema a ser desenvolvido e os requisitos são obtidos das respostas a essas questões. Podemos classificar as entrevistas em dois tipos:

- └ Entrevistas fechadas, nas quais o *stakeholder* responde a um conjunto de perguntas predefinidas.
- └ Entrevistas abertas, nas quais não existe um roteiro predefinido; a equipe de engenharia de requisitos explora vários assuntos com os *stakeholders* no sistema e desenvolve uma compreensão maior de suas necessidades.

Na prática, as entrevistas com os *stakeholders* são uma combinação desses tipos. As entrevistas são úteis para obter um entendimento geral sobre o que os *stakeholders* fazem, como eles podem interagir com o sistema e as dificuldades que enfrentam com os sistemas atuais.

É difícil eliciar o conhecimento de domínio durante as entrevistas por dois motivos:

- a) O uso de terminologia e jargões específicos por parte dos especialistas de domínio. É impossível para os especialistas explicar os requisitos de domínio sem usar tal terminologia. Em geral a usam de maneira precisa e específica, o que facilmente provoca mal-entendidos por parte dos engenheiros de requisitos.
- b) A familiaridade por parte dos *stakeholders* com alguns conhecimentos de domínio que são considerados difíceis de explicar ou são tão fundamentais que não vale a pena mencioná-los.

A entrevista não é uma técnica eficiente para elicitação de conhecimentos sobre os requisitos e as restrições organizacionais, pois existem relacionamentos sutis de poder e influência entre os *stakeholders* na organização.

Com relação às características que podemos elencar a respeito dos entrevistadores podemos citar:

1. Possuem mente aberta, ou seja, evitam ideias preconcebidas sobre os requisitos e buscam ouvir os *stakeholders*. Se estes propuserem requisitos inovadores, os entrevistadores estão dispostos a mudar de ideia sobre o sistema.
2. Ao mesmo tempo que induzem os entrevistados a iniciar as discussões com uma questão, uma proposta de requisitos ou sugerindo um trabalho conjunto em um protótipo do sistema, dizer às pessoas “conte-me o que você quer” provavelmente trará informações úteis como resultado.

### **2.3.1.3 Cenários**

Em geral as pessoas acham mais fácil citar exemplos da vida real do que abstrair descrições das funções. Elas podem compreender e criticar um cenário de como interagiriam com um determinado sistema. Já os engenheiros de requisitos podem usar as informações obtidas nessa discussão para elaborar os requisitos reais do sistema.

Os cenários podem ser particularmente úteis para adicionar detalhes a um esboço da descrição de requisitos, em que cada cenário abrange uma ou mais interações possíveis e diversos tipos de cenários podem ser desenvolvidos, cada um dos quais fornecendo diferentes tipos de informações sobre o sistema em diferentes níveis de

detalhamento. Em métodos ágeis, são usados cenários para descrever seus requisitos, como é o caso do eXtremeProgramming.

O uso da descrição de um cenário começa com um esboço da interação e, durante a etapa de elicitação, os detalhes são adicionados no intuito de criar uma descrição completa dessa interação.

De acordo como Sommerville (2007, p. 102), um cenário deve incluir:

1. Uma descrição do que os usuários esperam do sistema no início do cenário.
2. Uma descrição do fluxo normal de eventos no cenário.
3. Uma descrição do que pode dar errado e como isso é tratado.
4. Informações sobre outras atividades que podem ocorrer simultaneamente.
5. Uma descrição do estado de sistema no fim do cenário.

A elicitação baseada em cenários pode ser realizada informalmente. Os cenários podem ser escritos na forma de textos e complementados por diagramas, imagens de computador etc.

### **2.3.1.4 Casos de uso (use case)**

É uma técnica baseada em cenários para elicitação de requisitos. Tornaram-se uma característica fundamental da notação UML para descrição de modelos de sistema orientado a objetos e, em sua notação mais simples, um caso de uso identifica o tipo da interação e os agentes envolvidos.

A Figura 3.4 mostra os elementos essenciais da notação de caso de uso, em que os atores do processo são representados como bonecos, e cada caso de uso (classe de interação) é representada como uma elipse identificada. O conjunto de casos de uso representa todas as possíveis interações a serem representadas nos requisitos de sistema.

Os casos de uso identificam as interações individuais com o sistema, de forma que eles podem ser documentados por meio de texto ou de links com os modelos UML que desenvolvem o cenário mais detalhadamente. Já os diagramas de sequência são frequentemente usados para adicionar informações ao caso de uso. Os diagramas de sequência mostram os agentes envolvidos na interação, os objetos com os quais interagem e as operações associadas a esses objetos.

Figura 3.4 Notação de caso de uso



Fonte: Adaptada Sommerville (2007, p. 103).

A UML é um padrão real para a modelagem orientada a objetos, e, assim sendo, os casos de uso e a elicitação baseada em casos de uso têm sido cada vez mais usados para elicitação de requisitos.

Cenários e casos de uso são técnicas eficazes para elicitação de requisitos segundo pontos de vista de interação, onde cada tipo de interação pode ser representado como um caso de uso específico. Eles também podem ser usados em conjunto com alguns pontos de vista indiretos, sendo que esses pontos de vista recebem alguns resultados (como um relatório gerencial) do sistema. Contudo, como eles enfocam as interações, não são eficazes para eliciar restrições ou requisitos de negócios e não funcionais de alto nível com base nos pontos de vista indiretos ou para obter requisitos de domínio.

### 2.3.2 Etnografia

Os sistemas de software são usados dentro de um contexto social e organizacional, não existindo isoladamente, e seus requisitos podem ser derivados ou restringidos por esse contexto. Satisfazer a esses requisitos sociais e organizacionais é essencial para o sucesso do sistema, pois uma das razões pelas quais vários sistemas de software são entregues, porém nunca usados, é que não é dada a devida importância a esses requisitos.

Segundo Sommerville (2007, p. 104), “a **etnografia** é uma técnica de observação que pode ser usada para compreender os requisitos sociais e organizados, onde um analista se insere no ambiente de trabalho onde o sistema será usado, faz observações do dia a dia e anota as tarefas reais nas quais os participantes estão envolvidos”.

O uso da etnografia presta uma ajuda aos analistas na descoberta de requisitos implícitos de sistema que refletem os processos reais, e não os formais, com os quais as peças estão envolvidas. Geralmente as pessoas consideram muito difícil definir detalhes de seu trabalho diário, pois para elas isso é secundário. Elas compreendem seu próprio trabalho, porém, muitas vezes não compreendem o relacionamento do seu trabalho com os de outros setores da empresa. Outros fatores também afetam o trabalho, tais como fatores sociais e organizacionais, mas que não são óbvios para as pessoas, e que necessitam ser examinados por um observador imparcial para poderem se tornar claros. A diferença entre o trabalho suposto e o real é a razão mais importante pela qual os sistemas de escritório não tiveram efeito significativo na produtividade.

A etnografia é particularmente eficaz para descobrir dois tipos de requisitos:

- *Requisitos derivados da maneira como as pessoas realmente trabalham* em vez da maneira pela qual as definições de processo dizem que elas deveriam trabalhar.
- *Requisitos derivados da cooperação e do conhecimento das atividades de outras pessoas.*

Na Figura 3.5 podemos observar como a etnografia pode ser combinada com a prototipação, pois a etnografia informa o desenvolvimento do protótipo de forma que poucos ciclos de refinamento de protótipo sejam requeridos. Além disso, a prototipação enfoca a etnografia, identificando os problemas e as questões que podem ser

discutidos com o etnógrafo, que deve, então, procurar as respostas a essas questões durante a próxima fase do estudo de sistema (SOMMERVILLE, 2007).

**Figura 3.5 Etnografia e prototipação**



Fonte: Adaptada de Sommerville (2007, p. 105).

Os estudos de etnografia podem revelar detalhes importantes do processo frequentemente ignorados por outras técnicas de elicitação de requisitos. Entretanto por causa de seu foco no usuário final essa abordagem não é adequada na obtenção de requisitos organizacionais e de domínio. Assim, estudos etnográficos nem sempre podem identificar novas características a serem acrescidas a um sistema. Portanto, a etnografia não é uma abordagem completa para elicitação de requisitos e deve ser usada para complementar outras abordagens, como a análise de casos de uso.

## 2.4 Validação de requisitos

A validação de requisitos dedica-se a mostrar que os requisitos de fato definem o sistema que o usuário deseja. Ela se sobrepõe à análise, pois está relacionada à descoberta de problemas com os requisitos.

A importância da validação de requisitos está no fato de que os erros em um documento de requisitos podem acarretar custos excessivos de trabalho quando são descobertos durante o desenvolvimento ou depois que o sistema está em operação.

O custo de correção de um problema de requisitos, fazendo uma mudança de sistema, é muito maior do que a correção de erros de projeto e de codificação. Uma mudança de requisitos significa geralmente que o projeto e a implementação do sistema devem também ser mudados e o sistema deve ser novamente testado.

Durante o processo de validação de requisitos, devem ser realizadas verificações nos requisitos do documento de requisitos como podemos observar no Quadro 3.1 (SOMMERVILLE, 2007).

**Quadro 3.1 Verificações nos documentos de requisitos**

| <b>Tipo</b>                         | <b>Descrição</b>   |
|-------------------------------------|--|
| <b>Verificações de validade</b>     | Um usuário pode pensar que um sistema é necessário para desempenhar determinadas funções.  |
| <b>Verificações de consistência</b> | Os requisitos em um documento não devem ser conflitantes. Isso significa que não devem existir restrições ou descrições contraditórias para a mesma função do sistema.   |
| <b>Verificações de completeza</b>   | O documento de requisitos deve incluir requisitos que definam todas as funções e as restrições desejadas pelo usuário do sistema.  |
| <b>Verificações de realismo</b>     | Usando o conhecimento da tecnologia existente, os requisitos devem ser verificados quanto a se realmente podem ser implementados. Essas verificações também devem levar em consideração o orçamento e o prazo para o desenvolvimento do sistema. |
| <b>Facilidade de verificação</b>    | Para reduzir o potencial de divergências entre cliente e fornecedor, os requisitos do sistema devem sempre ser escritos de modo que sejam verificáveis.  |

Fonte: Adaptado de Sommerville (2007, p. 106).

Ainda segundo Sommerville (2007), uma série de técnicas de validação de requisitos pode ser usada em conjunto ou individualmente, tais como:

- └ *Revisões de requisitos.* Os requisitos são analisados sistematicamente por uma equipe de revisores.
- └ *Prototipação.* Um modelo executável do sistema é apresentado para usuários finais e clientes. Eles podem experimentar o modelo para verificar se atende às suas necessidades reais.
- └ *Geração de casos de teste.* Os requisitos devem ser testáveis; se os testes dos requisitos forem criados como parte do processo de validação, geralmente revelarão problemas de requisitos.

Não podemos subestimar os problemas de validação de requisitos, visto que é difícil demonstrar que um conjunto de requisitos atende às necessidades do usuário. Por outro lado, os usuários devem imaginar o sistema em uso e avaliar sua adequação ao trabalho. Torna-se difícil para profissionais de informática habilidosos realizar esse tipo de análise abstrata e é ainda mais difícil para os usuários do sistema.

### **2.4.1 Revisões de requisitos**

A revisão de requisitos é um processo manual que envolve pessoas de ambas as empresas (do cliente e do fornecedor), onde é verificado o documento de requisitos em busca de anomalias e omissões. O processo de revisão pode ser gerenciado da mesma maneira que as inspeções de programa.

As revisões de requisitos podem ser formais ou informais; as revisões informais envolvem os fornecedores, sendo discutidos os requisitos com o maior número possível

de *stakeholders*. Vários problemas podem ser detectados simplesmente conversando sobre o sistema com eles antes do comprometimento de uma revisão formal.

Na revisão formal de requisitos, a equipe de desenvolvimento deve “levar” o cliente pelos requisitos de sistema, explicando as implicações de cada um. Os revisores podem também checar a:

- ❑ facilidade de verificação;
- ❑ facilidade de compreensão;
- ❑ rastreabilidade;
- ❑ adaptabilidade.

Conflitos, contradições, erros e omissões nos requisitos devem ser apontados pelos revisores e registrados formalmente no relatório de revisão. Dessa forma é de responsabilidade dos usuários, do adquirente de sistema e do desenvolvedor de sistema negociar uma solução para os problemas identificados.

## 2.5 Gerenciamento de requisitos

Os requisitos de sistemas de software de grande porte estão sempre mudando. Como o problema não pode ser totalmente definido, os requisitos de software tendem a ser incompletos. Durante o processo de software, o entendimento dos stakeholders sobre o problema muda constantemente. Esses requisitos devem, então, evoluir para refletir essas novas visões do problema.

Quando os usuários adquirirem experiência com o sistema, eles descobrirão novas necessidades e prioridades:

1. Sistemas de grande porte geralmente têm uma comunidade diversificada de usuários, sendo que esses usuários têm diferentes requisitos e prioridades, os quais podem ser conflitantes ou contraditórios.
2. As pessoas que pagam por um sistema e os usuários de um sistema raramente são as mesmas pessoas. Os clientes do sistema impõem requisitos por causa de restrições organizacionais e de orçamento.
3. O ambiente de negócios e técnico do sistema muda após a instalação, e essas mudanças devem se refletir no sistema. Um novo hardware pode ser introduzido, pode ser necessária uma interface com outros sistemas, as prioridades de negócios podem mudar trazendo consequentes mudanças no apoio ao sistema, e um nova legislação e regulamentos podem ser introduzidos devendo ser implementados no sistema.

O gerenciamento de requisitos é um processo para compreender e controlar as mudanças dos requisitos de sistema. Você precisa manter o acompanhamento dos requisitos individuais e manter as ligações entre os requisitos dependentes, de modo que seja possível avaliar o impacto das mudanças de requisitos.

## 2.5.1 Planejamento de gerenciamento de requisitos

O planejamento é o primeiro estágio essencial no processo de gerenciamento de requisitos. O gerenciamento de requisitos é muito dispendioso. Durante o estágio de gerenciamento de requisitos, você deve decidir sobre:

1. *Identificação de requisitos.*

Cada requisito deve ser identificado unicamente de modo que possa ser feita a referência cruzada entre este e outros requisitos para que ele possa ser usado nas avaliações de rastreabilidade.

2. *Processo de gerenciamento de mudanças.*

É o conjunto de atividades que avaliam o impacto e custo das mudanças.

3. *Políticas de rastreabilidade.*

4. Definem os relacionamentos entre os requisitos e entre os requisitos e o projeto do sistema, que devem ser registrados, e como esses registros devem ser mantidos.

5. *Apoio de ferramentas CASE.*

O gerenciamento de requisitos envolve o processamento de grandes quantidades de informações sobre os requisitos. As ferramentas que podem ser usadas variam desde sistemas especializados de gerenciamento de requisitos a planilhas e sistemas simples de banco de dados.

A rastreabilidade é a propriedade de uma especificação de requisitos que reflete a facilidade de encontrar os requisitos relacionados. Há três tipos de informações de rastreabilidade que podem ser mantidos:

1. Informações de *rastreabilidade da origem* ligam os requisitos aos *stakeholders* que propuseram os requisitos e aos motivos desses requisitos.
2. Informações de *rastreabilidade de requisitos* ligam os requisitos dependentes dentro do documento de requisitos.
3. Informações de *rastreabilidade de projeto* ligam os requisitos aos módulos de projeto, nos quais esses requisitos são implementados.

## 2.5.2 Gerenciamento de mudanças de requisitos

O gerenciamento de mudanças de requisitos deve ser aplicado a todas as mudanças propostas aos requisitos. Ele traz a vantagem de usar um processo formal para gerenciamento de mudanças uma vez que todas as propostas de mudança são tratadas de forma consistente e as mudanças no documento de requisitos são feitas de maneira controlada. No Quadro 3.2 são mostrados os principais estágios para um processo de gerenciamento de mudanças.

**Quadro 3.2 Estágio de processos de gerenciamento de mudanças**

| <b>Estágios</b>                                       | <b>Descrição</b>  |
|---|---|
| <b>Análise do problema e especificação da mudança</b> | O processo se inicia com um problema de requisitos identificado ou, às vezes, com uma proposta de mudança específica.   |
| <b>Análise da mudança e estimativa de custo</b>       | O efeito da mudança proposta é avaliado usando as informações de rastreabilidade e o conhecimento geral sobre os requisitos do sistema. O custo de realizar a mudança é estimado em termos de modificações no documento de requisitos e, se adequado, do projeto e da implementação do sistema. |
| <b>Implementação da mudança</b>                       | O documento de requisitos e, quando necessário, o projeto e a implementação de sistema são modificados. Você deve organizar o documento de requisitos de modo que possa realizar as mudanças sem reescrita ou reorganização extensivas.   |

Fonte: Adaptado de Sommerville (2007, p. 110).

Se uma mudança de requisitos do sistema é solicitada com urgência, existe sempre uma propensão a realizar a mudança no sistema e, depois, retrospectivamente, modificar o documento de requisitos. Isso leva à defasagem entre o documento de requisitos e a implementação do sistema. Geralmente, feitas as mudanças no sistema, as mudanças no documento de requisitos podem ser esquecidas ou realizadas de maneira não consistente com as no sistema.

Os processos iterativos de desenvolvimento, como *extreme programming*, foram definidos para lidar com requisitos que mudam durante o processo de desenvolvimento.



### Questões para reflexão

Considerando o processo de engenharia de requisitos, reflita sobre os motivos que ainda levam muitos desenvolvedores a não fazer o uso de tal processo quando iniciam um projeto de software.



## Atividades de aprendizagem

1. Com relação às atividades descritas na seção, assinale dentre as alternativas a atividade que NÃO faz parte da engenharia de requisitos.
  - a) Estudo de viabilidade.
  - b) Análise de risco.
  - c) Levantamento de necessidades do cliente.
  - d) Verificação.
  - e) Gerenciamento.
2. “[...] é uma restrição sobre os serviços ou sobre as funções oferecidas pelo sistema, podendo ser uma restrição de timing (tempo), sobre o processo de desenvolvimento, sobre o desempenho ou sobre a confiabilidade do sistema etc [...]''. Lendo o trecho, podemos afirmar que se refere a
  - a) Um requisito não funcional.
  - b) Um requisito funcional.
  - c) Especificação de risco.
  - d) A iteração de processo.
  - e) Etnografia.

**Seção 3****Gerenciamento de projetos de software**

O gerenciamento de projetos de software é uma parte essencial da engenharia de software. Um bom gerenciamento não pode garantir o sucesso de um projeto, porém, um mau gerenciamento geralmente resulta em falha do projeto e como consequência o software é entregue com atraso, custa mais do que foi originalmente estimado e falha ao atender seus requisitos.

***Para saber mais***

Aprofundando seus conhecimentos sobre gerência de projetos leia:  
[<http://www.devmedia.com.br/gestao-de-projetos-de-software/9143>](http://www.devmedia.com.br/gestao-de-projetos-de-software/9143).

A responsabilidade dos gerentes de software inicia-se pelo desenvolvimento de planos e cronogramas do projeto: eles supervisionam o trabalho para assegurar que ele esteja sendo realizado dentro dos **padrões exigidos** e monitoram o progresso para verificar se o desenvolvimento está no **prazo** e dentro do **orçamento**.

### **3.1 Gerenciamento de projetos**

A necessidade de gerenciamento de projetos de software está no fato de que a engenharia de software profissional está sempre sujeita às restrições de orçamento e de cronograma da organização. Com isso o trabalho do gerente de projeto de software é assegurar que este atenda a essas restrições e entregue um software que contribua para as metas da empresa que está desenvolvendo o software.

O tipo de trabalho dos gerentes de software e dos outros gerentes de projetos de outras áreas da engenharia é o mesmo, porém o que difere o engenheiro de software é que ele possui algumas distinções que tornam seu gerenciamento mais difícil. O Quadro 3.3 traz algumas destas diferenças.

**Quadro 3.3 Diferenças entre o trabalho do gerente de software ou demais gerentes de projetos**

| Diferença                            | Descrição  |
|--------------------------------------|--|
| <b><i>O produto é intangível</i></b> | O gerente de um projeto de construção de edifício pode ver o produto que está sendo construído e, se o cronograma atrasar, o efeito sobre o produto é visível. Porém o software é intangível, não pode ser visto ou tocado, os gerentes de projetos de software têm dificuldades de verificar seu progresso. Eles contam com outras pessoas para produzir a documentação necessária para examinar o progresso. |

continua

continuação

|   |  |
|---|--|
| <b>Não existem processos-padrão de software</b>                                     | O processo de engenharia de alguns tipos de sistema, como pontes e prédios, é bem compreendido. Entretanto, os processos de software variam de uma empresa para a outra. Mesmo que nossa compreensão sobre esses processos tenha crescido, ainda não podemos prever, confiavelmente, quando determinado processo de software provavelmente apresentará problemas de desenvolvimento. |
| <b>Projetos de software de grande porte são, frequentemente, projetos “únicos”.</b> | Projetos de software de grande porte são diferentes de projetos anteriores. Assim sendo, mesmo com a experiência que os gerentes possuem é difícil prever problemas. Além disso, as mudanças tecnológicas podem tornar a experiência dos gerentes obsoleta, e lições anteriores aprendidas não podem ser reutilizadas em novos projetos.   |

Fonte: Adaptado de Sommerville (2007, p. 62).

Levando em consideração esses problemas, não é surpresa que alguns projetos de software se atrasem, ultrapassem o orçamento e estourem o prazo de entrega. Sistemas de software são geralmente novos e inovadores.

Projetos inovadores de engenharia frequentemente apresentam problemas de cronograma e, em face dessas dificuldades, é surpreendente que tantos projetos de software sejam entregues no prazo e dentro do orçamento.

### 3.2 Atividades de gerenciamento

O trabalho do gerente de software varia muito — sendo impossível definir uma descrição de suas atividades — dependendo da empresa e do produto de software que está em desenvolvimento. Porém boa parte dos gerentes assume a responsabilidade por uma ou por todas as seguintes atividades.

É impossível fornecer uma descrição de trabalho-padrão para um gerente de software. O trabalho varia muito, dependendo da organização e do produto de software que está sendo desenvolvido. No entanto, a maioria dos gerentes assume a responsabilidade, em algum estágio, por algumas ou todas as seguintes atividades definidas no Quadro 3.4.

Quadro 3.4 Atividades de gerenciamento

| Atividades                    | Descrição   |
|-------------------------------|---|
| <b>Elaboração da proposta</b> | A proposta descreve os objetivos do projeto e como ele será realizado, normalmente inclui a estimativa de custos e de cronograma e justifica por que o contrato deve ser concedido a determinada organização ou equipe. A elaboração da proposta é uma tarefa crítica, visto que muitas organizações de software dependem da existência de um número suficiente de propostas aceitas e contratos concedidos. A elaboração da proposta é uma habilidade adquirida pela prática e pela experiência. |

continua

continuação

|  |  |
|--|--|
| <b>Planejamento e desenvolvimento do cronograma do projeto</b> | O planejamento de projeto está relacionado à identificação das atividades, marcos e produtos gerados por um projeto. É elaborado um plano para guiar o desenvolvimento em direção aos objetivos do projeto.  |
| <b>Custo do projeto</b>  | A estimativa de custos é uma atividade relacionada com a estimativa dos recursos necessários para realizar o plano do projeto.   |
| <b>Monitoração e revisões do projeto</b>                       | A monitoração do projeto é uma atividade contínua. O gerente deve manter o acompanhamento do progresso do projeto e comparar o progresso com os custos reais e planejados. A monitoração informal pode prever problemas potenciais do projeto, revelando dificuldades à medida que elas ocorrem. Em vez de aguardar que um atraso no cronograma seja relatado, o gerente de software poderia designar algum especialista para o problema ou propor uma alternativa de programação.   |
| <b>Seleção e avaliação de pessoal</b>                          | Os gerentes geralmente precisam selecionar pessoas para trabalhar em seus projetos. O ideal é que haja o pessoal habilitado com experiência adequada disponível para trabalhar no projeto, mas na maioria dos casos os gerentes precisam aceitar uma equipe de projeto aquém do considerado ideal. As razões para isso são: o orçamento do projeto pode não ser suficiente para contratar uma equipe muito bem remunerada; uma equipe com experiência adequada pode não estar disponível na organização ou externamente e a organização pode querer desenvolver as habilidades de seus funcionários. O gerente de software precisa trabalhar dentro dessas restrições ao selecionar a equipe de projeto. |
| <b>Elaboração de relatórios e apresentações</b>                | Os gerentes de projeto são geralmente responsáveis pela preparação de relatórios sobre o projeto para o cliente e para as organizações contratantes. Eles devem redigir documentos concisos e coerentes que resumam as informações fundamentais com base em relatórios detalhados do projeto.  |

Fonte: Adaptado de Sommerville (2007, p. 62-63).

### 3.3 Planejamento de projeto

A eficiência da gerência de um projeto de software depende de um planejamento minucioso do progresso do projeto. Os gerentes devem prever os problemas que podem ocorrer e elaborar soluções. Um plano elaborado no início de um projeto deve ser usado como guia.

Esse plano inicial deve ser o melhor possível em face das informações disponíveis, devendo evoluir à medida que o projeto progride e melhores informações se tornem disponíveis. O Quadro 3.5 mostra um pseudocódigo que estabelece um processo de planejamento de projeto para o desenvolvimento de software. Mostra que tal planejamento é um processo iterativo, que se encerra quando o projeto é concluído, visto que, à medida que as informações se tornam disponíveis durante o projeto, o plano deve ser regularmente revisado.

As metas da empresa constituem um importante fator que deve ser considerado na formulação do plano do projeto. À medida que essas metas mudam, as metas do projeto também mudam e, portanto, são necessárias mudanças no plano do projeto.

**Quadro 3.5 Pseudocódigo do processo de planejamento**

```

Defina as restrições do projeto.
Faça a avaliação inicial dos parâmetros do projeto.
Defina os marcos do projeto e os produtos a serem entregues.
While projeto não foi concluído ou cancelado loop
    Elabore um cronograma do projeto
    Inicie as atividades de acordo com o cronograma
    Aguarde (por um período)
    Examine o progresso do projeto
    Revise as estimativas de parâmetros do projeto
    Atualize o cronograma do projeto
    Renegocie as restrições do projeto e os produtos a serem entregues
If surgirem problemas, then
    Inicie revisão técnica e possível nova revisão
endif
end loop

```

Fonte: Adaptado de Sommerville (2007, p. 54).

Analizando o pseudocódigo, no início devem-se avaliar as restrições — data de entrega definida, pessoal disponível, orçamento geral etc. — que afetam o projeto. Com base nessas informações devemos estimar os parâmetros do projeto, tal como sua estrutura, tamanho e distribuição das funções. Logo após devemos definir os marcos de progresso e os produtos a serem entregues. Assim sendo, após as definições iniciais, o processo, então, entra em um *loop*, onde deve ser elaborado um cronograma estimado para o projeto e as atividades definidas no cronograma são iniciadas ou liberadas para prosseguimento. Após algum tempo necessário para que as tarefas sejam executadas, deve-se examinar o progresso e anotar as discrepâncias em relação ao cronograma. Verificam-se as estimativas iniciais dos parâmetros do projeto por serem experimentais, sempre será necessário modificar o plano original. À medida que as informações se tornam disponíveis, você deve revisar as hipóteses originais e o cronograma do projeto. Se o projeto estiver atrasado, pode ser necessário renegociar as restrições e os produtos a serem entregues com o cliente, mas, se essa renegociação não for bem-sucedida e o cronograma não puder ser cumprido, devemos considerar a necessidade de uma revisão técnica, que deverá ter como objetivo encontrar uma abordagem alternativa que se limite às restrições do projeto e cumpra o cronograma.

### 3.3.1 O plano de projeto

Estabelece os recursos disponíveis para o projeto bem como a sua estrutura analítica e um cronograma para realizar o trabalho. Em alguns casos, o plano de projeto está relacionado apenas ao processo de desenvolvimento. Os detalhes do plano de projeto variam dependendo do tipo do projeto e da empresa. Entretanto, a maioria dos planos deve incluir as seguintes seções:

| Seções   | Descrição   |
|--|---|
| <b>1. Introdução</b>   | Descreve brevemente os objetivos do projeto e estabelece as restrições (por exemplo, orçamento, prazo etc.) que afetam o gerenciamento do projeto.  |
| <b>2. Organização do projeto</b>                               | Descreve o modo como a equipe de desenvolvimento está organizada, as pessoas envolvidas e seus papéis na equipe.  |
| <b>3. Análise de riscos</b>                                    | Descreve os possíveis riscos do projeto, a probabilidade da ocorrência desses riscos e as propostas de estratégias de redução de riscos.  |
| <b>4. Requisitos de recursos de hardware e de software</b>     | Especificam o hardware e o software de apoio necessários para realizar o desenvolvimento. Se o hardware precisar ser comprado, podem ser incluídas as estimativas de preços e o prazo de entrega. |
| <b>5. Estrutura analítica</b>                                  | Estabelece a estrutura analítica do projeto em atividades e identifica os marcos e os produtos a serem entregues com cada atividade.  |
| <b>6. Cronograma do projeto</b>                                | Apresenta as dependências entre as atividades, o prazo estimado necessário para atingir cada marco e a alocação de pessoas nas atividades.  |
| <b>7. Mecanismos de monitoração e elaboração de relatórios</b> | Definem os relatórios de gerenciamento que devem ser produzidos, quando eles devem ser produzidos e os mecanismos de monitoração de projeto usados.   |

Durante o projeto, devemos revisar o plano constantemente, pois algumas panes como o cronograma sofrem mudanças constantes, e para simplificar as revisões você deve organizar o documento em seções separadas que podem ser individualmente substituídas à medida que o plano evolui.

### 3.3.2 Marcos e produtos a serem entregues

Os gerentes precisam de informações para realizar seu trabalho. Como o software é intangível, essas informações podem ser fornecidas apenas como relatórios e documentos que descrevem o estado do software que é desenvolvido. Sem elas, é impossível avaliar quão bem o trabalho está progredindo e as estimativas de custos e o cronograma não podem ser atualizados.

Ao planejar um projeto, você deve estabelecer uma série de **marcos**. A cada marco, deve existir uma saída formal, como um relatório, que possa ser apresentado à gerência. Esses relatórios de marcos não precisam ser documentos extensos, podendo ser simples e breves sobre a atividade concluída.

Um produto é um resultado de projeto entregue ao cliente, sendo em geral disponibilizado no fim de alguma fase importante, como especificação ou design. Os produtos são geralmente marcos, porém os marcos não precisam ser necessariamente um produto, pois podem ser resultados internos de projeto usados pelo gerente para verificar o progresso do projeto, embora não sejam entregues ao cliente.



### *Para saber mais*

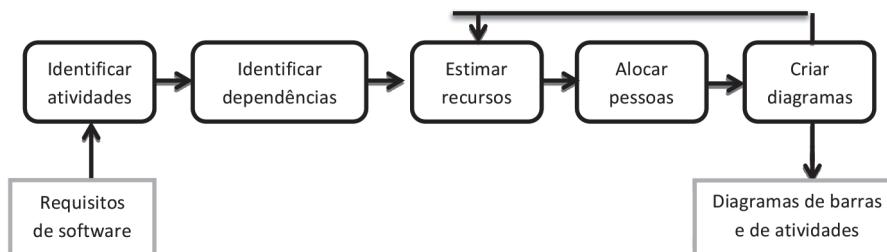
Um marco é um ponto final reconhecível de uma atividade do processo de software.

#### **3.3.2.1 Cronograma do projeto**

O desenvolvimento do cronograma do projeto é um dos trabalhos mais difíceis para um gerente de projeto, pois estima o tempo e os recursos necessários para concluir atividades, organizando-as em uma sequência coerente. A estimativa de cronograma é mais complicada pelo fato de que projetos diferentes podem usar métodos e linguagens de implementação diferentes.

Os cronogramas, portanto, devem ser continuamente atualizados à medida que mais informações sobre o progresso se tornem disponíveis. O desenvolvimento do cronograma de projeto (Figura 3.6) envolve a divisão do trabalho total de um projeto em atividades separadas e a avaliação do tempo necessário para completar essas atividades.

**Figura 3.6 Processo de desenvolvimento de cronograma de projeto**



Fonte: Adaptada de Sommerville (2007, p. 66).



### Para saber mais

Os diagramas de barras e as redes de atividades são notações gráficas usadas para ilustrar o cronograma do projeto. Os diagramas de barras mostram quem é responsável por cada atividade e quando as atividades estão programadas para serem iniciadas e terminadas. As redes de atividades mostram as dependências entre as diferentes atividades que constituem um projeto.

Os diagramas de barras e os diagramas de atividades podem ser gerados automaticamente baseados em um banco de dados de informações do projeto, usando uma ferramenta de gerenciamento de projetos.

Em geral, algumas das atividades são realizadas simultaneamente, devendo o gerente coordenar as atividades paralelas e organizar o trabalho de modo que a força de trabalho seja usada de maneira otimizada. É importante evitar uma situação em que todo o projeto fique atrasado pelo fato de a tarefa crítica não estar concluída.

As atividades de projeto devem durar pelo menos uma semana; ao estimar os cronogramas, não devemos considerar que todo o estágio do projeto estará livre de problemas. As pessoas que trabalham em um projeto podem ficar doentes ou podem sair, o hardware pode apresentar defeito e o software ou o hardware de apoio essencial podem ser entregues com atraso.

Uma boa regra prática é fazer a estimativa como se nada fosse dar errado e, então, aumentá-la para cobrir problemas não previstos. Um fator de contingência adicional para cobrir problemas não previstos pode também ser acrescentado à estimativa. Esse fator extra de contingência depende do tipo de projeto, dos parâmetros (prazo final, padrões etc.) e da qualidade e experiência dos engenheiros de software que trabalham no projeto. Acrescento 40% à minha estimativa original para problemas não previstos e mais 25% para cobrir aspectos não imaginados.

Geralmente os cronogramas de projeto são representados como um conjunto de diagramas que apresentam a estrutura analítica do projeto, as dependências de atividades e as alocações de pessoal.



### Para saber mais

No intuito de proporcionar um maior entendimento sobre o assunto, acesse o link: <<http://www.ebah.com.br/content/ABAAAGMuQAG/sommerville-engenharia-software-8-edicao#>> e faça a leitura das p. 66-68.

### 3.3.3 Gerenciamento de riscos

Uma das principais atividades dos gerentes de projeto é o gerenciamento de riscos, uma vez que consiste em prever os riscos que podem afetar tanto a organização que desenvolve os softwares como o cronograma do projeto ou a qualidade do produto que está sendo desenvolvido, e tomar providências para que esses riscos sejam evitados.

Os resultados da análise de riscos devem ser registrados no plano de projeto junto com uma análise das consequências de uma ocorrência desse risco. Com um gerenciamento eficiente de riscos, torna-se mais fácil lidar com os problemas e assegurar que eles não conduzam a um orçamento inaceitável ou atraso no cronograma.

Simplificando, você pode pensar em risco como algo que seria preferível não ocorrer, pois pode ameaçar o projeto, o software que está sendo desenvolvido ou a organização. Há, portanto, três categorias de risco relacionadas:

- Riscos de projeto:** são aqueles que afetam o cronograma ou os recursos de projeto. Por exemplo, perda de um programador experiente.
- Riscos de produto:** são aqueles que afetam a qualidade ou o desempenho do software que está sendo desenvolvido. Por exemplo, um framework que não funciona conforme esperado.
- Riscos de negócio:** são aqueles que afetam a organização que desenvolve ou adquire o software. Por exemplo, um concorrente que lança um produto antes de você, um produto mais inovador que o seu.

Obviamente, esses tipos de risco se sobrepõem, uma vez que, se um programador experiente deixa um projeto, isso acarreta um risco de projeto, pois a entrega do sistema pode atrasar. Também pode ser um risco de produto, visto que um substituto pode não ter tanta experiência e, por isso, cometer erros de programação. E por fim pode também ser um risco de negócio, uma vez que a experiência do programador desligado não está disponível para ser oferecida em novos projetos.

Os riscos que podem afetar um projeto dependem do projeto e do ambiente organizacional onde o software está sendo desenvolvido. No entanto, muitos deles são universais. Sommerville aponta na Tabela 3.3 alguns dos riscos mais comuns em um projeto.

Tabela 3.3 Risco de software

| Risco                   | Tipo de risco   | Descrição   |
|-------------------------|-----------------|---|
| Rotatividade de pessoal | Projeto         | Pessoal experiente vai deixar o projeto antes do final.               |
| Mudança de gerência     | Projeto         | Haverá uma mudança na gerência com novas prioridades.                 |
| Mudança de requisitos   | Projeto/Produto | Haverá um número maior de mudanças de requisitos do que foi previsto. |

continua

continuação

|                          |                 |   |
|--------------------------|-----------------|---|
| Atrasos de especificação | Projeto/Produto | As especificações das interfaces essenciais não estarão disponíveis dentro do prazo.  |
| Tamanho subestimado      | Projeto/Produto | O tamanho do sistema foi subestimado.   |
| Mudança de tecnologia    | Negócios        | A tecnologia na qual o sistema foi construído foi ultrapassada por novas tecnologias. |
| Concorrência de produto  | Negócios        | Um produto concorrente foi lançado no mercado antes da conclusão do sistema.          |

Fonte: Adaptada de Sommerville (2007, p. 70).

As incertezas inerentes aos projetos fazem que o gerenciamento de riscos seja essencial para os projetos de software. As incertezas se originam de requisitos mal definidos, de dificuldades na estimativa de prazo e recursos necessários para o desenvolvimento de software, da dependência de habilidades individuais e de mudanças de requisitos por mudanças nas necessidades do cliente. Assim, é necessário prever os riscos, compreender seu impacto no projeto, no produto e nos negócios, e tomar providências para evitá-los. Dessa maneira, pode ser necessário elaborar planos de contingência, a fim de poder tomar providências imediatas para recuperação, caso os riscos ocorram.

O processo de gerenciamento de riscos envolve os seguintes estágios:

- └ identificação de riscos;
- └ análise de riscos;
- └ planejamento de riscos;
- └ monitoração de riscos.

O processo de gerenciamento de riscos, como todos os outros planejamentos de projeto, é um processo iterativo que prossegue ao longo do projeto. Uma vez elaborado um conjunto inicial de planos, a situação é monitorada. À medida que mais informações sobre os riscos se tornarem disponíveis, eles deverão ser analisados novamente e novas prioridades deverão ser estabelecidas. Os planos de prevenção de riscos e de contingência podem ser modificados à medida que novas informações de risco surgirem.

É primordial a documentação dos resultados do processo de gerenciamento de riscos em um plano de gerenciamento de riscos. Esse plano deve incluir uma explicação dos riscos enfrentados pelo projeto, uma análise desses riscos e os planos necessários para gerenciá-los.

### **3.3.3.1 Identificação de riscos**

A identificação de riscos é o primeiro estágio do gerenciamento de riscos, e está relacionada com a descoberta dos possíveis riscos do projeto. A princípio, os riscos não devem ser priorizados ou avaliados nesse momento, porém, na prática, riscos com consequências muito pequenas ou com pouca probabilidade de ocorrer não são considerados.

A identificação de riscos é uma atividade realizada como um processo em equipe, usando uma abordagem de *brainstorming*, ou ser baseada na experiência das pessoas da equipe. Para auxiliar o processo de identificação deve ser elaborado um *check-list* de diferentes tipos de risco como podemos ver a seguir:

1. *Riscos de tecnologia.* Derivam de tecnologias de software ou hardware usadas para desenvolver o sistema.
2. *Riscos de pessoal.* Associados às pessoas da equipe de desenvolvimento.
3. *Riscos organizacionais.* Derivam do ambiente organizacional no qual o software está sendo desenvolvido.
4. *Riscos de ferramentas.* Derivam de ferramentas CASE e outros softwares de apoio, usados para desenvolver o sistema.
5. *Riscos de requisitos.* Derivam de mudanças de requisitos de cliente e do processo de gerenciamento de mudança de requisitos.
6. *Riscos de estimativas.* Derivam de estimativas de gerenciamento das características de sistema e estimativas de recursos necessários para construir o sistema.

A Tabela 3.4 apresenta alguns exemplos dos possíveis riscos em cada uma dessas categorias. Após concluir o processo de identificação, você deverá ter uma longa lista de riscos que podem ocorrer e quais podem afetar o produto, o processo e os negócios.

**Tabela 3.4 Tipos de riscos x riscos**

| Tipo de risco         | Riscos possíveis  |
|-----------------------|---|
| <b>Tecnologia</b>     | Banco de dados não pode processar tantas operações por segundo como desejado.   |
| <b>Pessoal</b>        | É difícil recrutar pessoal com habilidades necessárias.<br>O pessoal qualificado não está disponível nos momentos críticos.<br>O treinamento necessário para o pessoal não está disponível. |
| <b>Organizacional</b> | A empresa é reestruturada, com mudança de gerência responsável pelo projeto.<br>Reduções no orçamento do projeto por problemas financeiros da organização.                                  |
| <b>Ferramentas</b>    | As ferramentas CASE não podem ser integradas.<br>O código gerado pelas ferramentas CASE é ineficiente.  |
| <b>Requisitos</b>     | Mudanças de requisitos que requerem retrabalho maior de projeto são propostas.<br>Cliente não entende os impactos das mudanças de requisitos.   |
| <b>Estimativas</b>    | O tamanho do software foi subestimado.<br>O prazo para desenvolver o software foi subestimado.<br>Taxa de reparo foi subestimada.   |

Fonte: Adaptada de Sommerville (2007, p. 71).

### 3.3.3.2 Análise de riscos

Na análise de riscos, deve-se considerar cada risco identificado e fazer uma avaliação de sua probabilidade e seriedade. Não há uma maneira fácil de fazer isso e para tanto devemos contar com a própria avaliação e experiência das pessoas que

fazem tal análise, fazendo que os gerentes de projetos experientes sejam as melhores pessoas para auxiliar no gerenciamento de riscos.

Essas estimativas de risco geralmente não precisam ser avaliações numéricas precisas, mas deverão basear-se em um número de faixas:

- └ A probabilidade do risco deve ser avaliada como muito baixa (< 10%), baixa (10-25%), média (25-50%), alta (50-75%) ou muito alta (> 75%).
- └ Os efeitos do risco podem ser avaliados como catastróficos, sérios, toleráveis ou insignificantes.

Assim, devemos computar os resultados desse processo de análise usando uma tabela ordenada de acordo com a gravidade do risco. A Tabela 3.5 ilustra isso. Obviamente, a avaliação da probabilidade e da seriedade é arbitrária nesse caso, porém, na prática, para fazer essa avaliação, é necessário possuir informações detalhadas sobre o projeto, o processo, a equipe de desenvolvimento e da organização

**Tabela 3.5 Análise de riscos**

| Risco  | Probabilidade | Efeitos         |
|--|---------------|-----------------|
| Problemas financeiros na empresa forçam a redução no projeto                 | Baixa         | Catastróficos   |
| Impossibilidade de recrutar pessoal habilitado para o projeto                | Alta          | Catastróficos   |
| Funcionário está doente nos momentos críticos do projeto                     | Média         | Sérios          |
| São propostas mudanças de requisitos que requerem maior retrabalho           | Média         | Sérios          |
| Tempo necessário para desenvolver o produto foi subestimado                  | Alta          | Serios          |
| Ferramentas CASE não podem ser integradas                                    | Alta          | Toleráveis      |
| Os clientes não compreendem o impacto das mudanças de requisitos             | Média         | Toleráveis      |
| O treinamento necessário para os funcionários não está disponível no momento | Média         | Toleráveis      |
| O código gerado pela feramenta CASE é ineficiente                            | Média         | Insignificantes |

Fonte: Adaptada de Sommerville (2007, p. 72).

Após os riscos terem sido analisados e classificados, é necessário avaliar quais são os mais significativos. A avaliação deve depender da combinação da probabilidade da ocorrência do risco e de seus efeitos. Geralmente os riscos catastróficos devem sempre ser considerados, assim como todos os riscos sérios que tenham probabilidade de ocorrência acima da média.

### 3.3.3.3 Planejamento de riscos

O processo de planejamento de riscos considera cada um dos riscos importantes identificados e define estratégias para gerenciá-los. Outra vez, não há um processo simples a ser seguido para definir os planos de gerenciamento de riscos. Ele conta com a avaliação e a experiência do gerente do projeto. A Tabela 3.6 mostra as possíveis estratégias identificadas para os riscos principais da Tabela 3.5.

Essas estratégias dividem-se em três categorias:

1. *Estratégias de prevenção*. Significa que a probabilidade de que o risco ocorra será reduzida.
2. *Estratégias de minimização*. Significa que o impacto do risco será reduzido.
3. *Planos de contingência*. Significa que você está preparado para o pior e tem uma estratégia para lidar com o problema.

**Tabela 3.6 Estratégias de gerenciamento de riscos**

| Risco  | Estratégia   |
|--|--|
| Problemas financeiros da empresa                       | Preparar um documento que mostre à gerência como o projeto está contribuindo para as metas da empresa.   |
| Problemas de recrutamento                              | Alertar o cliente das dificuldades potenciais e da possibilidade de atrasos. Estudar a compra de componentes de software.                                  |
| Doença do pessoal da equipe<br>Componentes defeituosos | Reorganizar a equipe de maneira que possa ser feita a superposição de trabalho, desta forma fazendo que as pessoas compreendam as tarefas umas das outras. |
| Mudanças de requisitos                                 | Efetuar a substituição dos componentes defeituosos por outros com confiabilidade reconhecida.  |
| Reestruturação da empresa                              | Preparar um documento que mostra a importância do projeto para cumprimento das metas organizacionais.  |
| Desempenho de banco de dados                           | Checar a possibilidade de adquirir um banco de dados com melhor desempenho.  |
| Prazo de desenvolvimento subestimado                   | Estudar a possibilidade de aquisição de novos componentes de software e também a de um gerador de programas (relatórios).                                  |

Fonte: Adaptada de Sommerville (2007, p. 73).

### 3.3.3.4 Monitoração de riscos

A monitoração de riscos envolve a avaliação de cada um dos riscos identificados para decidir se está se tornando ou não mais ou menos provável de acontecer e se os efeitos sofreram alguma mudança. A Tabela 3.7 (SOMMERVILLE, 2007) apresenta alguns fatores que podem ser úteis na avaliação dos tipos de riscos.

A monitoração deve ser um processo contínuo e, a cada revisão feita pela gerência, deve-se considerar e discutir cada um dos principais riscos isoladamente.

**Tabela 3.7 Fatores de riscos**

| <b>Tipo de risco</b> | <b>Indicadores potenciais</b>  |
|----------------------|--|
| Tecnologia           | Entrega de hardware ou software de apoio com atraso, vários problemas são relatados.   |
| Pessoal              | Baixo moral da equipe, relacionamentos precários entre os membros, disponibilidade de emprego.   |
| Organizacional       | Boatos na empresa e falta de ação da gerência sênior.  |
| Ferramentas          | Relutância das pessoas da equipe em utilizar ferramentas que melhorem a produtividade, reclamações sobre ferramentas CSASE, demandas por hardware mais potentes. |
| Requisitos           | Várias solicitações de mudanças são feitas, reclamações de clientes.   |
| Estimativas          | Falha no cumprimento do cronograma, falha em eliminar os defeitos relatados.   |

Fonte: Adaptada de Sommerville (2007, p. 73).



### *Questões para reflexão*

1. Qual a importância para o desenvolvedor de software do uso da engenharia de requisitos bem como da documentação desse processo?
2. Levando em conta o gerenciamento de projeto de software, em sua opinião **“por que o papel do engenheiro de software ainda é um enigma tanto para as empresas desenvolvedoras quanto para os próprios desenvolvedores?”**.



### *Atividades de aprendizagem*

1. Levando em conta os conceitos sobre a gerenciamento de riscos é INCORRETO afirmar:
  - I. Pode-se responder ao risco de cinco formas diferentes: evitando, transferindo, reduzindo, aceitando e ignorando.
  - II. As ameaças precisam ser definidas quanto ao grau de exposição que apresentam para o ativo em questão. Quando se define o grau da ameaça, deseja-se determinar quanto existe daquela ameaça, independentemente do ativo ao qual se está referindo para aquela ameaça.

- III. A avaliação do risco propriamente dita nada mais é do que comparar a estimativa de risco contra os critérios de risco para determinar os níveis de riscos de incidentes de segurança da informação. Normalmente, quanto maiores o impacto e a probabilidade, maior será o risco.
- Assinale a alternativa que corresponda às afirmações verdadeiras:
- I, II e III
  - II e III
  - I e III
  - I e II
  - Somente a I
2. Assinale a alternativa que corresponda corretamente aos estágios de gerenciamento de riscos.
- Projeto — produto — negócios.
  - Tecnologia — organizacionais — pessoas — estimativa — requisitos.
  - Elaboração da proposta — planejamento e desenvolvimento do cronograma do projeto — monitoração e revisões do projeto — elaboração de relatórios e apresentações.
  - Identificação de riscos — análise de riscos — planejamento de riscos — monitoração de riscos.
  - Estratégias de minimização — estratégias de prevenção — plano de contingência.

**Fique ligado!**



Nesta unidade, foram abordados os seguintes aspectos relativos aos requisitos de software, à engenharia de requisitos e ao gerenciamento de projetos relacionados aos objetivos de aprendizagem:

- └ Os requisitos de um sistema de software definem o que o sistema deve fazer e as restrições.
- └ Os requisitos funcionais são declarações dos serviços que o sistema deve fornecer ou são descrições de como algumas computações devem ser realizadas.
- └ Os requisitos de domínio são requisitos funcionais derivados das características do domínio da aplicação.

- └ Os requisitos não funcionais restringem o sistema que está sendo desenvolvido, bem como o processo de desenvolvimento que deve ser usado.
- └ Os requisitos não funcionais podem ser requisitos de produto, organizacionais ou externos e estão relacionados às propriedades emergentes do sistema. Portanto, aplicam-se ao sistema como um todo.
- └ Os requisitos de usuário destinam-se às pessoas envolvidas no uso e na aquisição do sistema. Para tal, devem ser redigidos com o uso de linguagem natural, tabelas e diagramas de fácil compreensão para seus leitores.
- └ Os requisitos de sistema destinam-se a comunicar, de maneira precisa, as funções que o sistema deve fornecer. Podem ser escritos em linguagem natural, de maneira estruturada, e complementados com tabelas e modelos de sistemas, sendo destinados aos desenvolvedores de sistema.
- └ O documento de requisitos de software é a declaração aprovada dos requisitos de sistema. Deve ser organizado de tal maneira que os clientes de sistema e os projetistas de software possam usá-lo.
- └ O processo de engenharia de requisitos inclui um estudo de viabilidade, elicitação e análise, especificação, validação e gerenciamento de requisitos.
- └ A elicitação e a análise de requisitos constituem um processo iterativo que pode ser representado como uma espiral de atividades tais como obtenção, classificação e organização, negociação e documentação de requisitos
- └ Diferentes stakeholders no sistema têm diferentes requisitos. E em sistemas complexos devem ser analisados com base em uma série de pontos de vista.
- └ Pontos de vista podem ser pessoas ou outros sistemas que interagem com o sistema que está sendo especificado, stakeholders afetados pelo sistema ou pontos de vista de domínio que restringem os requisitos.
- └ Fatores sociais e organizacionais têm forte influência nos requisitos do sistema e podem determinar se o sistema é realmente usado.
- └ Validação de requisitos é o processo para verificar os requisitos em relação à validade, consistência, completeza, realismo e facilidade de verificação.
- └ Revisões de requisitos e prototipação são as principais técnicas usadas para a validação de requisitos.
- └ Mudanças de negócios, organizacionais e técnicas levam, inevitavelmente, às mudanças dos requisitos de um sistema de software.
- └ O gerenciamento de requisitos é o processo para gerenciar e controlar essas mudanças.
- └ O processo de gerenciamento de requisitos inclui o planejamento de gerenciamento, no qual políticas e procedimentos de gerenciamento

de requisitos são projetados, e o gerenciamento de mudanças, no qual você deve analisar as mudanças de requisitos propostas e avaliar seu impacto.

- └ Um bom gerenciamento de projeto de software é essencial para que os projetos de engenharia do software sejam desenvolvidos dentro do cronograma e do orçamento.
- └ O gerenciamento de software é diferente do gerenciamento em outras áreas da engenharia.
- └ O software é intangível, assim os projetos podem ser novos ou inovadores, portanto, não existe um conjunto de experiências para guiar seu gerenciamento. Os processos de software não são bem compreendidos.
- └ Os gerentes de software possuem diversos papéis. As atividades mais significativas são planejamento, elaboração de estimativas e desenvolvimento de cronograma do projeto.
- └ O planejamento e a elaboração de estimativas são processos iterativos que continuam ao longo de um projeto e à medida que mais informações se tornam disponíveis os planos e os cronogramas devem ser revisados.
- └ Um marco de projeto é um resultado previsível de uma atividade, no qual algum relatório formal de progresso deve ser apresentado à gerência.
- └ Os marcos devem ocorrer regularmente ao longo de um projeto de software, sendo que um produto é um marco disponibilizado para o cliente do projeto.
- └ O desenvolvimento do cronograma do projeto envolve a criação de várias representações gráficas de partes do plano de projeto onde incluem diagramas de atividades, que mostram os inter-relacionamentos entre as atividades de projeto, e diagramas de barras, que mostram a duração das atividades.
- └ Os principais riscos do projeto devem ser identificados e avaliados para definir sua probabilidade e as consequências para o projeto. Sendo assim devem-se elaborar planos para evitar, gerenciar ou lidar com os prováveis riscos se ou quando eles ocorrerem.
- └ Os riscos devem ser explicitamente discutidos em cada reunião de progresso do projeto.

## Para concluir o estudo da unidade



Neste momento podemos refletir a importância de se fazer um bom trabalho de levantamento de requisitos, uma vez que, se não for efetuado corretamente, invariavelmente poderá levar o projeto a falhas que muitas vezes só serão perceptíveis em uma fase adiantada do desenvolvimento.

Sabe-se que buscar informações com os usuários é um processo árduo, pois retirar informações de um usuário que desconfia e/ou não acredita no processo é algo que exige grande dedicação do analista, visto que definir as funcionalidades que um sistema deverá possuir é muito fácil, porém, ter que definir as restrições do mesmo é algo um pouco mais difícil, visto que devemos lidar com medos, insegurança, conflitos, entre outros.

Torna-se essencial buscar nos processos de engenharia de requisitos os meios para realizar tal levantamento bem como um bom gerenciamento do projeto, gerenciando, dessa forma, os requisitos básicos de um bom projeto de software, custo, prazo e qualidade.



## Atividades de aprendizagem da unidade

1. O levantamento de requisitos é uma etapa fundamental do projeto de sistemas. Dependendo da situação encontrada, uma ou mais técnicas podem ser utilizadas para a elicitação dos requisitos. A respeito dessas técnicas, analise as afirmações a seguir.
  - a) Workshop de requisitos consiste na realização de reuniões estruturadas e delimitadas entre os analistas de requisitos do projeto e representantes do cliente.
  - b) Cenário consiste na observação das ações do funcionário na realização de uma determinada tarefa, para verificar os passos necessários para sua conclusão.
  - c) As entrevistas são realizadas com os stakeholders e podem ser abertas ou fechadas.
  - d) A prototipagem é uma versão inicial do sistema, baseada em requisitos levantados em outros sistemas da organização.
  - e) As alternativas A e E estão corretas.

2. Levando em consideração gerenciamento de riscos, significa que devemos identificar riscos e traçar planos para minimizar seus efeitos sobre o projeto. Assinale a opção que contenha a descrição de um exemplo de risco de negócios.
- O tamanho do sistema foi subestimado.
  - A tecnologia sobre a qual o sistema está sendo construído foi superada por nova tecnologia.
  - As especificações de interface não estavam disponíveis dentro do prazo.
  - Haverá uma mudança no gerenciamento organizacional, com a definição de prioridades diferentes.
  - Haverá maior número de mudanças nos requisitos do que o previsto.
3. Com relação às políticas de rastreabilidade de requisitos, elas são definidas e decididas durante o estágio de:
- Agregação dos requisitos funcionais, apenas.
  - Implementação do sistema, apenas.
  - Gerenciamento de requisitos.
  - Implantação do sistema.
  - Eliminação dos requisitos não funcionais.
4. Assinale a alternativa que corresponda aos principais processos relativos à gerência de risco do projeto:
- O gerenciamento de recursos dos riscos, que tem como objetivo principal garantir o contínuo fornecimento de recursos financeiros para dar continuidade ao projeto.
  - A identificação dos riscos, que tem como foco definir as melhorias necessárias para o aproveitamento de oportunidades e respostas às ameaças.
  - A monitoração dos riscos, cuja finalidade é analisar os resultados específicos do projeto para determinar se eles estão de acordo com os padrões de qualidade relevantes e identificar as formas para eliminar as causas de desempenhos insatisfatórios.
  - O controle das respostas aos riscos, cuja tarefa é responder às mudanças nos riscos no decorrer do projeto.
  - A avaliação periódica do desempenho dos riscos, que tem como finalidade definir e avaliar o desempenho geral do projeto buscando assegurar a satisfação dos padrões relevantes de qualidade.

5. Considerando os componentes envolvidos em um projeto de desenvolvimento de software, considere:
- I. Pessoa ou grupo que fornece os recursos financeiros para o projeto.
  - II. Pessoas que estejam ativamente envolvidas no gerenciamento ou na execução do projeto.
  - III. Pessoas e organizações cujos interesses possam ser afetados de forma positiva pelo projeto.
  - IV. Pessoas e organizações cujos interesses possam ser afetados de forma negativa pelo projeto.

Assinale a alternativa que corresponda a um stakeholder:

- a) II, III e IV, apenas.
- b) II e III, apenas.
- c) II, apenas.
- d) I, apenas.
- e) I, II, III e IV.

---

## Referências

PRESSMANN, Roger S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, Ian. **Engenharia de software**. 8. ed. São Paulo: Pearson, 2007.

# Noções sobre modelagem de sistemas

*Polyanna Pacheco Gomes Fabris*

**Objetivos de aprendizagem:** Caro aluno, você será levado a estudar e conhecer conceitos importantes da modelagem de sistemas, abordaremos assuntos referentes à modelagem estruturada e à modelagem orientada a objeto (OO), neste mundo da orientação a objetos trabalharemos conceitos bem relevantes para seu aprendizado e a prática da OO. Vamos abordar a linguagem de modelagem unificada — UML, bem como uma visão geral dos seus diagramas.

## » Seção 1: **Introdução à modelagem estruturada de sistemas**

Caro aluno, nesta seção vamos abordar o conteúdo sobre modelagem estruturada. Conheceremos sua história, alguns diagramas como: diagrama entidade relacionamento (DER) e diagrama de fluxo de dados (DFD). E ao final da seção você terá algumas atividades para reforçar seu aprendizado.

Tenha uma boa leitura!

## **— Seção 2: Introdução à modelagem orientada a objeto e para web**

Caro aluno, nesta seção você conhecerá um pouco da história da modelagem orientada a objeto, da linguagem de programação orientada a objeto e dentro do paradigma da orientação a objetos vamos abordar alguns conceitos e características relevantes para seu aprendizado e prática. E para reforçar o conteúdo estudado você terá alguns estudos de caso e atividades de aprendizagem.

Tenha uma boa leitura!

---

# Introdução ao estudo

Caro aluno, vamos iniciar nosso aprendizado pela modelagem de dados.

Podemos dizer que modelos são abstrações que mostram a essência de um problema, e a modelagem de dados é um processo para a criação de um software, que deve ser organizado, utilizando alguma técnica predefinida, sendo que as etapas desse processo compõem o ciclo de vida do software. E é na etapa de análise de sistemas que precisamos definir qual modelagem será utilizada.

Mas afinal, o que é análise na informática? Análise é quando estudamos uma situação problema. Pense num cenário em que um cliente apresenta suas necessidades ou desejos para resolução de um problema de um determinado segmento de sua empresa. Com base nessas informações são necessários estudos, ou seja, análises para identificação do real problema e como ele pode ser solucionado.

Os modelos são o resultado desses estudos e consistem em uma proposta do sistema a ser implementado. Eles são criados pelo analista para auxiliar no entendimento tanto do cliente que solicitou e receberá o sistema, quanto da equipe que o implementará.

Para concluir esta introdução não poderíamos deixar de citar Confúcio, filósofo chinês, que dizia: “uma imagem vale mais que mil palavras”. E essa é a intenção ao representar por meio de imagens (modelos) nosso entendimento sobre a necessidade do cliente.

---

## Seção 1 Introdução à modelagem estruturada de sistemas

### 1.1 Introdução

Dentro de um processo de desenvolvimento de software conhecemos todas as fases para a construção de um software e uma delas é a fase de análise, fase essa de extrema importância para obtermos um software de qualidade, ou seja, um software que atende principalmente as necessidades do cliente. E como apoio à análise realizamos a modelagem do nosso sistema e nesta seção vamos abordar a modelagem estruturada.

### 1.2 Introdução à modelagem estruturada de sistemas

Segundo Bezerra (2007), para o paradigma da modelagem estruturada seus elementos são dados e processos, nos quais os processos agem sobre os dados para que um objetivo seja alcançado.

Essa modelagem teve seu início na década de 1970, com uma abordagem sistemática, utilizando técnicas de construção de modelos, tendo a preocupação naquilo “que o sistema tem de fazer”.

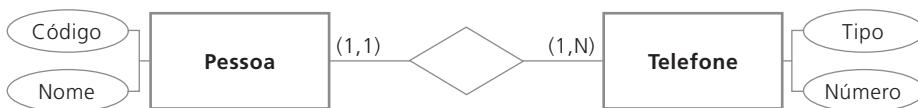
Já na década de 1980, ela foi revista e teve um aprimoramento voltado para se fazer uma análise para satisfazer os requisitos do usuário, colocando uma visão mais integrada das funções e dos dados do sistema.

Algumas fases da análise estruturada são a coleta de requisitos de sistemas, identificação dos atores e eventos, criação de diagramas como o de fluxo de dados (DFD) e diagrama de entidade-relacionamento (DER).

Antes de falarmos do DER, é importante abordarmos o MER (Modelo de Entidade e Relacionamento), que é um modelo conceitual de alto nível para apresentar as regras de negócio e não a implementação, ou seja, as informações obtidas no levantamento de requisitos para representar “o que” devemos fazer e não “como”, este, o “como fazer”, consiste na implementação do banco de dados. E a representação gráfica do MER é o DER.

O DER tem como objetivo representar as entidades e seus relacionamentos. É uma ferramenta de modelagem que define informações do modelo entidade-relacionamento para o banco de dados relacional, diferente do diagrama de classes que modela o mundo real. As entidades consistem nos dados concretos ou abstratos relevantes da organização para qual será destinada o sistema. Seguindo o modelo proposto por Peter Chen (2007), o DER é composto de entidades (representadas por um retângulo) e seus relacionamentos (representados por um losango), atributos (representados por elipses) e as linhas de conexão indicando a cardinalidade.

**Figura 4.1 Representação de um DER**



Fonte: Da autora (2014).

A cardinalidade é a quantidade de ocorrências de entidades que podem estar associadas (1:1, 1:N, N:N), representando a regra de negócio obtida no levantamento de requisitos.



### Para saber mais

Cardinalidades utilizadas no DER propostas por Peter Chen (2007) são:

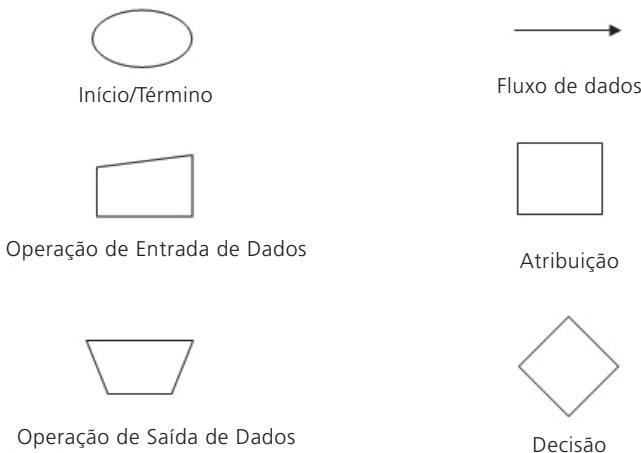
- ❑ 1..1 (lê-se um para um) — indica que as tabelas têm relacionamento único entre si. Você escolhe qual tabela receberá a chave estrangeira;
- ❑ 1..n (lê-se um para muitos) — a chave primária da tabela que tem o lado 1 vai para a tabela do lado N. No lado N ela é chamada de chave estrangeira;

— n..n (lê-se muitos para muitos) — quando tabelas têm entre si relacionamento n..n, é necessário criar uma nova tabela com as chaves primárias das tabelas envolvidas, ficando assim uma chave composta, ou seja, formada por diversos campos-chave de outras tabelas. O relacionamento então se reduz para um relacionamento 1..n, sendo que o lado n ficará com a nova tabela criada.

Já o DFD tem como objetivo representar graficamente o fluxo dos dados sendo uma etapa usada para criar uma visão geral do sistema, na qual podemos ver quais tipos de informações estão no sistema e onde os dados serão armazenados. Esse diagrama também pode ser chamado de diagrama de bolhas ou diagrama de fluxo de trabalho. Existem algumas regras para se criar um DFD: como todo o processo deve ter um fluxo de entrada e de saída, e todo o fluxo de dado deve ter uma origem e um destino, o objeto deve ter nome e a duplicação dos símbolos deve ser evitada. O DFD deve ter níveis de detalhamento

O DFD é uma boa ferramenta de comunicação e é utilizado como auxílio ao projeto.

**Figura 4.2 Símbolos usados no DFD**



Fonte: Da autora (2014).

A qualidade de uma modelagem estruturada é ter uma representação gráfica de fácil entendimento, ser particionada em uma rede de miniespecificações e interação permanente com o utilizador. Como problemas podem ser citados a manutenção, o custo elevado do sistema e as mudanças nele, as quais podem demandar tempo e recursos humanos.

A modelagem estruturada se utiliza de construção de modelos e faz uma notação que é própria da análise estruturada, com a finalidade de retratar o fluxo e o conteúdo das informações utilizadas pelo sistema, dividir o sistema em partições funcionais e comportamentais e descrever a essência do que será construído.



## *Atividades de aprendizagem*

1. Assinale V para verdadeiro e F para falso.

  - a) ( ) O DER identifica as entidades e seus relacionamentos.
  - b) ( ) O DER é um diagrama de eventos.
  - c) ( ) o DFD representa o fluxo de dados.
  - d) ( ) O DFD e o DER são diagramas da modelagem estruturada.
2. A modelagem estruturada teve início na década de 1970. Comente sobre os principais diagramas dessa modelagem.

**Seção 2**

# Introdução à modelagem orientada a objeto e para web

## 2.1 Introdução

Na seção 1, abordamos a importância da modelagem estruturada como apoio a análise de sistema para construção de um software com qualidade e nesta seção vamos continuar abordando a modelagem mais com enfoque na orientação a objetos.

## 2.2 Introdução à modelagem orientada a objeto e para web

A modelagem orientada a objeto (MOO) teve seu início na década de 1960, mas devido à limitação de hardware, ela ganhou força somente nas últimas duas décadas. A orientação a objeto (OO) possibilita uma melhor organização e reutilização de códigos-fonte, tornando mais fácil as atualizações e melhorias nos sistemas.

É interessante ressaltar que as linguagens de programação orientada a objeto vieram antes da modelagem orientada a objeto, ou seja, a modelagem se adaptou à programação.

Acompanhe a seguir a evolução histórica das Linguagens Orientadas a Objeto:

- └ 1966 — SIMULA (Kristen Nygaard, Noruega);
- └ 1980 — SMALLTALK (Xerox);
- └ 1983 — linguagem C with Classes passa a ser C++;
- └ 1986 — C++ (AT&T), SMALLTALK V , OBJECTIVE-C;
- └ 1988 — EIFFEL (Meyer, França);
- └ 1989 — Turbo Pascal 5.5 (Borland);
- └ 1995 — JAVA;
- └ 2001 — C#;
- └ 2002 — VB.NET;
- └ 2014 — ...

A linguagem que iniciou o conceito de classes foi a Simula 66, criada entre 1962 e 1968 por Kristen Nygaard e Ole-Johan Dahl. Já a linguagem Smalltalk foi a primeira linguagem de programação que possuía suporte para a orientação a objeto. Mas a linguagem que popularizou a orientação a objeto foi a C#, inicialmente conhecida como C com classes.

Em 1991, a empresa Sun MicroSystem criou o *Green Project*, considerado o berço do Java. A ideia que nasceu desse projeto, que era a interação entre o equipamento e usuário, evoluiu e com o “estouro” da internet e em janeiro de 1995 ele foi batizado de Java. As principais características do Java são: a orientação a objetos, sua portabilidade, possui uma grande biblioteca de rotinas e a segurança para executar progra-

mas via rede com restrição de execução. O Java não tem suporte para o conceito de herança múltipla, podendo ocorrer falhas ao chamar este método; uma possibilidade de se usar herança múltipla no Java é através de interfaces.

O C# ou C Sharp teve início em 2001; ele é uma linguagem visual dirigida por eventos e orientada a objetos que foi criada pela Microsoft como parte da plataforma .NET. Essa linguagem teve como base o C++, mas também possui influências de linguagens como object pascal e Java. Uma curiosidade sobre o símbolo '#', usado no nome C#, é que ele se refere a um símbolo musical denominado "sustenido", na música, ele significa que aumenta em meio tom uma nota musical. As principais características do C# são: ela é uma linguagem simples, totalmente orientada a objetos, fortemente tipada e flexível.

A modelagem orientada a objeto surgiu por causa da incompatibilidade da modelagem estruturada com a programação orientada a objeto. Alguns exemplos de modelagem orientada a objeto são Coad (1991), OMT (1991), OOSE (1992), Fusão (1995), UML (1996). Vamos comentar mais sobre a modelagem UML ainda nesta seção.

Outros motivos que influenciaram na criação da modelagem orientada a objeto foram os avanços na tecnologia de arquiteturas de computadores, que suportam sofisticados ambientes de programação e interfaces homem-máquina; e as linguagens de programação com recursos como modularização e ocultamento de informação. Alguns autores utilizam o termo "crise do software" para retratar os problemas associados ao modo como o software é criado e como é feita sua manutenção.

Uma vantagem de se usar a tecnologia de orientação é a facilidade de reutilização de código. Os modelos refletem o mundo real de um modo mais aproximado, descrevendo de maneira mais precisa os dados. Pequenas mudanças nos requisitos não implicam grandes alterações no sistema em desenvolvimento.

Neste momento você pode estar se perguntando: mas o que é a orientação a objeto? Bom, orientação a objeto é um paradigma para o desenvolvimento de sistemas, baseado na utilização de objetos que colaboram para construir sistemas mais complexos, essa colaboração entre os objetos é realizada por meio do envio de mensagens.



### Para saber mais

Paradigma é um conjunto de regras que delimitam fronteiras e descrevem como resolver problemas dentro dessa fronteira, o paradigma nos ajuda a organizar e coordenar a maneira como olhamos o mundo.

#### 2.2.1 Etapas da modelagem orientada a objeto

A análise orientada a objetos é um processo de construção de modelos em que se identifica e especifica um conjunto de objetos que interagem e comportam-se conforme os requisitos estabelecidos para o sistema.

Segundo Rumbaugh (1994), um modelo completo de sistema é composto por submodelos que expressam visões diferentes da mesma realidade, sendo elas divididas em:

- └ **Visão de objetos:** descreve estaticamente os objetos que compõem o sistema e seus relacionamentos por meio de diagramas de objetos;
- └ **Visão dinâmica:** descreve os aspectos do sistema que se modificam com o passar do tempo;
- └ **Visão funcional:** descreve as transformações dos valores dos dados de um sistema.

A modelagem visual é um jeito de pensar sobre problemas, verificando ideias do mundo real. As vantagens desse modelo são que facilitam a compreensão do problema e a comunicação entre técnicos e usuários além de montar a documentação do sistema.

Já um projeto orientado a objetos é o processo em que existe uma especificação detalhada do software a ser desenvolvido, de modo que essa especificação possa levar à direta implementação no ambiente-alvo.

E a programação orientada a objetos é um modelo de programação que se baseia em conceitos como classes, objetos, herança e polimorfismo. Tem como objetivo a resolução de problemas baseada na identificação de objetos e o processamento requerido por esses objetos, e então na criação de simulações desses objetos. Temos a programação por meio da especificação de classes e criação de hierarquias, sendo que propriedades comuns são transmitidas das superclasses para as subclasses pela herança.



### Questões para reflexão

Você sabia que temos uma distinção entre linguagens baseadas em objetos e linguagens orientadas a objetos? Vamos entender um pouco como essa distinção se dá.

Uma linguagem é baseada em objetos quando ela fornece apoio somente ao conceito de objetos. Temos como exemplo Ada e Visual Basic. Já uma linguagem orientada a objetos é aquela que fornece apoio a objetos, solicita que objetos sejam instâncias de classes e oferecem um mecanismo de herança, como exemplo o C++, Java e Smalltalk. Além dessa classificação também temos as linguagens híbridas e linguagens puras. Você pode verificar mais detalhes sobre cada uma delas nos livros de linguagem de programação.



### Para saber mais

Acesse os links a seguir para encontrar mais detalhes do histórico das linguagens de programação:

- └ <[http://pt.wikipedia.org/wiki/Java\\_\(linguagem\\_de\\_programa%C3%A7%C3%A3o\)](http://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o))>.
- └ <<http://pt.wikipedia.org/wiki/C%E2%99%AF>>.
- └ <[http://www.oficinadanet.com.br/artigo/526/c\\_sharp\\_csharp\\_o\\_que\\_e Esta\\_linguagem](http://www.oficinadanet.com.br/artigo/526/c_sharp_csharp_o_que_e Esta_linguagem)>.
- └ <<https://www.portaleducacao.com.br/informatica/artigos/6137/historia-e-caracteristicas-da-linguagem-c>>.
- └ <<http://www.infoescola.com/informatica/historia-do-java>>.
- └ <<http://olhardigital.uol.com.br/noticia/linguagem-javascript-um-pouco-de-historia/35728>>.

Alguns dos conceitos e características utilizados em orientação a objeto são: objetos, atributos, estados, polimorfismo, herança, encapsulamento, interações, classes, confiabilidade, reusabilidade, manutenibilidade, extensibilidade e outros que serão detalhados ainda nesta unidade.

- └ A reusabilidade é uma das principais características da orientação a objeto, sendo que ela faz a reutilização dos componentes do sistema. Com ela diminuímos o custo do projeto reutilizando componentes e objetos fazendo com que o desenvolvimento do software seja mais rápido sem diminuir a qualidade.
- └ A manutenibilidade visa à facilidade em fazer manutenção ou realizar alguma customização.
- └ A confiabilidade vem do encapsulamento, pois permite um maior controle e segurança às classes dos objetos.
- └ Extensibilidade é a medida da facilidade em se adicionar novas funcionalidades (operações) a um componente de uma modelagem existente. Alguns autores dizem que a extensibilidade é uma “vantagem” obtida com a herança e o polimorfismo.

Segundo Rumbaugh (1994), a orientação a objeto trata-se de uma nova maneira de pensar os problemas utilizando modelos organizados a partir de conceitos do mundo real, sendo o principal componente o objeto, que combina dados e comportamento.

A orientação a objeto tem bases conceituais e origem no campo de estudo da cognição, influenciando nas áreas da inteligência artificial e linguística.

Uma linguagem visual utilizada para modelar sistemas orientados a objetos é a UML (do inglês *Unified Modeling Language* ou Linguagem de Modelagem Unificada). A UML tem como objetivo descrever um sistema usando diagramas orientados a objetos.

A UML foi desenvolvida por Grady Booch, James Rumbaugh e Ivar Jacobson, sendo que eles possuem um grande conhecimento na área de modelagem orientada

a objeto, desenvolveram a UML, unindo seus conhecimentos e acrescentando novos conceitos e visões de linguagens.



### Para saber mais

Que Grady Booch, James Rumbaugh e Ivar Jacobson eram conhecidos como “os três amigos”, devido a união do que havia de melhor nas três metodologias orientadas a objetos mais conceituada (Booch, OMT e OOSE).

A UML possui vários diagramas, sendo os diagramas estruturais (classe, componentes, pacotes, estrutura composta), diagramas comportamentais (caso de uso, estados, atividade) e diagrama de interação (sequência, interação, tempo, colaboração). A seguir vamos descrever alguns destes diagramas:

- ❑ **Diagrama de Casos de Usos:** é um documento que descreve a sequência de eventos de um ator (que pode ser um humano ou entidade que interage com o sistema) para completar um processo.
- ❑ **Diagrama de Classe:** representa as relações entre as classes. Será detalhado no item 1.2.11 desta seção.
- ❑ **Diagrama de Sequência:** representa a sequência de processos (mensagens) em um programa de computador.
- ❑ **Diagrama de Colaboração:** mostra a interação de objetos e seus relacionamentos, mostrando as mensagens que podem ser trocadas entre eles. Os diagramas de sequência e de colaboração são isomórficos.
- ❑ **Diagrama de Estados:** pode ser denominado diagrama de transição de estados. Representa o estado em que um objeto pode se encontrar no decorrer da execução de processos de um sistema.
- ❑ **Diagrama de Atividades:** representa os fluxos conduzidos por processamentos.
- ❑ **Diagrama de Componentes:** mostra como as classes deverão se encontrar organizadas por meio da noção de componentes de trabalho.
- ❑ **Diagrama de Implantação:** pode ser denominado diagrama de instalação. Mostra componentes de hardware e software e sua interação com outros elementos.
- ❑ **Diagrama de Estrutura Composta:** descreve o relacionamento entre os elementos.
- ❑ **Diagrama de Tempo:** pode ser denominado diagrama temporal. Mostra o comportamento dos objetos e sua interação em uma escala de tempo.
- ❑ **Diagrama de Pacotes:** mostra os pacotes (representa um grupo de classes) divididos em agrupamentos lógicos mostrando as dependências entre eles.



### *Para saber mais*

A UML foi uma tentativa de Jim Rumbaugh e Grady Booch de combinar dois métodos de modelagem orientada a objeto: Booch e OMT. E após um tempo Ivar Jacobson se juntou a eles e criaram a primeira versão da UML.



### *Atividades de aprendizagem*

Vamos verificar como está seu conhecimento até agora?

1. É correto sobre a orientação a objeto:
  - I. Orientação a objeto é um paradigma para o desenvolvimento de sistemas, baseado na utilização de objetos que colaboram para construir sistemas mais complexos.
  - II. Todo o processo deve ter um fluxo de entrada e de saída.
  - III. Tem como característica o polimorfismo e herança.
  - IV. Tem como característica a criação de diagramas como o de fluxo de dados (DFD) e diagrama de entidade relacionamento (DER).
  - a) Itens I e II e IV são verdadeiros.
  - b) Itens I e IV são verdadeiros.
  - c) Somente o item II é falso.
  - d) Somente o III é verdadeiro.
  - e) Itens I, III e IV são verdadeiros.
2. Na questão acima qual(is) o(s) item(ns) que não se refere(m) a modelagem orientada a objeto, e por quê?

## 2.2.2 Objetos

Objetos são itens do mundo real, mas na computação consiste na ocorrência da classe. Todo objeto possui identidade própria e um estado, o qual pode ser alterado quando recebe uma mensagem ou evento.

Um objeto possui um estado, normalmente implementado por meio de seus atributos. Uma identidade única, ou seja, a propriedade de um objeto distingue-se de outros objetos. Essa identidade não é o nome do objeto, nem o endereço de memória onde ele está armazenado, mas sim um conceito de linguagens de programação que não é visível para os “usuários”. Trata-se de um comportamento que define como

ele reage às requisições de outros objetos, em termos de mudanças de estados e passagem de mensagens.

A mensagem ocorre quando se realiza uma chamada a um objeto, invocando um dos seus métodos e ativando um comportamento de sua classe.

Os objetos são responsáveis por atuar sobre os seus atributos e também sobre outros objetos, para isso desempenham diversas “operações”.

Podemos dizer também que os objetos são ***instâncias*** de uma classe [ver item 2.2.5] e a instância é cada ocorrência de um objeto criado na memória a partir de uma classe existente. “Vamos trocar tudo isso em miúdos”?

### **Exemplos:**

**Quadro 4.1 Exemplos da distinção entre classes e instâncias**

| Classe         | Objeto da Classe (Instância)                             |
|----------------|--|
| <b>Pessoa</b>  | Thiago, Dominik, Nicole, Manuela, Erika, Leo, Isabela... |
| <b>Animal</b>  | Cachorro, gato, macaco...                                |
| <b>Veículo</b> | Gol, HB20, Fiesta, Fusca, Ferrari...                     |

Fonte: Da autora (2014).

Quando estamos identificando os possíveis objetos ou as possíveis classes, estamos tratando do nosso poder de abstração, ou seja, da habilidade mental que permite aos seres humanos visualizar os problemas do mundo real com vários graus de detalhe.

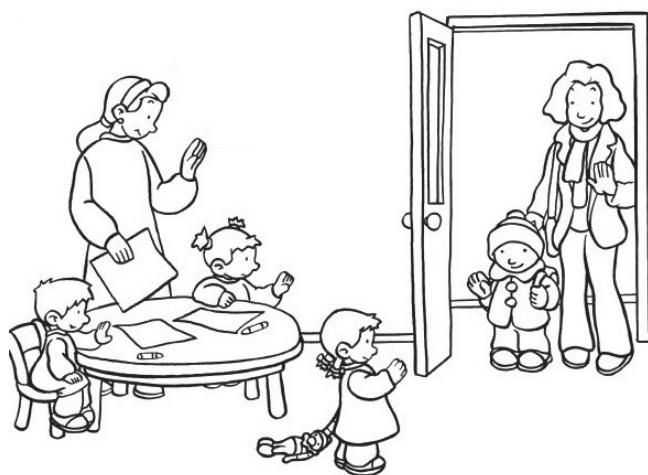
Podemos pensar no cenário em que temos de fazer o levantamento de requisitos para um novo projeto de sistema solicitado, seja por e-mail, telefone ou pessoalmente. E nesse momento nosso cliente nos apresentará suas necessidades, ou seja, seus desejos para um novo sistema e nós temos de pensar em uma solução em nível de sistema para atendê-lo.

E é importante salientar que nenhum objeto é uma ilha, eles podem ser relacionados a outros de forma direta ou indireta, forte ou fracamente. Segundo Mike O’Docherty (2005) os objetos podem ser relacionados por associação ou agregação.

A associação trata de uma conexão formal entre os objetos, sendo que não há uma dependência completa entre eles. Os objetos são parte de um grupo, porém não são dependentes um do outro.

**Exemplo:** Considerando uma sala de aula, com professores e alunos, todos estão unidos nesse ambiente, ou seja, “associados”, porém, um aluno pode deixar a sala de aula, esse aluno não estará mais no grupo, porém, o grupo continua com os demais envolvidos.

Figura 4.3 Um aluno deixa a sala de aula, mas a aula continua com os demais, podemos relacionar essa situação com uma associação



Fonte: Oliveira (2010).

A agregação une os objetos para criar um objeto maior, ela já estabelece uma dependência entre os objetos.

**Exemplo:** Considerando a estrutura física de uma sala de aula, a relação entre paredes e porta, afinal, não é possível entrar em uma sala ou sair dela sem passar pela porta, não é mesmo?

Figura 4.4 Espaço físico de uma sala de aula, representado uma “agregação”, visto a dependência dos objetos na estrutura física



Fonte: Archideaphoto/Shutterstock (2014).

### 2.2.3 Atributos

Atributos são todas as variáveis de um objeto. Descrevem as características dos objetos, definindo os dados que devem ser armazenados (BEZERRA, 2007). Quando colocamos um atributo na classe, devemos informar o tipo de visibilidade, nome do atributo, tipos de dados e se eles têm um valor inicial ou não.

#### Notações:

- **Visibilidade:** Refere-se ao escopo de acesso permitido para um membro de uma classe, definidas em pública (+), protegida (#), privada (-) [ver item 2.2.10].
- **Nome do atributo:** iniciar com letra minúscula (quando a palavra for composta, a segunda inicia com letra maiúscula).
- **Tipo de dado:** apresenta a espécie de informação que pode ser armazenada no atributo. É informado após os dois pontos (:).

#### Exemplos:

##### — **Classe UML:** <<visibilidade>> <<nomeAtributo>>: <<tipo de dado>>

```
#nomeCliente: String  
- tipoCliente: String
```

##### — **Implementação Java:** <<visibilidade>> <<tipo de dado>> <<nomeAtributo>>;

```
protected String nomeCliente;  
private String tipoCliente;
```



#### Para saber mais

Você lembra alguns tipos de dados?

Por exemplo, a linguagem de programação Java para fazer suas representações para o computador possui oito tipos primitivos (por darem origem a todos os outros), sendo eles: boolean, byte, char, double, float, int, long e short. E são divididos em:

- Tipos inteiros: byte, char, int, long e short.
- Tipos de ponto flutuante: double e float.
- Tipo boleano: boolean.

Nota: Em Java esses tipos primitivos também são palavras reservadas e você não pode, por exemplo, criar uma variável chamada double.

### 2.2.4 Operações ou métodos

A operação é a definição do comportamento de uma classe, porém, sua ação só ocorre quando essa operação é chamada por um objeto. A implementação de uma operação é chamada de **método**.

Notações:

- Visibilidade: Refere-se ao escopo de acesso permitido para um membro de uma classe, definidas em pública (+), protegida (#), privada (-) [ver item 2.2.10].
- Nome: identifica um recurso comportamental específico de uma classe de objeto, para ser eficaz, deverá ser o mais significativo e expressivo possível.
- Tipo de dado: apresenta a espécie de informação que pode ser armazenada no atributo.
- Tipo de retorno: é a saída da operação, podendo retornar um tipo de dado ou não possuir retorno e ser representado com a palavra reservada void.
- Lista de parâmetro: é uma lista dos atributos que juntos definem a entrada para uma operação.

**Exemplos:**

Classe UML: <<visibilidade>> <<nomeOperação (nome\_parametro : tipo\_dados)>>;<<tipo de retorno>>

```
- setNome(novoNome : String): void
- getName(): String
```

Implementação Java: <<tipo de acesso>> <<tipo de retorno>> <<nome>> <<(tipo\_parametro nome\_parametro)>> {Implementação do método}

```
/*
 * Método responsável por “setar” um novo nome para a variável nome.
 * @param novoNome será o valor que a variável nome receberá.
 */
```

```
public void setNome(String novoNome){
    nome= novoNome;
}
```

```
/*
 * Método responsável por retornar o valor contido na variável nome.
 */
```

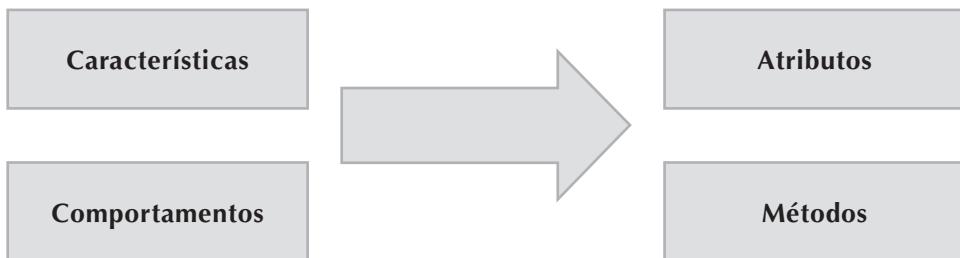
```
public String getName(){
    return nome;
}
```

## 2.2.5 Classe

A classe é um conjunto de objetos que possuem características e comportamentos semelhantes, elas representam algo do mundo real, um objeto, definindo suas características e comportamentos. As classes são implementadas por métodos e atributos

(FURLAN, 1998) tendo uma identidade própria e um estado, que pode ser alterado por um evento. As classes são os blocos de construção mais importante de qualquer sistema orientado a objetos. Traçando um pequeno paralelo entre o mundo real e o mundo computacional, temos:

**Figura 4.5 Blocos de construção das classes**



Fonte: Da autora (2014).

Para atribuição dos nomes das classes, é orientado seguir alguns padrões, seja por restrição da linguagem ou por questões de qualidade em que a orientação é que sejam adotados padrões e que sejam seguidos dentro das fábricas de software.

Vejam algumas orientações para criação das classes:

- └ a primeira letra deve ter a inicial maiúscula;
- └ o nome deve estar no singular;
- └ o nome não deve começar com números, mas pode conter números;
- └ o nome não pode ser acentuado. A linguagem Java até fará a compilação e não acusará erros, mas é aconselhável a não acentuação;
- └ em caso de nomes compostos, eles devem formar uma única palavra, em que a primeira e a segunda palavras têm a letra inicial maiúscula.

#### **Exemplo:**

- |                |                |
|----------------|----------------|
| └ PessoaFisica | └ PessoaJurica |
|----------------|----------------|

Na UML, as classes são representadas por retângulos que contêm seu nome, atributos e métodos. As classes podem ser classificadas como concretas ou abstratas.

A representação da classe é feita por um retângulo subdividido em três partes: Nome da Classe, Atributo e Operação.

#### **Exemplo:**

**Figura 4.6 Representação de classe**

|                |
|----------------|
| Nome da Classe |
| Atributos      |
| Operação       |

Fonte: Da autora (2014).

### Exemplo de uma classe:

**Figura 4.7 Representação da classe cliente**

| Cliente                               |
|---------------------------------------|
| + codigoCliente : int                 |
| # nomeCliente: String                 |
| # tipoCliente: String                 |
| + incluirCliente (nome: String): void |
| # deletarCliente (codigo: int): void  |
| - informarTipo (código: int): int     |

Fonte: Da autora (2014).



*Questões para reflexão*

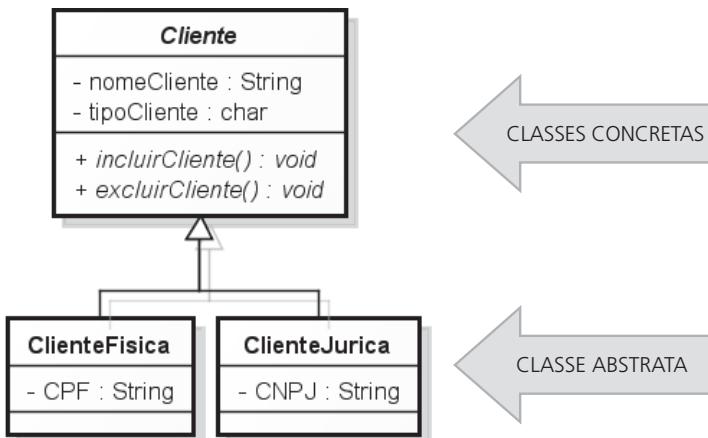
Se não existisse a Orientação a Objetos seria possível os três amigos criarem a UML?

#### 2.2.5.1 Classe abstrata

A classe abstrata, também conhecida como classe genérica, é desenvolvida para representar entidades e conceitos abstratos, não gera objetos, porque ela tem, pelo menos, uma operação abstrata definida. Uma operação também pode ser abstrata se ela não possuir implementação. A classe abstrata servirá como base para a criação de outras classes, sendo que ela é sempre uma superclasse que não possui instâncias. Ela define um modelo para uma funcionalidade e fornece uma implementação incompleta, a parte genérica dessa funcionalidade, que é compartilhada por um grupo de classes derivadas. Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.

As classes concretas implementam todos seus métodos e permitem a criação de instâncias, esse tipo de classe não possui métodos abstratos e podem ser classes derivadas de uma classe abstrata.

Figura 4.8 Exemplo de classes abstratas e concretas



Fonte: Da autora (2014).

Na UML, as classes abstratas se diferem das concretas por serem representadas com o nome em itálico. **Por exemplo:** A classe “*Cliente*” representada acima ficaria “*Cliente*”.

#### Exemplo de Classe abstrata em Java e C#:

```
abstract class NomeClasse{  
}
```

#### Exemplo de Operação abstrata em Java e C#:

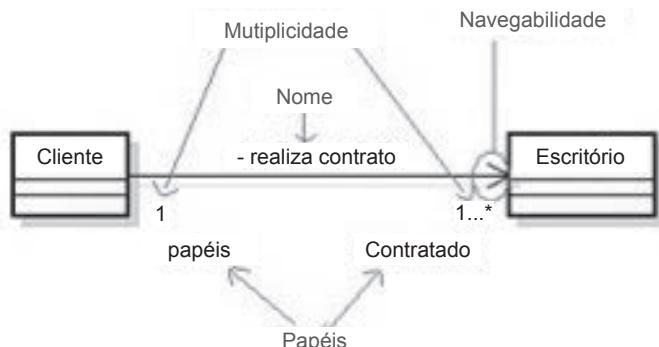
```
public abstract void nomeOperacao();
```

Para realizar a ligação entre as classes temos os relacionamentos: associação, agregação, composição, dependência e multiplicidade.

#### 2.2.6 Relacionamento entre classes

Relacionamentos são as ligações entre as classes. Os relacionamentos têm nome, sentido da leitura, naveabilidade, papéis, multiplicidade e tipo.

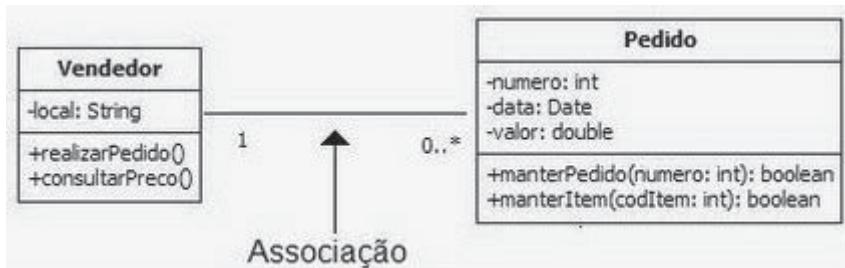
Figura 4.9 Exemplo de relacionamento



Fonte: Da autora (2014).

- ❑ **Associação:** esse relacionamento permite a comunicação de um objeto de uma classe com o objeto de outra classe. Ele é representado por uma linha sólida entre duas classes.

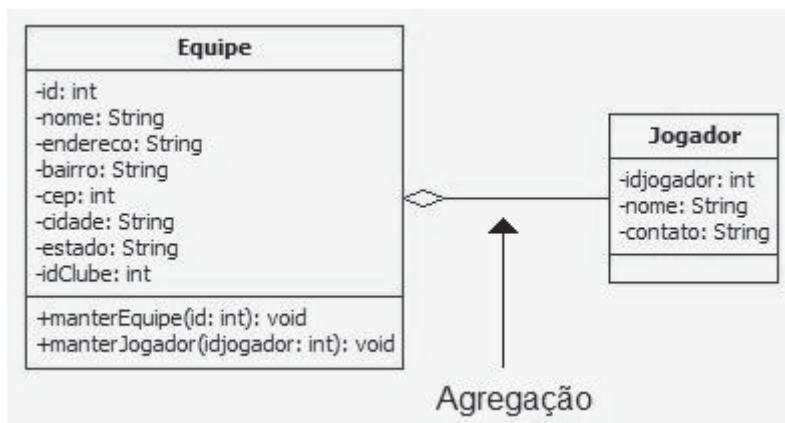
Figura 4.10 Modelo de associação



Fonte: Da autora (2014).

- ❑ **Agregação:** ela mostra que o objeto-parte faz parte do objeto-todo, e é representada por uma linha com losango sem preenchimento no parte da classe “dona”.

Figura 4.11 Modelo de agregação



Fonte: Da autora (2014).

- ❑ **Generalização:** também conhecida como herança ou classificação, é o relacionamento entre a classe-mãe e as classes-filhas.
- ❑ **Dependência:** esse relacionamento indica que os objetos de uma classe usam de serviço de objetos de outra classe. Ao mudar o objeto da classe independente, afetará o objeto da classe dependente.
- ❑ **Multiplicidade:** é a quantidade de instância de uma classe relacionada com a quantidade de instância da outra classe; depende de pressupostos e de

como são definidas as fronteiras de um problema. Ela pode ser infinita nos números inteiros e não negativos. A Tabela 4.1 representa alguns exemplos de multiplicidade.

**Tabela 4.1 Multiplicidade entre as classes**

| Multiplicidade  | Significado                       |
|-----------------|-----------------------------------|
| 0..1            | zero ou um                        |
| 1               | Somente um                        |
| 0...* ou 0...n  | Maior ou igual a zero             |
| * ou n          | Maior ou igual a zero             |
| 1...* ou 1...n  | Maior ou igual a 1                |
| 1...10          | Intervalo de 1 a 10               |
| 1...5 , 10...15 | Intervalo de 1 até 5 ou 10 até 15 |

Fonte: Da autora (2014).

## 2.2.7 Interfaces

A interface define a forma de comunicação entre duas entidades, podendo ser entendida como uma abstração, estabelecendo interação da entidade com o mundo exterior. Ela pode oferecer um serviço de tradução entre as entidades que não “falam a mesma língua”.

Neste item vamos citar as interfaces do usuário e interface física, nos quais interface física é utilizada para conectar componentes de hardware, já a interface do usuário é utilizada na interação homem-máquina.

**Figura 4.12 Exemplo de interface**



Fonte: Carvalho (2014).

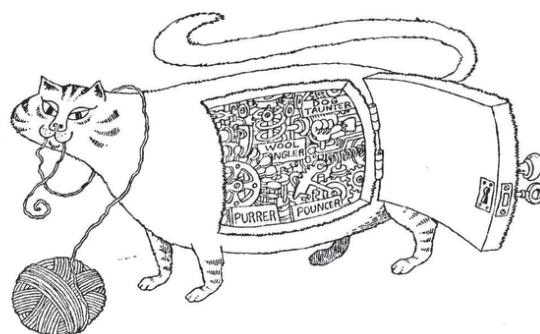
## 2.2.8 Encapsulamento

Podemos dizer que encapsular seria “esconder” do usuário os processos internos de um objeto, classe ou método. Dentro da linguagem orientada a objeto, podemos encapsular o estado do objeto, ou seja, cada objeto de uma classe pode ter limitadores de acesso. Esses limitadores são público (+), protegido (#), privado (-) e default(~) que servem para os atributos e operações (métodos).

- └ Público (public), quando acessado por qualquer classe, sendo que:
  - └ Uma classe pública ficará visível ou acessível a todas as classes e componentes do sistema.
  - └ Uma operação ou atributo público também passam a ficar visíveis ou acessíveis para as demais classes do sistema, porém, só serão acessados por meio de uma instância de um objeto dessa classe.
- └ Protegido (protected), quando os atributos e operações são visíveis ou acessíveis somente pela própria classe e subclasses [ver item 2.2.11].
- └ Privado (private), quando os atributos e operações são visíveis ou acessíveis somente pela sua classe.
- └ Default, quando os atributos e operações são visíveis ou acessíveis dentro de um pacote (package) do sistema.

O encapsulamento permite um maior controle e segurança às classes de objetos, tornando-o mais confiável.

**Figura 4.13 Exemplo de encapsulamento**



Fonte: Bezerra e Woszezenki (2014).



### Questões para reflexão

A visibilidade apresentada pode ser aplicada somente a atributos?

## 2.2.9 Herança

Herança é quando uma classe (classe-filha ou subclasse) utiliza atributos e/ou métodos de outra classe (classe-mãe ou superclasse). Esse conceito é eficiente em tornar o código-fonte, mas organizado e facilita quando alterações são necessárias.

Vimos no item 2.2.5 sobre classes o conceito de classe abstrata e concreta, você se lembra desse conceito? Ele é muito utilizado na herança. Vamos fazer um pequeno exercício para testar seu conhecimento.

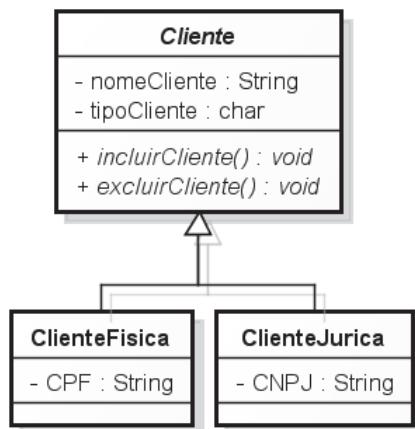


### Atividades de aprendizagem

1. Indique com V para verdadeiro e F para falso:
  - ( ) Classe abstrata é a superclasse (ou classe mãe).
  - ( ) Classes concretas são as classes-filhas.
  - ( ) Classe concreta permite a criação de instâncias.
  - ( ) Classes abstratas são desenvolvidas para representar entidades e conceitos abstratos.
2. Uma propriedade, operação ou atributo representado no diagrama de classes da UML, que poderá ser visto e usado apenas pela classe na qual foi declarado, bem como pelas suas classes descendentes, deve ser definido com visibilidade descrita por meio da palavra-chave:
  - a) Protected.
  - b) Public.
  - c) Local.
  - d) Package.
  - e) Private.

Outro conceito importante na herança é o de reusabilidade, que é a reutilização de componentes de software e diminuição do tempo de desenvolvimento. A reusabilidade facilita reaproveitar os códigos, objetos encapsulados, componentes e frameworks; também um dos conceitos abordados nas características da OO.

**Figura 4.14** Representa a herança da classe Cliente para as subclasses ClienteFisica e ClienteJuridica



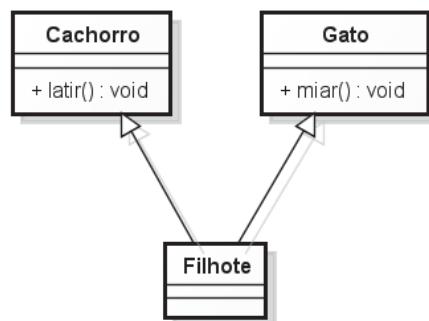
Fonte: Da autora (2014).

Também podemos ter o conceito de herança múltipla, isso ocorre quando temos de definir uma subclasse com mais de uma superclasse. Se pensarmos no nível do conceito, a herança múltipla é utilizada para modelar o mundo real de um modo mais preciso, mas, colocando em prática, ela pode levar a problemas na implementação pois nem todas as linguagens de programação orientadas a objetos têm base para a herança múltipla.

Vamos criar um exemplo para heranças múltiplas; nesse caso uma subclasse deve herdar características e comportamentos de duas ou mais classes (superclasses).

Imagine que você tenha uma classe para representar Filhote, porém, essa classe herda características e comportamento de duas classes já existentes Cachorro e Gato. Nesse caso a classe Filhote será uma especialização das classes generalizadas Cachorro e Gato.

**Figura 4.15** Exemplo de herança múltipla



Fonte: Da autora (2014).

**Exemplo de herança em Java:**

```
public class ClasseFilha extends ClassePai{
}
```

**Usando o exemplo da Figura 4.13 teremos:**

```
public class ClienteFisica extends Cliente{
}
```

**Exemplo de herança em C#:**

```
public class ClasseFilha : ClassePai{
}
```

**Usando o exemplo da Figura 4.13 teremos:**

```
public class ClienteFisica : Cliente{
}
```

**Questões para reflexão**

Vamos supor que você tenha uma classe Pai com um atributo Privado.

Pergunta: A classe-filha que herdar as suas características poderá visualizar esse atributo?

**2.2.10 Polimorfismo**

O polimorfismo é caracterizado quando duas ou mais classes têm métodos de mesmo nome, de forma que uma função possa utilizar um objeto de qualquer uma das classes polimórficas, sem necessidade de tratar de forma diferenciada conforme a classe do objeto.

**Para saber mais**

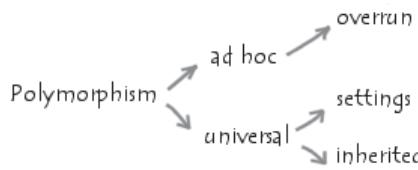
O termo *polimorfismo* tem origem grega e significa “muitas formas”? Vamos quebrar a palavra (*Poli* = muitas e *morphos* = formas).

Podemos definir como sendo um código que produz vários comportamentos, ou seja, esse código pode ser aplicado a diferentes classes.

Alguns dos padrões de projeto de software baseiam-se no uso de polimorfismo, como: abstract factory, composite, observer, strategy e template method.

Existem alguns tipos de polimorfismo: Universal (inclusão ou paramétrico) e ad-hoc (sobre carga — quando duas funções/métodos com o mesmo nome, mas assinaturas diferentes).

**Figura 4.16 Tipos de polimorfismo**



Fonte: Kioskea.net (2014).

O polimorfismo é a base para vários padrões de projeto, podemos destacar alguns:

- └ Abstract factory.
- └ Strategy.
- └ Composite.
- └ Template method.
- └ Observer.

## 2.2.11 CRUD

O CRUD é um termo utilizado em programação para representar as quatro operações básicas. São elas: criar (create), ler (read), atualizar (update) e excluir (delete).

O termo CRUD foi inicialmente popularizado por James Martin em seu livro **Managing the Data-base Environment** de 1983.

Podemos utilizar outros siglas para definir as mesmas operações sendo:

- └ ABCD: Add, Browse, Change e Delete
- └ BREAD: Browse, Read, Edit, Add e Delete
- └ VADE(R): View, Add, Delete, Edit (e Restore, para sistemas com processos transacionais)
- └ VEIA: Visualizar, Excluir, Inserir, Alterar

A matriz CRUD, ou matriz de interações, pode ser utilizada para verificar as relações entre funcionalidades (ou atividades) e entidades (ou tipos de objetos de dados) dentro de um sistema de informação.

Ela é construída de forma que as funcionalidades são listadas num dos seus eixos, e as entidades, no outro. As células de interseção denotam o tipo de interação existente, ou seja, mostram que entidade será afetada pela execução de determinada funcionalidade e explicita as propriedades CRUD para tal interseção. Portanto, cada uma das suas células descreve as ações que uma atividade exerce sobre o tipo de objeto de dados associado, que podem ser: Create (inclusão), Read (leitura), Update (atualização) e Delete (exclusão).

**Figura 4.17 Matriz CRUD**

| Entidades \ Funcionalidades | ENTI DADE A | ENTI DADE B | ENTI DADE C | ENTI DADE D | ENTI DADE E | ENTI DADE F | ENTI DADE G | ENTI DADE H | ENTI DADE I | ENTI DADE J | ENTI DADE K | ENTI DADE L | ENTI DADE M | ENTI DADE N | ENTI DADE O | ENTI DADE P | ENTI DADE Q | ENTI DADE R | ENTI DADE S | ENTI DADE T |      |
|-----------------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|------|
| FUNCIONALIDADE - 01         |             | R           | CR          |             |             | R           |             |             |             |             |             |             |             | R           |             |             |             |             |             |             |      |
| FUNCIONALIDADE - 02         |             |             |             | R           | RU          |             |             |             |             |             |             |             | CR          |             |             |             |             | RU          |             |             |      |
| FUNCIONALIDADE - 03         |             |             |             |             |             | RU          |             |             |             |             |             |             | CRUD        |             |             |             | RD          | R           | RUD         |             |      |
| FUNCIONALIDADE - 04         | RD          |             | R           |             |             |             |             |             |             | RU          |             |             |             | CR          |             |             | RD          | R           | RU          |             |      |
| FUNCIONALIDADE - 05         |             | CR          |             |             |             | RU          |             | R           |             |             |             |             |             | RU          |             |             |             | RU          |             |             |      |
| FUNCIONALIDADE - 06         |             |             |             |             |             |             | CR          |             | R           |             |             |             |             | RU          |             | CR          |             | RU          |             |             |      |
| FUNCIONALIDADE - 07         | R           |             |             |             |             |             |             |             | R           |             |             |             |             | RD          |             |             | RU          |             |             |             |      |
| FUNCIONALIDADE - 08         |             | R           | RU          |             |             |             |             |             |             |             |             |             |             | R           |             |             |             | R           |             |             |      |
| FUNCIONALIDADE - 09         |             |             |             | RD          | RD          |             |             |             |             |             |             |             | RU          |             |             |             |             | CRUD        | R           | RU          |      |
| FUNCIONALIDADE - 10         | RU          |             |             |             |             | CR          |             | CR          |             |             |             |             | R           |             |             |             | R           |             |             |             |      |
| FUNCIONALIDADE - 11         |             | R           | R           |             |             |             |             |             |             |             |             |             |             | CRUD        |             |             |             |             | RU          |             |      |
| FUNCIONALIDADE - 12         |             |             |             | RU          | R           |             |             |             |             |             |             |             |             | RD          |             |             |             |             |             | RU          |      |
| FUNCIONALIDADE - 13         |             |             |             |             | RUD         |             | R           |             |             | RU          |             |             |             |             |             |             |             |             |             |             | RU   |
| FUNCIONALIDADE - 14         |             |             |             |             |             |             | R           |             | R           |             |             |             |             |             |             | CR          | RU          |             | CR          |             |      |
| FUNCIONALIDADE - 15         |             |             |             |             |             |             |             | RD          |             |             |             |             |             | R           |             |             |             |             |             | R           |      |
| FUNCIONALIDADE - 16         |             | CRUD        | RD          |             |             |             |             |             |             |             |             |             |             |             | RU          |             |             |             |             |             |      |
| FUNCIONALIDADE - 17         |             |             |             | CRUD        |             | CRUD        |             |             |             |             |             |             |             |             |             |             | CRD         |             |             |             |      |
| FUNCIONALIDADE - 18         | CR          |             |             |             |             |             |             |             | RD          |             | R           |             |             | RU          |             |             |             | RUD         | R           |             |      |
| FUNCIONALIDADE - 19         |             |             |             |             |             |             |             |             | RD          |             | R           |             |             |             | R           |             |             |             |             |             | CRUD |
| FUNCIONALIDADE - 20         | R           |             | R           | RD          |             |             |             |             | RD          |             |             |             |             |             |             |             |             |             |             |             |      |

Fonte: Pluriverso (2014).

## 2.2.12 Diagrama de classe

Diagramas são representações visuais de uma estrutura; o diagrama de classe é uma representação das estruturas e relacionamentos das classes. É uma modelagem importante para o desenvolvimento do software. O diagrama de classes é o mais importante diagrama da UML e a partir desse diagrama outros diagramas são elaborados.

Segundo Fowler e Kendall (2000, p. 57), “um diagrama de classes descreve os tipos de objetos no sistema e os vários tipos de relacionamentos estáticos que existem entre eles”. O diagrama de classe é composto pelas classes e seus relacionamentos, que podem ser associação, composição e dependência.

Para organizar o diagrama de classes é comum os analistas adotarem um padrão para nomeá-las.

Podemos classificar as classes em classe de negócio, classe de controle, classe de interface:

- Na classe de negócio ou de informações lógicas, seus objetos possuem dados que ficam disponíveis por longos períodos de tempo.  
Exemplo: Classe Cliente.
- As classes de controle possuem os processos/algoritmos; elas incluem funcionalidades que não podem ser atribuídas às classes de interface ou às classes de negócio. Ex.: Cotação.
- Classes de Interface possuem os objetos técnicos. Incluem funcionalidades que são diretamente dependentes do ambiente de sistema. Essa classe altera as entradas do ator nos eventos do sistema e apresenta as saídas. Ex.: Menu.



## Questões para reflexão

A orientação a objetos (OO) não tira a necessidade de implementar os sistemas, e nem está relacionada apenas à fase de implementação. Ela também não garante a reutilização dos objetos, mas oferece mecanismos para que ocorra e sempre será função do desenvolvedor garantir isso. Com base nisso, você consegue identificar vantagens e desvantagens na OO?

### 2.2.13 Resolução de um estudo de caso

Vamos colocar em prática o que vimos até o momento. Neste tópico teremos um estudo de caso, em que vamos juntos encontrar um resultado. Lembrando que cada analista que verificar a atividade poderá encontrar respostas diferentes. Esse estudo de caso trata de um sistema de matrícula.

A escola “O Sabido” deseja informatizar seu sistema de matrícula. Essa escola oferece vários cursos. O coordenador do curso define quais as disciplinas a serem oferecidas no semestre. Cada curso tem várias disciplinas, e cada disciplina pode ter várias turmas. Cada disciplina tem um professor.

O aluno solicita na matrícula o curso e o semestre a estudar. O sistema verifica as informações de matrícula e, se estiver correto, o sistema matricula o aluno, caso tenha algo errado, o sistema gera um erro e solicita novo cadastro. O aluno pode se matricular em mais de um curso.

O sistema deve informar dados como o nome e a titulação do professor, sala e o turno das turmas, o nome e a matrícula do aluno e nome da disciplina.

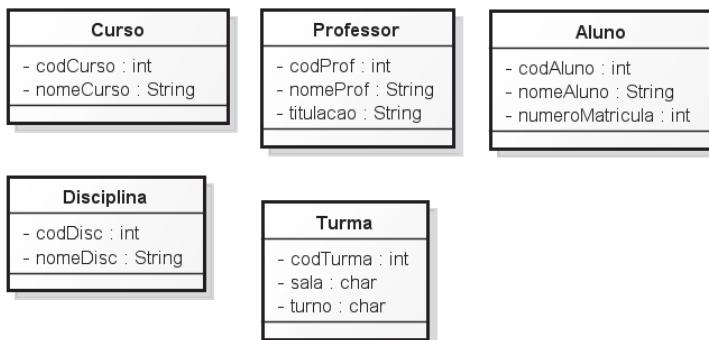
Com base nessa descrição, vamos iniciar buscando pelas classes. De início podemos citar as classes: aluno, disciplina, turma, professor e curso. Você encontrou mais alguma?

**Figura 4.18 Classes encontradas no modelo de estudo de caso**



Fonte: Da autora (2014).

As classes a seguir possuem atributos. Agora é sua vez, olhando para essas classes e o estudo de caso, quais atributos você consegue identificar?

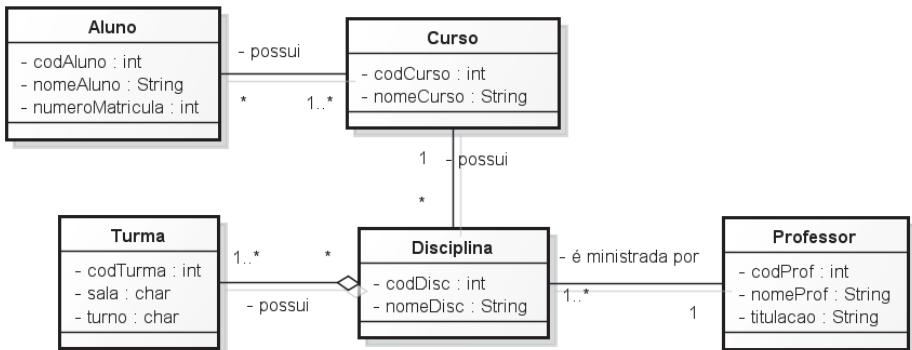
**Figura 4.19 Classes com atributos**

Fonte: Da autora (2014).

Verifique a multiplicidade no texto:

- └ “cada disciplina tem um professor”,
- └ “o aluno pode se matricular em mais de um curso”.

Vamos verificar como fica no diagrama?

**Figura 4.20 Diagrama de classe do sistema de matrícula**

Fonte: Da autora (2014).

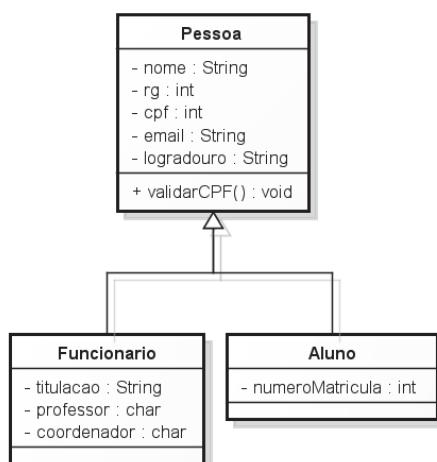
Como seria a leitura desse diagrama? O curso possui uma ou várias disciplinas. Uma disciplina possui uma ou mais turmas. Continue a leitura do diagrama.

E você achou outra solução para esse estudo de caso? Onde colocaria as informações sobre o coordenador, teria uma classe coordenador ou um atributo na classe professor que identificaria o atual coordenador do curso? Vamos fazer um teste para ver como esse diagrama ficaria se fizéssemos uma pequena alteração no texto? Vamos lá...

Para o aluno, professor e coordenador, devemos gravar dados como seu nome, telefone fixo e telefone móvel, endereço para correspondência, e-mail, RG e CPF. Os documentos devem ser validados. Para o professor e coordenador temos de gravar a titulação. O coordenador também pode dar aulas.

Podemos utilizar aqui o conceito de herança? Como ficaria no diagrama? Vamos a mais um exemplo.

**Figura 4.21 Utilização de herança no modelo de sistema de matrícula**



Fonte: Da autora (2014).



### Questões para reflexão

Verifiquem que mudar um detalhe que aparentemente é simples pode fazer uma grande diferença no sistema pronto, por isso é importante dar uma atenção maior na fase de análise.

#### 2.2.14 Estudo de caso — Advocacia

Nesta unidade, você aprendeu o que é uma classe, seus atributos, métodos e o relacionamento entre eles. Agora é hora de colocar em prática o que aprendeu. Neste estudo de caso, você terá a oportunidade de localizar as classes, fazer o relacionamento, informar seus atributos e métodos. Então, mãos à obra...

O advogado João Carlos Saranza solicita um sistema para controle dos clientes do seu escritório. Ele comenta que tem clientes como pessoa física e também atende a pessoas jurídicas. Para o cadastro de clientes é necessário anotar dados como nome, endereço, telefones. Para a pessoa jurídica é necessário guardar o CNPJ e o nome do responsável. Para a pessoa física é necessário documentos pessoais como RG e CPF e estado civil.

O advogado informa que costuma registrar vários telefones do cliente, como o da casa, celular, trabalho e ramal. Para o endereço também é necessário registrar o da casa e do trabalho.

Vinculados aos clientes estão os contratos, cada caso/ação em que o advogado atua, é necessário um novo contrato.

**Exemplo:** Se a cliente Valeska Houser o contratou para um dívida trabalhista, ele faz um contrato para essa ação. Caso a mesma cliente precise do serviço em uma ação de pensão, ele executa outro contrato com ela. Nesse cadastro é necessário informar o status da ação, ou seja, se está ativa, cancelada ou encerrada. Ele cita que algumas ações são vinculadas a vários clientes.

Seu escritório possui várias equipes de advogados, separados por tipo de ação, sendo trabalhista, civil, família, previdenciário e ambiental.

Caro aluno, agora é a sua vez. **Você** é o analista, então, inicie identificando:

- └ as classes;
- └ os atributos;
- └ os métodos;
- └ os relacionamentos entre as classes.

Lembrando que durante a análise pode ser que você identifique melhorias para o sistema.

Agora que já lemos muito, vamos realizar mais atividades?



## Atividades de aprendizagem

1. Você lembra quais são as características da orientação a objeto? Das citadas abaixo, qual é a correta?
  - a) **Confiabilidade:** é uma linguagem visual utilizada para modelar sistemas orientados a objetos.
  - a) **Reusabilidade:** reutilização dos componentes do sistema.
  - a) **Extensibilidade:** visa à facilidade em dar manutenção ou realizar alguma customização.
  - a) **Manutenabilidade:** permite um maior controle e segurança às classes.
  - a) **Todas estão incorretas.**
2. Relacionando as colunas

|                     |   |
|---------------------|---|
| a) UML              | 1. reutilização dos componentes do sistema.                                     |
| b) Manutenibilidade | 2. permite um maior controle e segurança às classes.                            |
| c) Confiabilidade   | 3. visa à facilidade em dar manutenção ou realizar alguma customização.         |
| d) Reusabilidade    | 4. é uma linguagem visual utilizada para modelar sistemas orientados a objetos. |
| e) Extensibilidade  | 5. é a medida da facilidade em se adicionar novas funcionalidades.              |

Assinale a alternativa que traz a combinação correta.

- a) A4, B3, C5, D1, E2.
- b) A1, B3, C2, D4, E5.
- c) A4, B3, C2, D1, E5.
- d) A4, B5, C2, D1, E3.
- e) A2, B3, C4, D1, E5.

### **2.2.15 Estudo de caso — Revenda de automóveis**

Vamos trabalhar com mais um estudo de caso para refinar seus conhecimentos?

Uma oficina de automóveis quer modificar a forma de gerenciar seus registros de serviços de manutenção.

A responsável pela oficina deseja criar um software para gerenciar esses itens, e para realizar o desenvolvimento contratou um aluno do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, o Marcos Firmino.

Esse aluno, lembrando-se dos ensinamentos da disciplina de análise, disse que inicialmente precisaria fazer um estudo do sistema existente e verificar quais itens precisam ser ajustados. Firmino informou que desenvolverá o sistema sendo que inicialmente fará uma boa documentação e as validações com a cliente antes de realizar as implementações.

O aluno cria algumas perguntas para apoiar no levantamento de requisitos e a partir delas descreve o seguinte cenário:

*A oficina “Seu Carro Novo De Novo”, solicita um novo sistema para controle de manutenção de serviços. A proprietária, a sra. Anacleia Giacomo, apresenta algumas funcionalidades que considera essenciais para o novo sistema, em que:*

- ↳ *o sistema deve manter informações sobre os carros e serviços prestados (como troca de peças, revisão etc.);*
- ↳ *o sistema deve gerar relatórios de controle de serviços, cadastro de clientes e funcionários e cadastro de carros.*

*Sobre os funcionários, é necessário manter informações como:*

- ↳ *nome, endereço, telefone, data de admissão, número de registro, documentos pessoais como RG, CPF, número do PIS..., setor de lotação, cargo/ função e no caso de ex-funcionários também é necessário armazenar a data de desligamento.*

*Para o registro do cliente, temos de armazenar:*

- ↳ *nome, endereço, telefone, histórico de serviços, documentos pessoais (RG e CPF), local de trabalho e data de registro.*

Para o histórico de serviço, é preciso guardar informações como o cliente, o carro, a data de realização, qual funcionário realizou o serviço e a descrição do que foi feito no veículo.

É importante informar que vários cadastros não podem ter dados excluídos, como o cadastro de cliente, carro, funcionário e histórico. Mas em caso de lançamento incorreto a gerente poderá cancelá-lo.

Para todos os controles listados ao sistema deverá permitir: incluir, alterar e consultar dados.

Atualmente a oficina trabalha com carros das montadoras Ford, Fiat, GM, VW, Toyota e Mitsubishi, mas a sra. Anaclesia já adianta que quer ampliar as marcas a serem atendidas.

Para o controle de carro é necessário conter dados como:

   montadora, o ano de fabricação, cor, modelo, placa e nome do proprietário.

Existem vários tipos de serviços prestados, o cliente pode solicitar desde uma revisão completa ou parcial nos veículos, o cliente também pode solicitar a parte sobre “estética” do veículo, com o serviço do “martelinho de ouro”, instalação de som, parte elétrica ou conserto em geral.

Um serviço que a oficina oferece é dar consultoria para o cliente, quando ele resolve trocar ou comprar um novo carro. Eles verificam, entre outros, qual modelo (sedã, utilitário, hatch, wagon e picape) é o mais indicado para o cliente e, no caso de compra de um usado, eles acompanham o cliente para avaliar o veículo.

Após ouvir todas as necessidades da cliente, Firmino sugeriu que o sistema tivesse seus controles realizados via web para que a sra. Anaclesia pudesse acessar o sistema mesmo em uma viagem.

O aluno finaliza os registros e inicia a construção da modelagem do sistema. Com base em suas anotações ele solicita a ajuda para preencher algumas questões.



## Atividades de aprendizagem

- Podemos identificar algumas classes ao ler o estudo de caso. Marque quais as alternativas são referentes a classes:

|  |   |
|--|---|
| ( <input type="checkbox"/> ) funcionário | ( <input type="checkbox"/> ) carro            |
| ( <input type="checkbox"/> ) nome        | ( <input type="checkbox"/> ) data de admissão |
| ( <input type="checkbox"/> ) cliente     | ( <input type="checkbox"/> ) telefone         |

- Relacione as classes com seus atributos

|                 |  |
|-----------------|--|
| (A) funcionário | ( <input type="checkbox"/> ) montadora   |
| (B) cliente     | ( <input type="checkbox"/> ) nome, endereço, telefone, data de registro                    |
| (C) carro       | ( <input type="checkbox"/> ) nome, endereço, telefone, data de registro, local de trabalho |

3. Verificando as classes funcionário e cliente, podemos identificar alguns atributos em comum, pensando em facilitar a manutenção do sistema, o aluno resolve trabalhar com a característica de herança. Para isso ele identifica os atributos em comum, montando a tabela abaixo. Veja como ficariam as classes:

| <b>Classe Funcionário</b> |                  | <b>Classe Cliente</b> |                   |
|---------------------------|------------------|-----------------------|-------------------|
| Nome                      | Endereço         | Nome                  | Endereço          |
| Telefone                  | Data de admissão | Telefone              | Histórico serviço |
| Número registro           | RG               | RG                    | CPF               |
| CPF                       | PIS              | Local de trabalho     | Data registro     |
| Lotação                   | Cargo            |                       |                   |
| Data desligamento         |                  |                       |                   |

Verificamos que os atributos nome, endereço, telefone, RG, CPF se repetem nas duas classes nesse caso podemos criar a classe “Pessoa” como a superclasse, e Funcionário e Cliente ficam como subclasses.

Como ficaria essa separação?

| <b>Pessoa (superclasse)</b> |
|-----------------------------|
| Nome                        |
| Endereço                    |
| Telefone                    |
| RG                          |
| CPF                         |



### *Questões para reflexão*

Você sabe qual das características da orientação a objetos é a base para os vários padrões de projeto?

***Fique ligado!***

Caro aluno, vamos lembrar aqui as principais questões abordadas nesta unidade? Foram elas:

- conceitos de modelagem de sistemas, tanto estruturada quanto orientada a objetos (OO);
- dentro da estruturada, abordamos o DER e o DFD;
- dentro da OO, abordamos suas principais características e alguns conceitos fundamentais;
- a evolução das linguagens orientadas a objetos;
- alguns diagramas da UML;
- alguns estudos de caso para trabalhar e despertar sua capacidade de abstração por meio de uma situação problema.

***Para concluir o estudo da unidade***

Vimos nesta unidade a modelagem estruturada e orientada a objeto, conhecemos sua história e evolução. Descrevemos sobre seus diagramas, colocando uma ênfase nos diagramas de fluxo de dados e entidade relacionamento no item sobre modelagem estruturada e na modelagem de orientação a objeto. Abordamos o diagrama de classe.

Aprendemos sobre os tipos de visibilidade (público, privado, protegido), vimos as características da orientação a objeto como reusabilidade, objetos, atributos, estados, interações, classes, confiabilidade, reusabilidade, manutenibilidade, extensibilidade, e como são importantes os conceitos de polimorfismo, herança e encapsulamento.

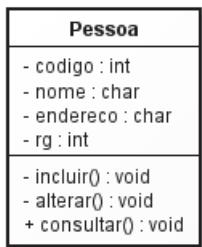
E tivemos a oportunidade de resolver atividades para colocar em prática o que aprendemos.



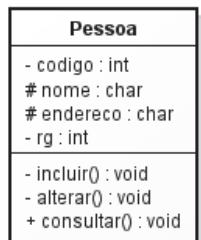
## Atividades de aprendizagem da unidade

1. A modelagem de dados é um processo para a criação de um software. Segundo Rumbaugh (1994), um modelo completo de sistema é composto por submodelos que expressam visões diferentes da mesma realidade. Na visão de Rumbaugh, quais seriam esses submodelos?
  - a) Visão de objeto, visão dinâmica, visão funcional.
  - b) Visão de objeto, visão de atributos, visão dinâmica.
  - c) Visão de classe, visão dinâmica, visão atributo.
  - d) Visão de objeto, visão classe, visão funcional.
  - e) Visão de classe, visão dinâmica, visão estática.
2. Em orientação um objeto muito utilizado é o conceito de herança, que é quando uma classe (classe-filha ou subclasse) utiliza atributos e/ou métodos de outra classe (classe-mãe ou superclasse). Geralmente o usuário de sistema não percebe esse conceito, você conhece algum sistema que se utiliza dele?
3. A linguagem orientada a objeto teve seu início na década de 1960, antes mesmo da modelagem orientada a objeto. Você sabe o motivo que fez com que surgisse a modelagem orientada a objeto?
4. Uma linguagem visual utilizada para modelar sistemas orientados a objetos é a UML — unified modeling language. UML tem como objetivo descrever um sistema, usando diagramas orientados a objetos. Durante esta unidade falamos muito sobre um dos diagramas da UML. Você saberia informar quais são os diagramas da UML?
5. No diagrama de classe identificamos as visibilidades dos atributos e métodos. São elas pública, protegida e privada. Descreva sobre cada uma e identifique os símbolos.
6. Pelo texto abaixo, identifique qual a alternativa correta:  
Temos uma classe pessoa, que possui os atributos endereço e telefone que são protegidos, um atributo nome que pode ser visualizado por outras classes, nenhuma outra classe pode ter acesso ao atributo RG. Essa classe possui métodos como incluir, alterar e consultar.

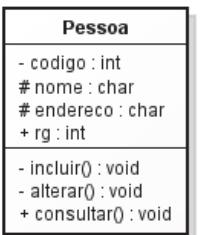
a)



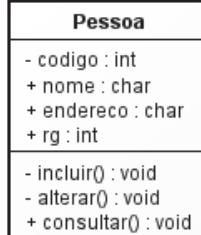
b)



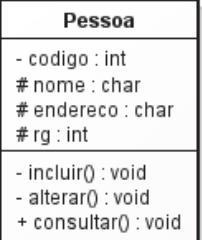
c)



d)



e)



---

## Referências

BEZERRA, Eduardo. **Princípios da análise e projeto de sistemas com UML**. Rio de Janeiro: Elsevier, 2007.

BEZERRA, E. A.; WOSZEZENKI, C. Encapsulamento. **Linguagem de Programação C++**. Universidade Federal de Santa Catarina. Departamento de Engenharia Elétrica, CTC. Disponível em: <[http://gse.ufsc.br/~bezerra/disciplinas/cpp/aulas/dia2\\_4.html](http://gse.ufsc.br/~bezerra/disciplinas/cpp/aulas/dia2_4.html)>. Acesso em: 16 jul. 2014.

CARVALHO, U. W. O que significa “Interface”? **Tecla Sap** — sua língua falada em inglês. 2014. Disponível em: <<http://www.teclasap.com.br/o-que-significa-interface/>>. Acesso em: 16 jul. 2014.

CHEN, Peter. **Active conceptual modeling of learning**: Next Generation Learning-Base System Development. Editado por Leah Y. Wong. Springer, 2007.

COAD, Peter; YORDON, Edward. **Análise baseada em objetos**. 2. ed. Rio de Janeiro: Campus, 1998.

O'DOCHERTY, Mike. **Objwect-oriented analysis and design**. Londres: John Wiley e sons, 2005.

FIGUEIREDO, Marcos Leandro; SOARES, Hélio Rubens. **Comparação entre a modelagem orientada a objeto e a modelagem estruturada relacional**. Centro Universitário do Triângulo — UNITRI — Uberlândia — MG. Disponível em: <<https://www.yumpu.com/pt/document/view/12764245/comparacao-entre-a-modelagem-orientada-a-unicaldas-online>>. Acesso em: 24 maio. 2014.

FOWLER, Martin; SCOTT, Kendal. **UML essencial**. Porto Alegre: Bookman, 2000.

FURLAN, José Davi. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

GUEDES, G.T. **UML 2: Uma abordagem prática**. 2. ed. São Paulo: Novatec, 2011.

KIOSKEA.NET. POO — O polimorfismo. 2014. Disponível em: <<http://pt.kioskea.net/contents/416-poo-o-polimorfismo>>. Acesso em: 16 jul. 2014.

MARTIN, James. **Managing the data-base environment**. São Paulo: Pearson Education, 1983.

MOREIRA, J. R. Análise de programação. Disponível em: <<http://dc522.4shared.com/doc/WJN2FOZz/preview.html>>. Acesso em: 16 jul. 2014.

OLIVEIRA, Patrícia. Desenho Regrinhas de Convivência. **Blog Ricos e Desenhos**. 2010. Disponível em: <<http://www.riscosedesenhos.com.br/2010/09/17/desenhos-regrinhas-de-convivencia/>>. Acesso em: 16 jul. 2014.

PLURIVERSO. Matriz de Interações (ou Matriz CRUD). 2014. Disponível em: <<http://pluriverso.com.br/software/matriz-de-interac%C3%B5es-ou-matriz-crud>>. Acesso em: 16 jul. 2014.

RUMBAUGH, James et al. **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Campus, 1994.



unopar

ISBN 978-85-68075-89-0



9 788568 075890 >